Adobe

Business Activity Monitoring Server Reference

Adobe® LiveCycle® ES

Update 1

July 2008

Contents

1 Introduction

This document provides detailed descriptions of each of the objects and features of Adobe® LiveCycle® ES (Enterprise Suite) Update 1 (8.2) Business Activity Monitoring. The specific topics covered include:

- "Access Filters" on page 15 describes the filters that restrict the data a user sees without having to define a new view or cube for each user.
- "Agents" on page 23 describes the objects that know how to receive or retrieve information from external sources.
- "Alerts" on page 27 describes the notifications of exceptional events sent to users or external systems.
- "Business Activities" on page 41 describes the container objects that collect the scenarios that identify exceptional business conditions.
- "Context" on page 43 describes context data, how it flows into the system, and how to create it.
- "Cubes" on page 50 describes multidimensional data cubes and how to create them.
- "Data Types" on page 55 describes the supported SQL-99 data-types and their semantics.
- "Dimensions" on page 69 describes dimensions and levels for use by cubes.
- "Events" on page 76 describes event data, how it flows into the system, and how to create it.
- "Flat Files" on page 81 describes how the system uses text files to source event data.
- "Formulas" on page 108 describes how to construct formulas in the Business Activity Monitoring.
- "Functions" on page 114 describes C-SQL functions that may appear in commands and rule formulas.
- "HTTP Post" on page 197 how to use HTTP to post events to an event stream.
- "Java Messaging Service (JMS)" on page 204 describes how the system uses JMS to source event data.
- "JDBC" on page 216 describes how the system uses JDBC interfaces to retrieve context data, receive event data, and to allow other Java applications to access the business views in memory.
- "JDBC Access to View Data" on page 228 describes the application programming interface (API) that allows JDBC 2.0 applications to retrieve data from a view, and to retrieve the metadata that describes the views in the installation.
- "Objects" on page 243 describes the details that all Business Activity Monitoring objects have in common, including name, optional description, and status.
- "Operators and Constants" on page 249 describes the supported operators and constants.
- "Permissions" on page 253 describes the controls that identify which users may access, create, and edit Business Activity Monitoring objects and user accounts.
- "Processes" on page 267 describes how Business Activity Monitoring uses and presents business process diagrams and statistics.
- "Query Windows" on page 272 describes query windows, which are sets of rows used when making calculations regarding the current event window.
- "Reportlets" on page 290 describes objects that provide information about an event that puts the event into context.
- "Roles" on page 296 describes how to use roles to assign permissions to a set of users.
- "Rules" on page 299 describes the objects that analyze business views looking for metrics that meet specific conditions.

- "Scenarios" on page 318 describes the collections of rules, alerts, and reportlets that identify exceptional business conditions in a business view.
- "SELECT" on page 321 describes C-SQL select statements that manage information in the Business Activity Monitoring.
- "TIBCO Rendezvous" on page 337 describes how the system uses TIBCO Rendezvous for event data.
- "Users" on page 332 describes the accounts by which each user is *known* to the system.
- "User-Defined Functions" on page 346 describes user-defined functions (UDFs) for use in formulas.
- "Views" on page 350 describes the data models that provide a real-time picture of a business activity.
- "Web Services" on page 362 describes how the system uses Web services to retrieve context data.
- "XML/XSD" on page 375 describes how to create Business Activity Monitoring objects with XML.
- "Glossary" on page 390 define common terms used throughout Business Activity Monitoring.

2 Access Filters

Access filters allow different users to see different rows of the same view or cube, depending on the criteria specified in the filter. These filters restrict the data a user sees without having to define a new view or cube for each user. For example, consider this view of total sales by region:

```
Total Sales Region
----------- -------
  763000.00 West
  489500.00 Central
  522950.00 South
  650740.00 East
```

By defining an access filter that says, for example, "OrderTotals.Region=Employees.Region", you can limit users to see only the rows that apply to their business region. As such, a user from the Central region looking at the view would see:

```
Total Sales Region
----------- -------
  489500.00 Central
```

The filters are logical expressions similar to the Where Clause of a view definition. (See "Access Filter Conditions" on page 16 for a complete description.)

Access filters are defined on a view-by-view and cube-by-cube basis, and are applied to users and roles having Filtered/Read-Only permission on the view or cube. For details about how access filters work, see "Access Filter Behavior and Restrictions" on page 18.

Applying access filters to a view or cube requires that you first create the filter, and then assign it to users or roles, as described in these sections:

- "Creating a View Access Filter" on page 19
- "Creating a Cube Access Filter" on page 20
- "Assigning an Access Filter to Users and Roles" on page 21

For detailed information about views, see "Views" on page 350; for cubes, see "Cubes" on page 50.

Access Filter Conditions

Access filter conditions are logical expressions that are applied to each row in the view, or dimension level in a cube. A user looking at the view or cube sees those rows where the expression evaluated to true. At a minimum, each filter should contain some condition that evaluates data found in the view or cube. For example, the following simple condition shows only the rows in OrderTotals that are in the *East* business region:

```
OrderTotals.Region='East'
```

Naming Users

The filter previously described must be assigned to each user or role in the East region to limit their access. A more powerful expression is one that names the users. The CURRENT_USER() function returns the login name of the user looking at the view. You can include that function in the filter condition to apply the filter to specific users. For example, this condition also identifies two users, and as such, only these two users would see the results for the East region:

```
OrderTotals.Region='East' AND
(CURRENT_USER()='Skyler' OR CURRENT_USER()='Nina')
```

Note: Access filters are logical expressions that can include Boolean operators (AND, OR, and NOT), and can use parentheses for grouping.

A limitation of the two examples above is that they have literal values hard-coded into the expressions: the region name and the user names. Using literals is problematic because it requires you to edit the filters whenever the names change. Further, you would need one for each region.

A more powerful expression is one that can be applied to all users by dynamically retrieving information about the user and applying it to the view.

Context Filters

In addition to the current view, access filters can retrieve data from a context table. If you define a context table that contains information about the users, you can compare that information to the data in the view to create a dynamic context filter. Consider this filter that uses an Employees context table:

```
OrderTotals.Region=Employees.Region AND
CURRENT_USER()=Employees.User_Name
```

Now you can apply the filter to many users and roles, and only those users assigned to the same business region as the data will see the data.

Note: To use dynamic look-ups you must provide the information in an external context table.

Users as Context

Context tables usually support events by providing additional information about the event. When used in an access filter, a context table provides information that supports the filter: namely, information about the current user. As such, a “users” context table must have at least one column that contains the user name that matches the login name that the user uses to log in to Business Activity Monitoring.

Be aware that CURRENT_USER() returns the user’s login name as defined in Business Activity Monitoring, in the same character case, and as it appears in the BAM Workbench. As such, it is important that the character case match exactly. (Note that some DBMS provide case-insensitive compares, so this might not be an issue to your installation.) To avoid mismatches, you might want to enter the names in the context table in a single case, and then use UPPER() or LOWER() in the filter expression, such as the following:

```
UPPER(CURRENT_USER())=Employees.User_Name
```

Note: You cannot use UPPER() or LOWER() on the reference to the Employees context table. See "Context Column Limitations in Queries" on page 46 for details.

Similarly, all text columns referenced in a filter need to be aware of case issues.

To use a context table in an access filter, add the table to the filter's workset when defining the filter. See "Creating a View Access Filter" on page 19 for details.

Summary

In summary, a "users" context table must include:

- One row for each user that will be assigned a Filtered/Read-Only access permission. If the user is not found in the context table, the filter will most likely fail to find any rows for that user.
- At least one text column that contains the user login name. If the DBMS provides case-sensitive matches, enter the names exactly as defined in Business Activity Monitoring, or at least with the same characters in the one text-case if you plan to use UPPER() or LOWER() in your filters.
- One column for each reference in the filter, and the data types must match. For character values, the strings in the view must exactly match the strings in the context table.

Also be aware that if the context table data is cached, the filters can fail if the user data is not in the cache. In other words, if you add a user to the database, you might also want to invalidate the context table cache before the user attempts to look at filtered views or cubes. If the user is not found in the context, the filter returns false.

Access Filter Behavior and Restrictions

Access filters are applied only when a user with Filtered/Read-Only permission on a view looks at or requests data from the view, or defines a new view on top of a such a view. The filters do not affect users or roles with Read-Only or Read-Write permission on the view, nor do they apply to users receiving reportlets sent as an attachment to alert notifications.

Specifically:

- The default access permission to the classes of View and Cube objects is No Access for all new users.

  Before any user can see the results, they must be assigned — directly or as a member of a role — Read-Only or Read-Write permission on the classes of View and Cube objects (all views and cubes) at least Filtered/Read-Only permission on the specific view.

- When a user is assigned multiple access filters to the same view or cube — perhaps as the result of being a member of multiple roles each with assigned filters — the user sees those rows where *any* of the filters is true for the row.

  For example, one filter might restrict a user to see on “West” region data, but another might allow the user to see all results for a specific family of products. The result is that the user will see all results for the family, regardless of region.

- Reportlets always include all data from the view that they reference, regardless of any access filters associated with the view.

  Users who receive reportlets as part of alert notifications always see the entire view referenced by the reportlet.

- When a user with filtered access to a view creates a new view on top of the filtered view, the new view inherits that user’s filtered results, but not the filter definition.

  Subsequently, anyone else looking at the derived view sees the results as filtered for the creating user. For example, if Sklyer can see only Total Sales from the “West” region, and he creates a new view called WrapUp derived from the Total Sales view, anyone else with permission to look at WrapUp sees the data for the West region, regardless of their own access permission to Total Sales.

- Similarly, when a user with filtered access to a view creates a rule based on the view, the rule inherits that user’s filtered results.

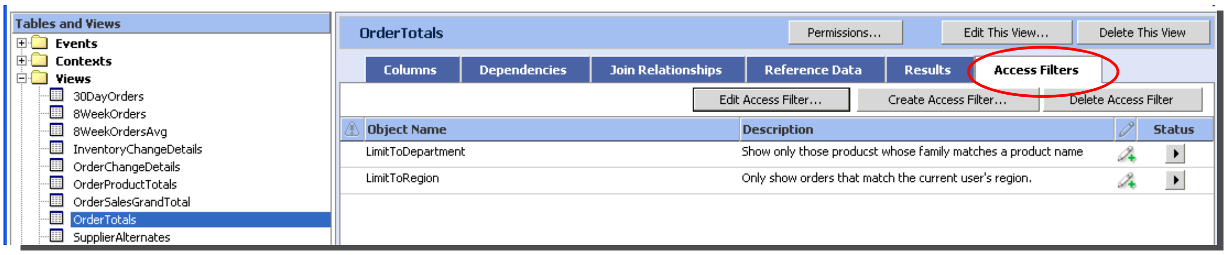
  As such, that rule only sees events that match the users access filter condition, and any subscribers to the alert associated with the view only receive alerts for the filtered events.

Creating a View Access Filter

To create an access filter, you must have Read-Write permission on the view.

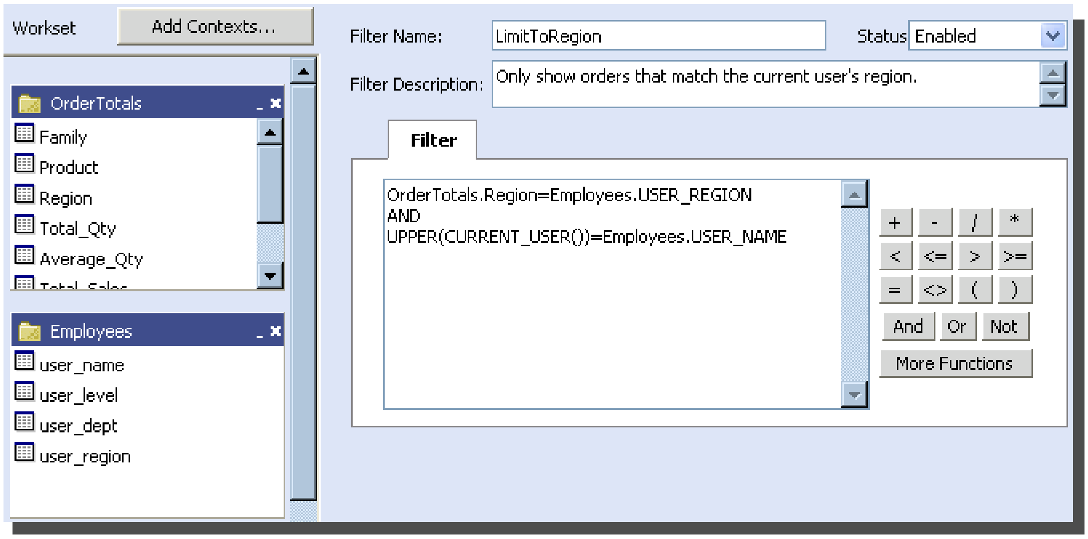
➤ To create an access filter:

1. In the BAM Workbench, Workbench tab, Views list, select the view that will have the filter.
2. Select the Access Filters tab to see the list of filters currently associated with this view.



Object Name	Description	Status
LimitToDepartment	Show only those producst whose family matches a product name	
LimitToRegion	Only show orders that match the current user's region.	

3. Click Create Access Filter to create the new filter.
4. Add a context table to the Workset by clicking Add Context. This example includes the Employees context table to retrieve information about the current user.
5. In the Edit Access Filter dialog, assign the filter's name and optionally provide a description. Define the filter condition following the instructions in "Access Filter Conditions" on page 16.



Filter Name: LimitToRegion

Status: Enabled

Filter Description: Only show orders that match the current user's region.

```
OrderTotals.Region=Employees.USER_REGION
AND
UPPER(CURRENT_USER())=Employees.USER_NAME
```

6. Save the access filter.

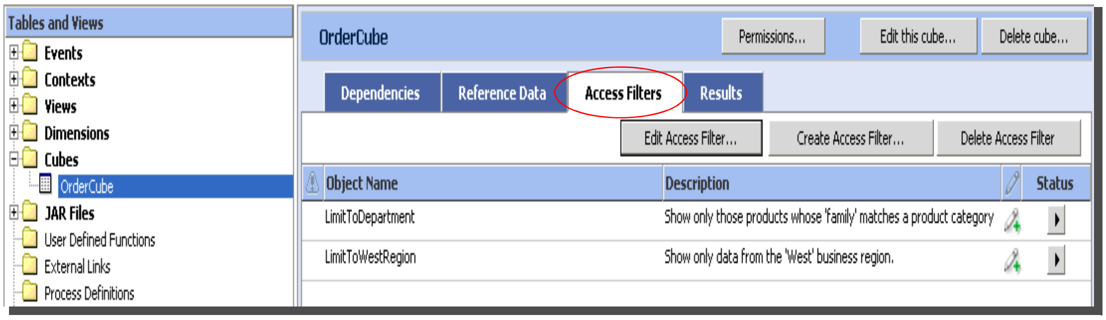
You can now assign the filter to users having Read-Filtered access to the view.

Creating a Cube Access Filter

You must have Read-Write permission on the cube.

➤ To create an access filter:

1. In the BAM Workbench, Workbench tab, Cubes list, select the cube that will have the filter.
2. Select the Access Filters tab to see the list of filters currently associated with this view.

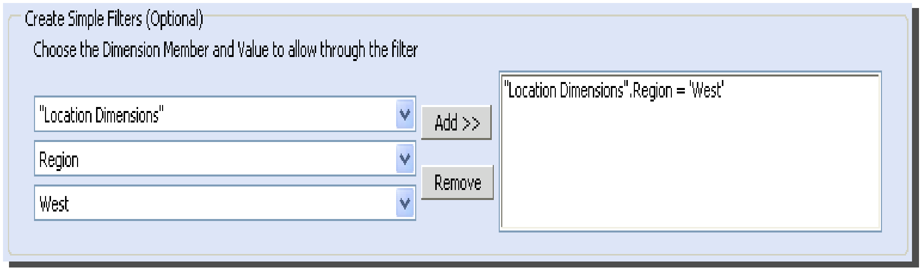


Object Name	Description	Status
LimitToDepartment	Show only those products whose 'family' matches a product category	
LimitToWestRegion	Show only data from the 'West' business region.	

3. Click Create Access Filter to create the new filter.
4. In the Configure Cube Filter dialog, assign the filter's name and optionally provide a description.
5. Define the filter condition per the details in "Access Filter Conditions" on page 16.

➤ To define a simple filter

- Choose the dimension, level, and value a shown in the following illustration:


Create Simple Filters (Optional)
Choose the Dimension Member and Value to allow through the filter
Add >>
Remove

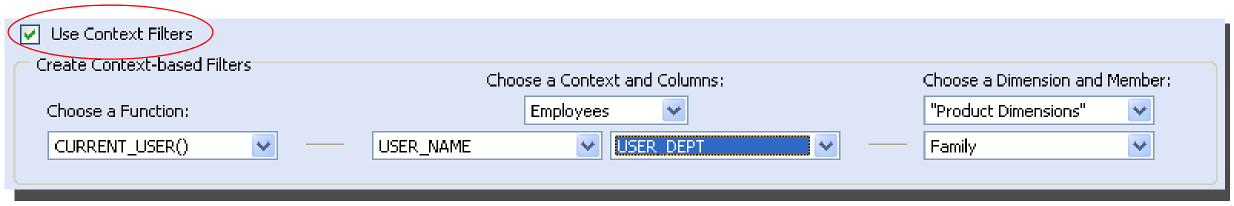
"Location Dimensions"
Region
West

"Location Dimensions".Region = 'West'

➤ To define a context filter

1. Choose the Use Context Filters option.
2. Choose the context table and identify the column that contains the user's name and the column that maps to a dimension level, as show in the follow illustration.

   Note: This filter is the same as the one shown in "Context Filters" on page 16.


Use Context Filters
Create Context-based Filters
Choose a Function:
Choose a Context and Columns:
Choose a Dimension and Member:

Use Context Filters ✓
Choose a Function: CURRENT_USER()
Choose a Context and Columns: Employees; USER_NAME; USER_DEPT
Choose a Dimension and Member: "Product Dimensions"; Family

3. Save the access filter.

You can now assign the filter to users having Read-Filtered access to the view.

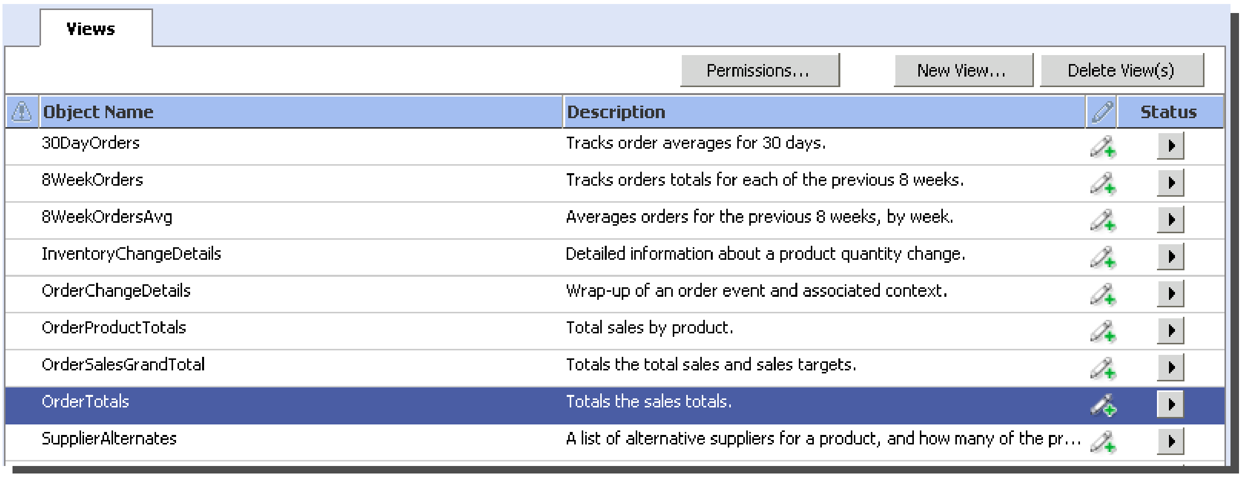
Assigning an Access Filter to Users and Roles

To perform this procedure:

- You must have Read-Write permission on the view or cube.
- The filters must already be defined. See "Creating a View Access Filter" on page 19," for details.

➤ **To assign an access filter to a user or role on a view:**

1. In the BAM Workbench, Workbench tab, click the Views folder in the Workset to list all of the views.
2. Select the view to assign the filtered read permission, and click Permissions.

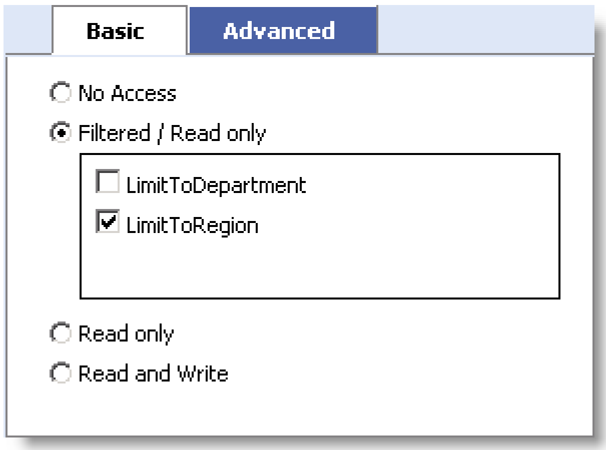


Object Name	Description	Status
30DayOrders	Tracks order averages for 30 days.	
8WeekOrders	Tracks orders totals for each of the previous 8 weeks.	
8WeekOrdersAvg	Averages orders for the previous 8 weeks, by week.	
InventoryChangeDetails	Detailed information about a product quantity change.	
OrderChangeDetails	Wrap-up of an order event and associated context.	
OrderProductTotals	Total sales by product.	
OrderSalesGrandTotal	Totals the total sales and sales targets.	
OrderTotals	Totals the sales totals.	
SupplierAlternates	A list of alternative suppliers for a product, and how many of the pr...	

3. Select one or more users or roles who will have the filtered access, and click Change Permissions.

4. Choose the Filtered/Read only permission, and choose one or more access filters to use.

   When multiple filters are assigned, the user sees the rows that meet any of the conditions. For example, when one filter shows only rows that are in the user's department classification, and another shows rows applicable to their business region, the user sees the row if either condition is true.



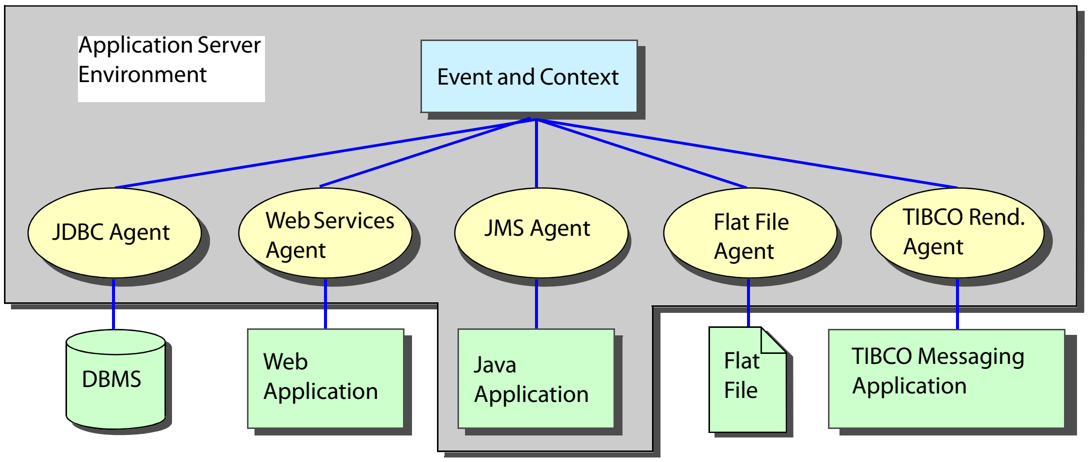
( ) No Access
(●) Filtered / Read only
- [ ] LimitToDepartment
- [✓] LimitToRegion

( ) Read only
( ) Read and Write

Save the assignments and the permissions are immediately put into effect.

3 Agents

Agents are Business Activity Monitoring processes that know how to receive or retrieve information from external sources. When an agent locates new event or context information, it passes that data to an Events and Context table for use by the Business Views.



Note: One agent may support multiple event or context tables.

While an agent knows *how* to communicate with an external source, event and context tables know *what* information is desired. As such, most of the event and context tables define the details that tell the agent what to look for.

➤ **In This Chapter:**

External Sources

Business Activity Monitoring provides agents to several external context and event sources. Some events stream (are pushed) into the system as they happen. Other events are loaded (pulled) as the result of a request, such as to a database or from a text file. Context data, however, are always pulled from the source. Table 1: Sources and Agents summarizes the available sources and identifies the source agent they support.

Table 1: Sources and Agents

External Source	Agent	Event Push	Event Pull	Context Pull
Java Database Connectivity (JDBC), usually from a Relational database (RDBMS)	JDBC (see page 216)	No	Yes	Yes
Java Messaging Service (JMS) from a Java application	Java Messaging Service (JMS) (see page 204)	Yes	No	No
Text file	Flat Files (see page 81)	No	Yes	No
TIBCO Rendezvous (RV) from a business application using TIBCO message streams.	TIBCO Rendezvous (see page 337)	Yes	No	No
Web services from a Web application over an HTTP connection	Web Services (see page 362)	Yes	No	Yes
JDBC agent used to access the database for a SAP system	ERP	No	Yes	No
Salesforce system.	Salesforce	No	Yes	Yes

Creating Agents

These steps summarize how to create an agent. For details about creating agents of specific types, see the descriptions of those types.

Note: For JBoss implementations, you must configure a corresponding JNDI definition in a celequest_context-ds.xml file. For more information, see the JBoss deployment instructions in the installation guide.

Before you create an agent, you must have create permission for agents (see "Creating Permission" on page 258 for details) and the connection specifications for the specific agent type.

➤ **To create an agent:**

1. Open the Administration Console tab of the BAM Workbench.
2. Select Agents.
3. Click New Agent.
4. Choose the source type for the Agent (as described in Table 1 on page 24).
5. Fill in the details for the specific source type. For details, see
   - "Flat File Agents" on page 89
   - "JMS Queue Agents" on page 210
   - "JMS Topic Agents" on page 213
   - "TIBCO Rendezvous Agents" on page 344
   - "Web Service Agents" on page 370

Save the object as enabled, and it will immediately be ready to receive events or context.

Note: See "Creating an SAP Agent" on page 314" for information on creating an ERP agent.

Editing Agents

These steps summarize how to modify an existing an agent. For details about agents attributes, see the description of the specific source.

Before you create an agent, you must have Read and Write permission for the agent (see "Accessing Permissions" on page 255 for details).

➤ **To edit or alter an agent:**

1. Open the Administration Console tab of the BAM Workbench.
2. Select Agents.
3. Double-click the agent to alter.
4. Fill in the details for the specific source type.

Save the object as enabled and it will immediately be ready to receive events or context.

4 Alerts

Alerts are the notifications of exceptional events sent to users or external systems. An alert may come in the form of a simple message indicating that an event has occurred, or it may more detailed, including information that indicates the cause and possible courses of action.

Each alert message is comprised of text that describes the exceptional incident to the subscriber. This is the text that appears in the BAM Workbench and in any other device identified by the subscriber's delivery profile. How the message is rendered depends on the device that displays it to the subscriber.

➤ **In this Chapter:**

Creating Alerts

To create an alert you need these permissions:

- Create permission for business activities (see "Creating Permission" on page 258 for details)
- Read and Write permission on the business activity that will contain the alert
- Read Only permission on the view or cube that will feed the alert

There are three ways to create alerts with the BAM Workbench Scenario Modeler:

- Stand-alone
- Associated with a rule
- Clone an existing alert

➤ To create a stand-alone alert:

1. Select an existing Business Activity.
2. Select an existing scenario to contain the alert.
3. Select the Alerts folder.
4. Click the New Alert button.
5. Choose the data source that will feed the alert.
   - If the scenario has a "default view", that one appears selected by default. Choose another source to monitor by clicking Select Data Source.
   - For a view, choose the view.
   - For a cube, choose the dimension level in a cube. Optionally you may also apply a filter that further restricts the data that the cube feeds to the alert.
   - If the source contains data, that data appears to provide a sample of what to expect. When the source is empty, the form displays just the column names and the message "No Data Available."
6. Fill in the fields in the Alert Definition form.

➤ To create an alert associated with a rule:

1. Follow the instructions for "Creating Rules" on page 300.
2. Fill in the fields in Step 2 of 2: the Alert Definition form.

➤ To clone an existing alert:

Copy the definition of an existing alert to a new alert.

1. Edit the alert you want to clone.
2. Change the alert name, and change the other attributes that differ from the original alert.
3. Choose "Save as New Alert".

Alert Attributes

Note: It is recommended you modify the notification settings that control the maximum number of alerts that can be sent within a specified time interval. For more information, see "Setting Alert/Notifications Limitations" on page 119 in *Using Business Activity Monitoring Workbench*.

Every alert has the following attributes:

Attribute	Description
Alert Name	Identifies the alert object. The name can contain letters and numerals only. This name must be unique among alerts within the same scenario. See "Object Namespace" on page 248 for details.
Status	Specifies if the rule is enabled (receiving new event information) or disabled. **Note:** When the containing scenario is disabled, you cannot make the alert enabled. The scenario must be enabled before the alert may be enabled.
Importance	A hint about how important a message is. Values are HIGH, NORMAL (default), or LOW. Messages arriving in the BAM Dashboard are sorted into folders corresponding t the importance level. Further, e-mail messages are flagged accordingly with the "Importance" mail header field per mail standards.
Description	Optional description that may contain any text characters.
Data source	View or cube dimension level that defines the columns in the alert. Note that this should be the same source as the associated rule, or one derived from that source; otherwise, the generated alert might not contain valid information.
Subscribers	Users who receive the alert. See "Alert Subscribers" on page 31 for details.
Subject	Text message that is the subject of the alert, similar to an e-mail subject line. Can contain column references to the underlying business view.
Body	Text message that is the body of the alert. Can contain column references to the underlying business view, and can contain acknowledgements and reportlets. For details about acknowledgements, see Acknowledgements, for reportlets, see "Reportlets" on page 290."

Message Subject and Body Text

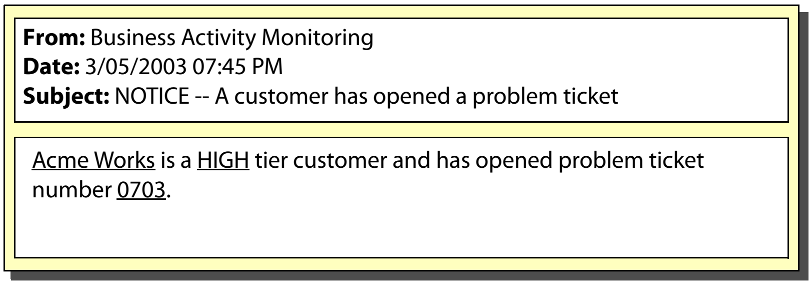
Each alert message is comprised of text that describes the exceptional incident to the subscriber. This is the text that appears in the BAM Workbench, BAM Dashboard, and in any other device identified by the subscriber's delivery profile. How the message is rendered depends on the device that displays it to the subscriber.

The Subject and Body alert attributes define the text of the message. Each attribute contains static text and fields. When the alert generates the message, it replaces the fields with the values from the columns of the same name in the business view row that caused the alert.

For example, consider this default message definition:

Subject: NOTICE -- A customer has opened a problem ticket.
Body: CUST_NAME is a TIER tier customer and has opened problem ticket number TICKET.

When the alert is activated, it generates a message similar to this e-mail:



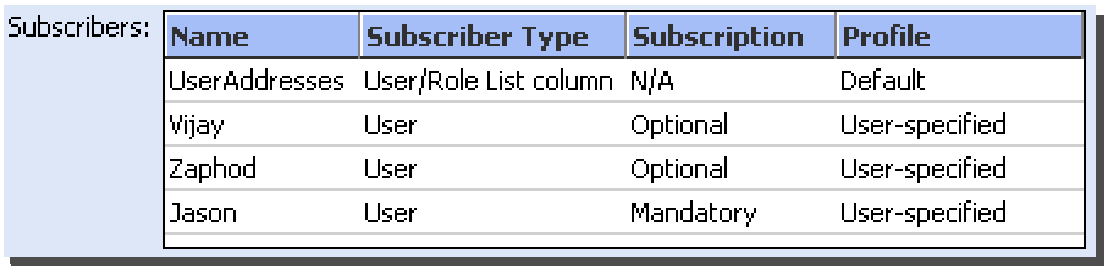
From: Business Activity Monitoring
Date: 3/05/2003 07:45 PM
Subject: NOTICE -- A customer has opened a problem ticket

Acme Works is a HIGH tier customer and has opened problem ticket number 0703.

Note: You can include any valid HTML code in the body of the message. If you reference an external object, such as a graphic, make the reference to a HTTP server; do not reference a local file because it will not be included in the message.

Alert Subscribers

When you define an alert, you can also declare one or more Users to receive the alert notification. By clicking the Add/Remove button next to the subscribers list in the Alert editor, you can designate individual users and Roles to receive the alert, or identify columns in the alert's view that provide lists of users, roles, or e-mail addresses to receive the notification.

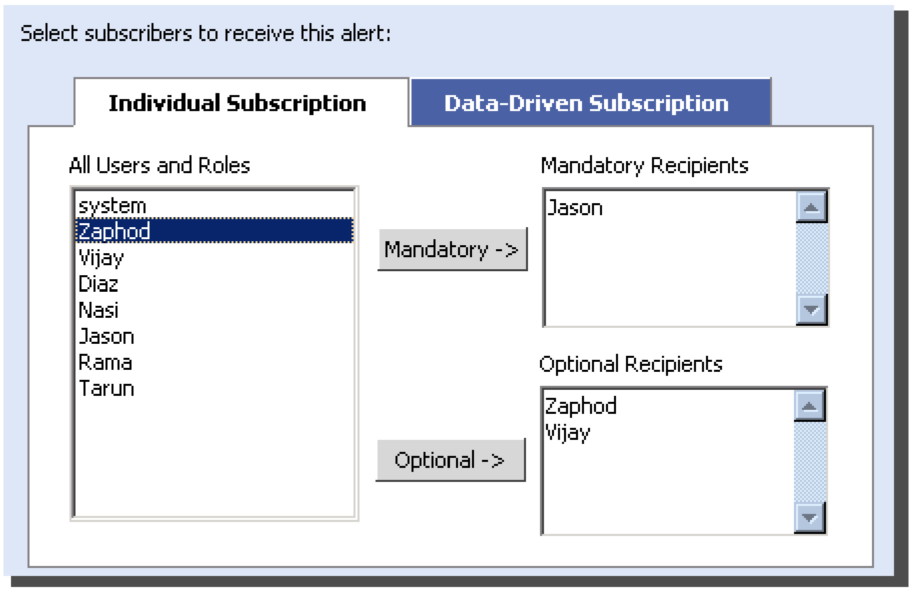


Subscribers:

Name	Subscriber Type	Subscription	Profile
UserAddresses	User/Role List column	N/A	Default
Vijay	User	Optional	User-specified
Zaphod	User	Optional	User-specified
Jason	User	Mandatory	User-specified

The Alert Subscribers dialog has two tabs:

- Individual Subscription tab is where you choose the users and roles to receive the notification.
  - Mandatory subscribers always receive the notification. These users or roles cannot voluntarily unsubscribe to the alert in the BAM Dashboard; rather, they must be removed from this dialog.
  - Optional subscribers receive notifications, but they may unsubscribe using the BAM Dashboard.



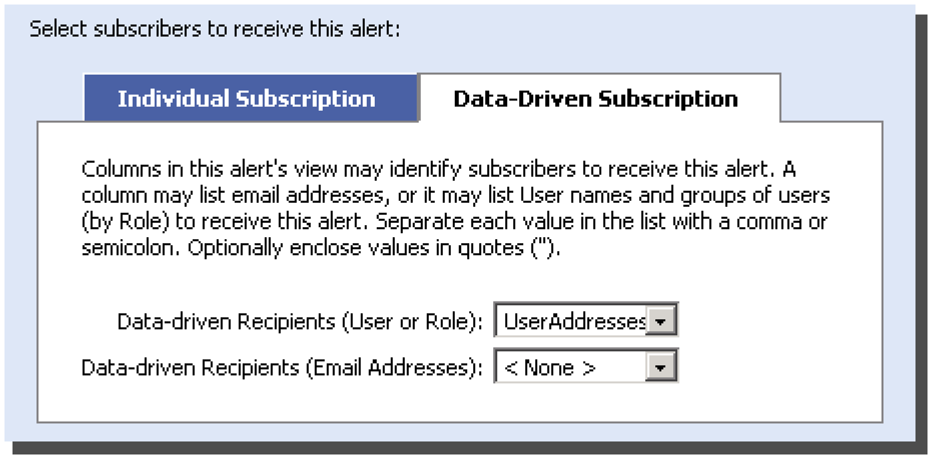
Select subscribers to receive this alert:

All Users and Roles: system, Zaphod, Vijay, Diaz, Nasi, Jason, Rama, Tarun

Mandatory Recipients: Jason

Optional Recipients: Zaphod, Vijay

- The Data-Driven Subscription tab is where you identify columns in the view that contain the names or addresses of users or roles to receive the notification. A column may contain either a list of users and roles, or a list of e-mail addresses to receive the notifications. Each list of values in a column is separated by a comma or semi-colon, and each item may optionally be enclosed in quotes (").


Select subscribers to receive this alert:

Columns in this alert's view may identify subscribers to receive this alert. A column may list email addresses, or it may list User names and groups of users (by Role) to receive this alert. Separate each value in the list with a comma or semicolon. Optionally enclose values in quotes (").

Data-driven Recipients (User or Role): UserAddresses
Data-driven Recipients (Email Addresses): < None >

Note: Data-driven subscriptions are mandatory: the users, roles, and e-mail addresses that receive them cannot "unsubscribe." Further, users who receive these subscriptions as a result of an e-mail address list do not see them in the BAM Dashboard list of subscriptions.

When the column contains multiple instances of the same, exact e-mail address, only one message is sent. However, slight differences in the entries will generate one message for each instance. For example, these to variations of the same address myname@lava.com and "Me" <myname@lava.com> generate two messages.

Managing Alert Notification Messages

Users receive alert notifications the devices identified by their delivery profiles in the "Delivery Profiles Tab" on page 334. For details about using the Alert Manager in the BAM Dashboard to view, subscribe and delete notifications, see "Interacting with Alerts" on page 139.

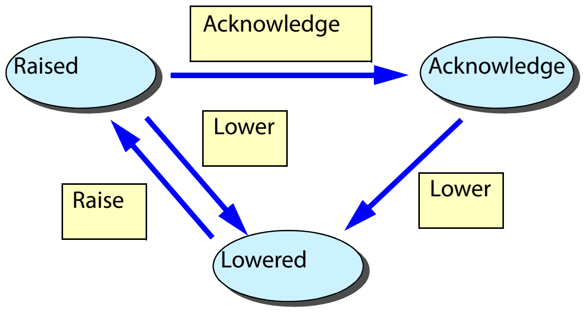
Alerts remain in the Alert Manager list until deleted specifically by the user or automatically by the system. The system removes messages after a count of days specified by the system administrator in the Systems Settings dialog box, as described in "Working with System Settings" on page 108 in the BAM Workbench documentation.

Also, you can control the maximum number of alerts that can be sent within a specified time interval. For more information, see "Setting Alert/Notifications Limitations" on page 119 in the BAM Workbench documentation.

Alert States

Business Activity Monitoring provides two kinds of alerts:

- Stateless alerts are one-time notifications about the business condition when the condition happens. Stateless alerts are said to be fired when the rule condition is met. For example, a stateless alert might fire a notification to a warehouse manager when a product inventory count falls below a specific threshold. Note that every subsequent change in inventory levels for that product also sends a notification as long as the inventory count remains below the threshold.
- Stateful alerts have a status that is raised or acknowledged as long as the business condition exists, and which is lowered when the condition does not exist. With a stateful alert, warehouse managers receive the alert when the inventory falls below threshold, and do not receive another until the alert is lowered, presumably after inventory levels have been restored above the threshold. When multiple parties have interest in an alert, one may choose to handle the raise alert and *acknowledge* it. This is done by clicking the Acknowledge link in the message body. The alert's creator places the link in the message body when creating the alert. See Acknowledgements for details.



By combining a stateful alert with a Holds for time period, you can delay the notification. For example, only alert the warehouse manager when a product's inventory count has remained below a threshold for one day: "the alert condition *holds for* 1 day". This way the manager doesn't receive the notification if the inventory drops within a day of being restocked.

Alert Escalation

You can monitor the alert's state and generate new alerts when conditions demand. For example, if an alert has not been handled in a timely manner, a new alert can be sent to more significant users, in effect escalating the original alert. To test for these states, use the IS_RAISED function. See "Monitoring Alerts" on page 303 for details.

Consolidating Multiple Messages

Events may contain multiple rows of information. When the event meets a rule condition, that rule generates one alert for each row of the event. Often it is desirable to send only one message describing all of the alerts. This is called a *consolidated alert*.

For example, consider a new purchase order entering the system (an event). If the quantity of items in-stock is insufficient to fulfill the order, an alert might note that condition. When multiple line items on the order have insufficient inventory, each generates a new alert. To send just one notification instead, use a consolidated alert.

➤ **To create a consolidated alert:**

On the Create Rule form, check the Consolidate multiple messages from same alert option.

Setting an Alert to Invoke an External Web Service

You can configure an alert message to invoke an external Web service.

The procedure in Business Activity Monitoring requires you to independently create a packaged Java function that constructs the Web service message based on the alert payload. Specifically, this function determines which data elements in the alert payload are mapped to the inputs expected by the Web service. This function is then uploaded to Business Activity Monitoring as a user-defined function (UDF). The Web service is specified in a Web service profile that calls the UDF. The Web service profile can then be set as a subscriber to the desired alert.

Note: Creating this function requires programming expertise. For more information, see "About the Web Service User-Defined Function" on page 35.

➤ **To configure an alert to invoke a Web service:**

1. Create a new alert, as described in "Creating Alerts" on page 28.

   Note: Do not subscribe any users to the new alert at the moment. You will add subscribers later.

2. Create a function that constructs the Web service message using parts of the alert data, and package as a JAR.

   For more information, see "About the Web Service User-Defined Function" on page 35.

3. Upload the JAR as a user-defined function in the BAM Workbench.

   For more information, see "Creating and Using a UDF" on page 347.

4. Create a new user and save it.

   Note: You must create and save the new user so you can change the default dashboard profile properties.

5. Edit the new user as follows:

   - In the Edit User dialog box, click the Delivery Profiles tab.
   - Open the default dashboard profile.
   - Deselect the **Automatically add this profile...** option.
   - Click **OK**.

6. For the new user, create a Delivery Profile as follows:
   - In the Edit User dialog box, select the **Delivery Profile** tab.
   - Click the **Create New Profile** button.
   - In the resulting dialog box, for **Profile Type**, select *Web Service*.
   - For **Profile Name**, enter an appropriate value.
   - For **Web Service URL**, enter the endpoint for the WSDL.
   - For **Method**, specify the desired method (operation) to be invoked in the Web service.
   - Supply **Username** and **Password**, as required.
   - For **UDF**, select the JAR you uploaded for this Web service invocation.
   - Ensure that the **Automatically add this profile...** option is unselected.
   - Click **OK**.
7. Return to the configuration for the alert you created in step 1 above.
8. Add the user you just created as a subscriber.
9. Click **Save**.

   When the alert fires, the Web service will be invoked.

About the Web Service User-Defined Function

The function you create must extend the `com.celequest.api.function.webservice.IAlertWSMessageConstructor` class.

This function receives as input a DOM element that describes the alert data generated by Business Activity Monitoring and which conforms to the alertMessage.xsd (included in the BAM_HOME/samples directory). It generates as output an AXIS message containing the SOAP that is sent to the Web Service.

com.celequest.api.function.webservice.IAlertWSMessageConstructor interface:

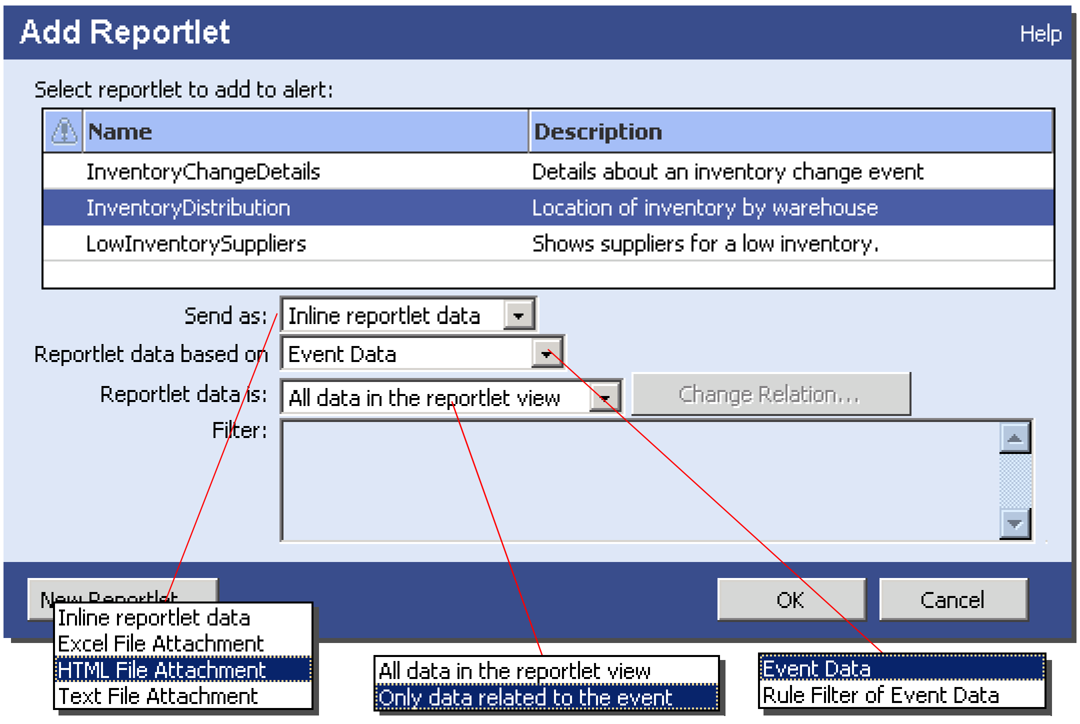
```
package com.celequest.api.function.webservice;
import com.celequest.api.function.*;
import org.apache.axis.Message;
import org.w3c.dom.Element;
import java.lang.Exception;

public interface IAlertWSMessageConstructor extends IUDFunction
{

public Message constructMessage(Element alertData, String wsdlURI,
String soapOperation) throws Exception;
}
```

Alert Reportlets

Reportlets describe the contents of a view and present that information in a report that is attached to the alert message. Frequently reportlets provide information about an event that puts the event into context.

➤ **To include reportlets in the body of the alert notification:**

1. Open the Add Reportlet dialog.
2. Select the reportlet from the list.
3. Choose OK to add it to the alert.



Add Reportlet Help

Select reportlet to add to alert:

Name	Description
InventoryChangeDetails	Details about an inventory change event
InventoryDistribution	Location of inventory by warehouse
LowInventorySuppliers	Shows suppliers for a low inventory.

Send as: Inline reportlet data
Reportlet data based on: Event Data
Reportlet data is: All data in the reportlet view
Filter:

Send As

Alerts displayed in the BAM Dashboard embed the reportlet as an in-line, HTML table. The Send as option specifies the format of the reportlet to attach to the alert notification sent to user subscription profiles. The reportlet can be embedded in the body of the message, or included as an attachment in one of the available formats.

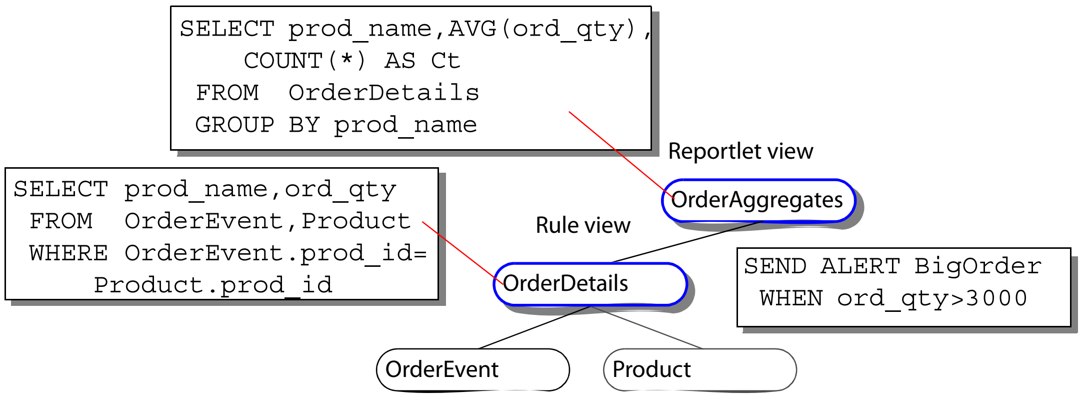
The next section, "Reportlet Filtering" on page 36, describes *Reportlet data based on* and *Reportlet data is* options.

Reportlet Filtering

When you create the reportlet, you identify the view or cube from which the reportlet draws its data. However, the data that appears in the reportlet depends on the type of the source (stateless or stateful), and how the rows of data are filtered as specified on the Add Reportlet dialog. In general,

- *Reportlet data based on* specifies whether a reportlet's stateful source should include data based on all events, or just those that have met the rule condition.
- *Reportlet data is* limits whether the rows in the reportlet's source include only those related to the event that activated the alert, or all previous event data also in the source.

To better understand how these settings affect the reportlet data, consider these two views that track and report on product orders. The *OrderDetails* stateless view is a summary of each order event, while the *OrderAggregates* stateful view tracks the average quantity for each product ordered.



Now consider these events:

```
prod_name  ord_qty
---------  -------
nails         1000
plywood       1000
nails         4000
nails         4000
plywood       5000
```

After the events have entered the stream, the *OrderAggregates* view has these values:

```
prod_name  AVG(ord_qty)  Ct
---------  ------------  --
nails              3000   3
plywood            3000   2
```

By default, a reportlet using the *OrderAggregates* view shows the details for both products, regardless of which product event might have generated the alert. Further, even though only the last three events met the rule condition of ord_qty greater than 3,000, the reportlet shows the results from all events, which might not be what you intended.

The two filtering options on the Add Report dialog alter the results by filtering the results that appear in the reportlet.

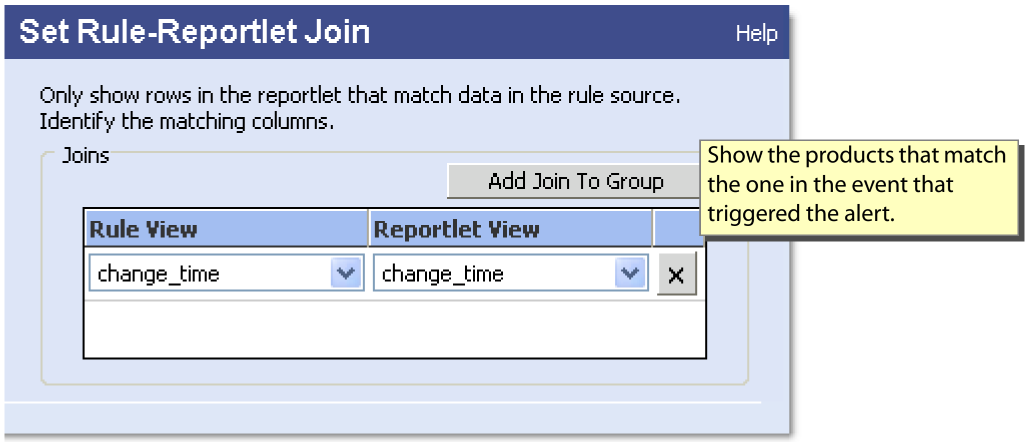
The Reportlet data is option causes the reportlet to show one of the following:

- All of the data in its view or cube face
- Only those data related to the event found by the rule

To show only the event-related data, you must define the relationship between the event and reportlet sources. For example, if you want the reportlet to only show the result for "products" in both views, define the relation by picking the prod_name column from both views. This tells the reportlet to show only those rows in the *OrderAggregates* view whose prod_name value matches the name in the *OrderDetails* view. Then, the reportlet shows the nails value only when the rule generates the alert.

```
prod_name   AVG(ord_qty)  Ct
---------   -----------   --
nails              3000    3
```

Similarly, when working with cube face, you pick columns that best identify the event to the reportlet. For example, this illustration shows PROD_NAME in the rule view being joined to the PRODUCT dimension in the reportlet cube:


Set Rule-Reportlet Join

Help

Only show rows in the reportlet that match data in the rule source.
Identify the matching columns.

Joins

Add Join To Group

Rule View	Reportlet View
change_time	change_time

Show the products that match the one in the event that triggered the alert.

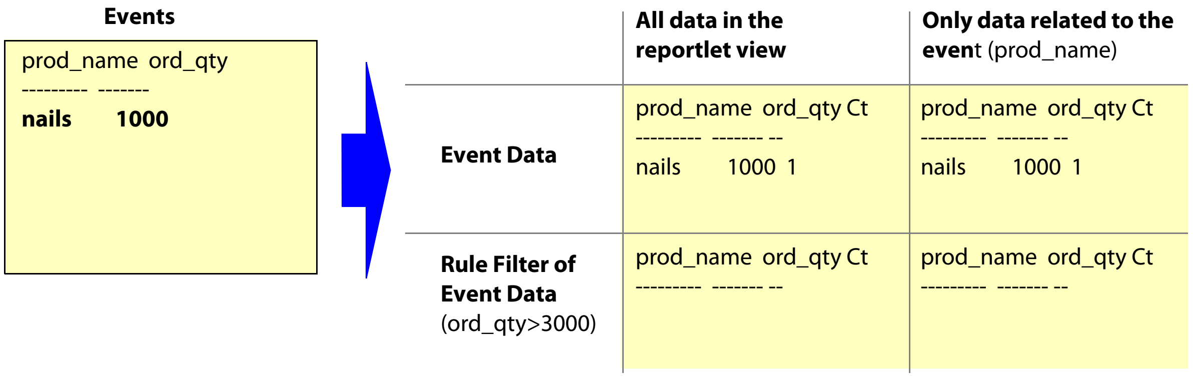
A variation would be to link the product family instead of the product name. In that case, the reportlet shows all of the products in the same product-family as the one that triggered the event.

Reportlet Data Based On Option

The *Reportlet data based on* option specifies whether a stateful view should include data based on all events, or just those that have met the rule condition. Following the example above, the *OrderAggregates* view AVG(ord_qty) column has a value of 3,000 for “nails” after both events have be processed. This is what happens when the setting for this option is Event Data. However, the rule condition says to generate an alert only when the order quantity is greater than 3,000. To track only events that have met the rule condition, change the setting for this option to Rule Filter of Event Data. Then the reportlet shows 4,000 as the average because 4,000 is the average of the two events greater than 3,000.

Example

These illustrations show the view results on the example data when you use the two reportlet filtering options. Notice that the first event does not pass the rule filter, and does not appear in those views.

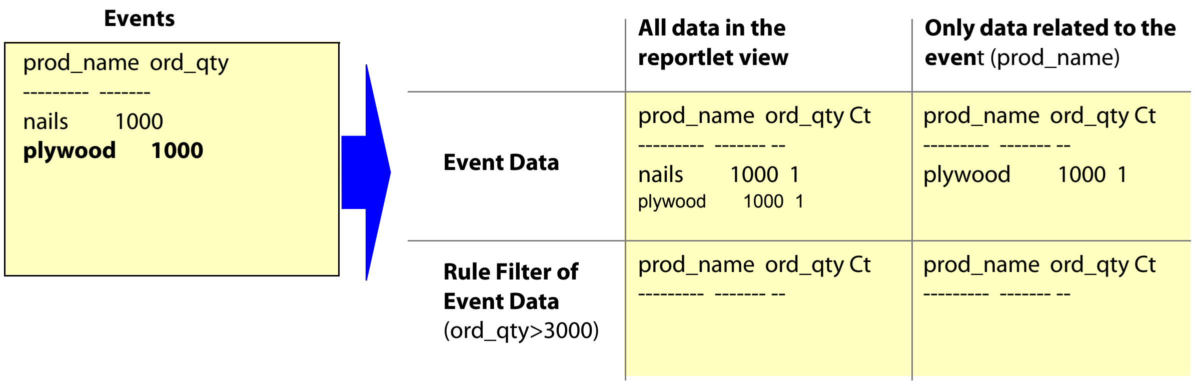


Events

```
prod_name  ord_qty
---------  -------
nails      1000
```

	All data in the reportlet view	**Only data related to the event** (prod_name)
Event Data	prod_name ord_qty Ct --------- ------- -- nails 1000 1	prod_name ord_qty Ct --------- ------- -- nails 1000 1
Rule Filter of Event Data (ord_qty>3000)	prod_name ord_qty Ct --------- ------- --	prod_name ord_qty Ct --------- ------- --

Similarly, the second event also does not pass the rule filter. Notice though that the view that shows event related data now only includes the plywood event.

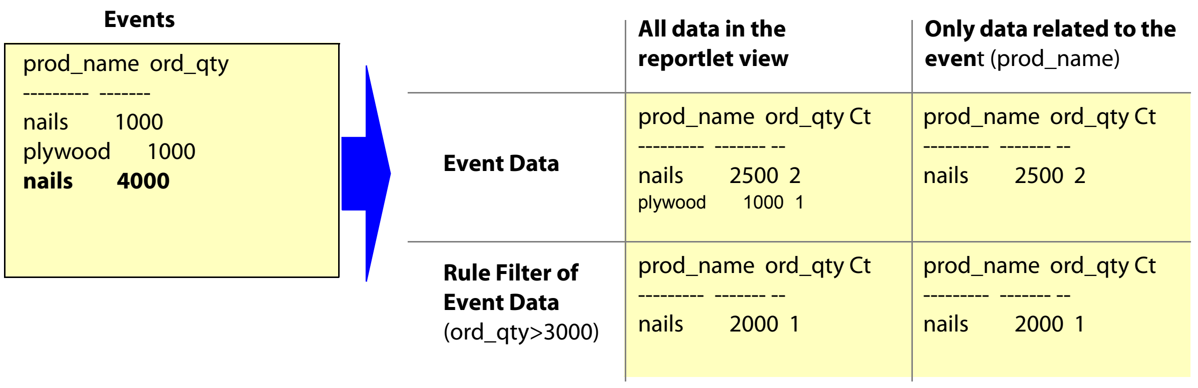


Events

```
prod_name  ord_qty
---------  -------
nails      1000
plywood    1000
```

	All data in the reportlet view	**Only data related to the event** (prod_name)
Event Data	prod_name ord_qty Ct --------- ------- -- nails 1000 1 plywood 1000 1	prod_name ord_qty Ct --------- ------- -- plywood 1000 1
Rule Filter of Event Data (ord_qty>3000)	prod_name ord_qty Ct --------- ------- --	prod_name ord_qty Ct --------- ------- --

The third event now passes the rule filter, and as such, appears in the bottom views. And once again, nails is the product in the event-related views.



Events

```
prod_name  ord_qty
---------  -------
nails      1000
plywood    1000
nails      4000
```

	All data in the reportlet view	**Only data related to the event** (prod_name)
Event Data	prod_name ord_qty Ct --------- ------- -- nails 2500 2 plywood 1000 1	prod_name ord_qty Ct --------- ------- -- nails 2500 2
Rule Filter of Event Data (ord_qty>3000)	prod_name ord_qty Ct --------- ------- -- nails 2000 1	prod_name ord_qty Ct --------- ------- -- nails 2000 1

The third nails event also passes the rule filter.

Events

prod_name	ord_qty
nails	1000
plywood	1000
nails	4000
nails	**4000**

	All data in the reportlet view	**Only data related to the event** (prod_name)
Event Data	prod_name ord_qty Ct --------- ------- -- nails 3000 3 plywood 1000 1	prod_name ord_qty Ct --------- ------- -- nails 3000 3
Rule Filter of Event Data (ord_qty>3000)	prod_name ord_qty Ct --------- ------- -- nails 4000 2	prod_name ord_qty Ct --------- ------- -- nails 4000 2

The final event again changes the event-related views. Notice that all events are reflected in the upper-left view, while only those that passed the rule filter are in the lower-left view.

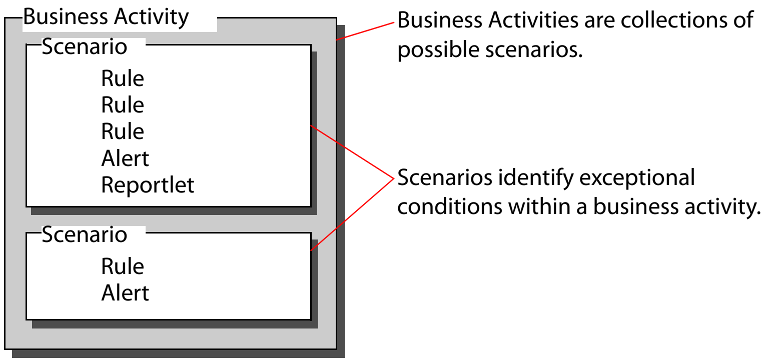
Events

prod_name	ord_qty
nails	1000
plywood	1000
nails	4000
nails	4000
plywood	**5000**

	All data in the reportlet view	**Only data related to the event** (prod_name)
Event Data	prod_name ord_qty Ct --------- ------- -- nails 3000 3 plywood 3000 2	prod_name ord_qty Ct --------- ------- -- nails 3000 2
Rule Filter of Event Data (ord_qty>3000)	prod_name ord_qty Ct --------- ------- -- nails 4000 2 plywood 5000 1	prod_name ord_qty Ct --------- ------- -- nails 5000 1

5 Business Activities

A *business activity* is a collection of possible scenarios that identify exceptional business conditions. Each scenario contains rules that identify specific possible conditions, and the alerts and reportlets to send to key personnel when the condition is found to exist.


Business Activity
Scenario
Rule
Rule
Rule
Alert
Reportlet
Scenario
Rule
Alert
Business Activities are collections of possible scenarios.
Scenarios identify exceptional conditions within a business activity.

You must have at least one business activity before creating any scenarios, rules, alerts, or reportlets.

Tips:

- Deleting a business activity deletes its contained scenarios, and all of the scenario's objects.
- Disabling a business activity disables its contained scenarios, rules, alerts, and reportlets.

➤ In this Chapter:

Creating Business Activities

You need Create permission for business activities (see "Creating Permission" on page 258 for details).

➤ **To create a new business activity**

1. Open the Scenario Modeler.
2. Click New Business Activity…
3. Fill in the fields of the New Business Activity dialog.

Business Activity Attributes

Every scenario has the following attributes:

Attribute	**Description**
Name	Identifies the business activity. The name can contain letters and numerals only. This name must be unique among business activities and users; you cannot have a user with the same name as a business activity. See "Object Namespace" on page 248 for details.
Status	Specifies if the business activity is enabled (receiving new event information) or disabled. When an activity is disabled, all of the objects it contains are also disabled, including the rules, alerts, and reportlets.
Description	Optional description that may contain any text characters.

Deleting Business Activities

You need Read and Write permission on the business activity.

Note: Deleting a business activity deletes its contained scenarios and all of the scenario's objects.

➤ **To delete a business activity:**

1. Open the Scenario Modeler.
2. In the tree view, select the Business Activities folder.
3. In the list of business activities, select the activity to remove.
4. Click Delete Business Activity.

6 Context

Context supports event processing by providing meaningful information about the event. Contexts are the business information stored in databases, data warehouses, or are provided by Web Services. Context tables receive data from Agents that know how to communicate with information sources. When you define a context table, you also instruct the agent how to identify the information from the source.

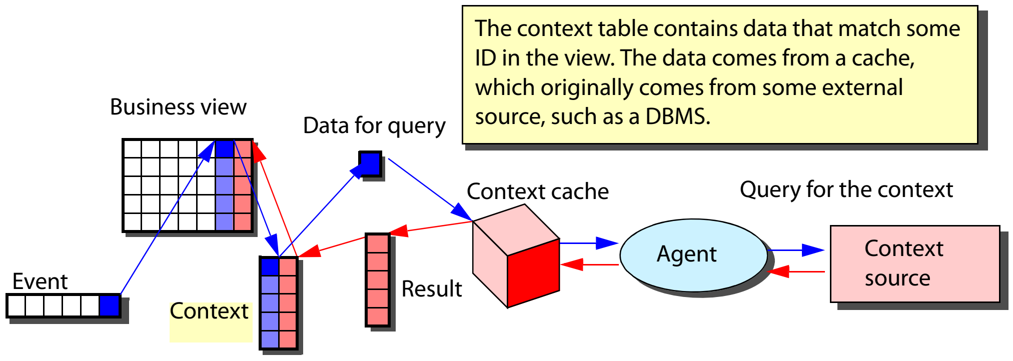
➤ **In this Chapter:**

How It Works

When a business view requires context information, it does so based on some information already in the view. For example, a view that is processing a purchase order might have received a product identification number along with the event data. If the view also requires the suppliers of that product, it would retrieve the supplier names from a context table that contains the names that matches the ID. In the view definition, a WHERE clause would join the context table to the event, similar to the following:

```
WHERE event.product_id = context.suppliers_of_product_id
```

When the view performs this join, it passes the ID from the event to the context table. If the matching supplier data are already in the context cache, the table uses that data and passes it to the business view. If the data are not already in memory, the ID is passed — either as an SQL query or by value for a stored procedure — to an agent, which sends data to the DBMS or Web service for processing. The result of the query is then loaded into the context table and subsequently included in the business view.


The context table contains data that match some ID in the view. The data comes from a cache, which originally comes from some external source, such as a DBMS.
Business view
Data for query
Context cache
Query for the context
Agent
Context source
Event
Context
Result

The context source may be databases accessed through a JDBC or business applications accessed through Web Services. For details about these type of sources, see “JDBC Tables” on page 217, or “Web Service Context” on page 366.

Creating Context Tables

Every context table has a name, description, status attribute, and agent. These are defined in the following table.

Attribute	**Description**
Name	Identifies the table and is the name accessed by the Business Views that depend on this table. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	(*optional*) Description of the table.
Status	Whether or not the object is enabled (able to receive and pass data) or disabled (not receiving or passing data).
Agent	The agent that retrieves the context information, and passes the data to the event or context object. See "Agents" on page 23 for information about agent types.
Disable context after errors	Count of consecutive errors to receive before the system disables this context. Once disabled, a context must be re-enabled manually.

Before creating a context table, you must have Create permission for tables (see "Creating Permission" on page 258), and Read Only access permission on the agent that will feed the table.

➤ To create a context table:

1. Open the BAM Workbench tab.
2. Click New Context…
3. Choose the source type, each type has its own specific attributes.

   For details, see:

   - "JDBC" on page 216
   - "Web Services" on page 362
4. Fill in the fields in the New Context form.

Save the table as enabled and it will immediately be ready to receive context.

Editing Context Tables

Editing the attributes of a context table causes the object to lose state, and possibly invalidates dependant views. For example, if you remove a column, any view or rule that references that column becomes invalid. (However, if you redefine the column in the table, the dependant views are automatically revalidated.)

Before editing a context table, you must have Read and Write permission for tables (see "Accessing Permissions" on page 255), and Read Only access permission on the agent that feeds the table.

➤ **To edit a context table:**

1. Open the BAM Workbench tab.
2. Select the event or context object.
3. Chose Edit This Context.
4. Change the definitions in the Edit Context form. Note that each type has its own specific attributes. For details, see
   - "JDBC" on page 216
   - "Web Services" on page 362

Save the table as enabled and it will immediately be ready to receive events or context.

Context Column Limitations in Queries

Context can be retrieved with *no limitations from a JDBC query source*. However, the following limitations apply when retrieving context from a JDBC stored procedure source or a Web service source:

- When the context column is referenced as part of a query, somewhere in the WHERE or FROM clause it *must* appear in an equality expression and then only as an atomic predicate (no other operators on the same side of the equal sign). For example, the following is permitted:

  ```
  WHERE context_column = 10*event_column
  ```

  But the following is *not permitted* because the left-side predicate, which contains the context column, is an expression that includes an operator (/):

  ```
  WHERE context_column/10 = event_column
  ```

- The required atomic reference *may not* appear in a disjunct (OR) expression. The following fails:

  ```
  WHERE (context_column = event_column OR A > B)
  ```

  However, it *may* appear in a conjunct (AND) expression:

  ```
  WHERE (context_column = event_column AND A > B)
  ```

- Once there is at least one equality reference in the query, you may use the column in any other way. For example, the following two queries are permitted:

  ```
  WHERE (context_column = event_column AND
         context_column/10 = other_event_column)
  ```

  ```
  WHERE ((context_column = event_column AND
          context_column >= other_event_colum) OR
         (A > B))
  ```

  But the following *fails* because there is no equality reference in the query:

  ```
  WHERE context_column >= event_column
  ```

- The required equality expression *may not* reference another context column in the same table. For example:

  ```
  t1.context_column = t1.other_context_column
  ```

- However, the equality expression *may* reference a context column in another table, for example:

  ```
  t1.context_column = t2.other_context_column
  ```

Sybase Limitations

When making a query to a Sybase database, be aware of these limitations:

- All names, including tables and columns, are case-sensitive.
- All queries must be in the form `SELECT * FROM table` only; you cannot include any SELECT clauses. To filter the results, load them into a business view, and then filter that view.

Caching Context Queries

Caching allows you to store the results of context queries in memory. Subsequent requests for the same information are then retrieved from memory instead of impacting the DBMS with a redundant query. When caching is active, and a view requests context, it searches the cache first. If the desired data are not in the cache, Business Activity Monitoring issues a query to the database if on-demand caching is enable; however, if prefetch caching is enabled, Business Activity Monitoring issues a query to the prefetch cache. (See the following sections "On-Demand Caching" and "Prefetch Caching" for a description of these caching methods.)

The context cache has these parameters:

- Cache data for this context

  Either cache or do not cache context query results to the recent query cache. This parameter must be set to enable the remaining parameters to be set. You must also add at least one invalidation schedule when selecting this parameter.

- Enable prefetch

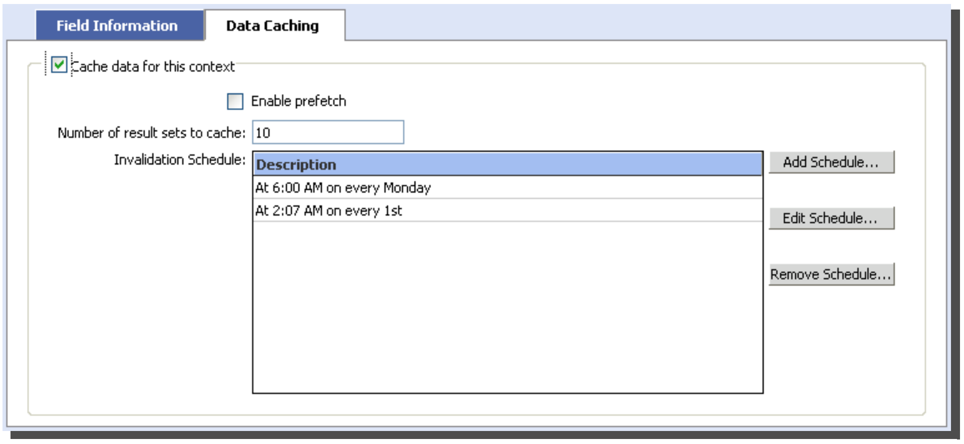
  Either enable or disable context prefetch. This caches the entire external context table into memory from the external data source and becomes a replacement for the external query source.

- Number of results to cache

  Count of query results to cache in memory. Each set of results may contain one or more rows of context related to the event.

- Invalidation schedule

  Identifies when to invalidate the cache and discard all information currently in the cache.



✓ Cache data for this context

Enable prefetch

Number of result sets to cache: 10

Invalidation Schedule:

Description
At 6:00 AM on every Monday
At 2:07 AM on every 1st

On-Demand Caching

On-demand caching occurs when you have selected "Cache data for this context" on the data caching tab and have **not** selected "Enable prefetch." With on-demand caching, the recent-query cache is created that maintains results on a least recently used (LRU) basis. This cache keeps track of when each result set was last requested. When the cache is full, it keeps the most recently accessed rows and discards those that have not been accessed in the longest period of time. If data is requested that is not in the recent query cache, the data is retrieved from the external data source.

Note: Rows containing frequently requested data will remain in the recently query cache the longest to reduce impact on the database. However, if details about the information can change often, define an invalidation schedule to account for the changes and thereby invalidate the cache.

When a scheduled invalidation occurs, the recent query cache is cleared and updated on subsequent queries. If context data is not rapidly changing, it is best to invalidate the recent query cache less often. For example, if the context is fairly static, you might want to invalidate the cache weekly or monthly. However, if the context database is updated nightly, you might want to invalidate the recent query cache nightly as well to ensure the latest data.

On demand caching can deliver better performance than if no caching is used. However, you might be able to improve performance further by using prefetch caching.

Prefetch Caching

When you enabling the prefetch cache, the entire external context table is cached into memory from the external data source and becomes a replacement for the external query source. The recent query cache functions as described in the previous section, "On-Demand Caching"; however, when data is requested that is not in the recent query cache, the data is retrieved from the prefetch cache instead of the external data source. When an invalidation occurs according to your specified invalidation schedule, the recent query cache is cleared and a query is issued to the external data source to update the in-memory copy of the context table. The recent query cache is updated from the prefetch cache on subsequent queries.

When enabling the prefetch data cache, you should consider the following:

- More memory is used because the information from the external data source is stored in memory, and an index is built for each column.
- The prefetch cache is only updated at the scheduled invalidation time. If data is not in the prefetch cache, no data is returned. Updates to the external data source are not reflected in the prefetch cache until after the next scheduled invalidation.

The prefetch cache retries up to 5 times to refresh on the invalidation schedule if an exception occurs. If it cannot refresh from the external data source, the recent query cache is then used until the next scheduled refresh.

7 Cubes

A *cube* is a set of data organized by dimensions and measures for the purpose of aggregating different subsets of the larger set of data. When rendered as a Dashboard Object, cubes allow you to quickly choose categories that "filter" data to show the results that meet your selection. For example, a cube of "sales" data might provide aggregations of the same data by product, by time, or by sales region dimensions. Looking at the cube you might choose to view the total sales of a product (Nails) within a business region (West) during a fiscal quarter (Q1):

```
West    Q1  January  Hardware  Nails    120,000
West    Q1  March    Hardware  Nails     98,000
                                        218,000 Total
```

Further, by quickly removing the product dimension specification, you switch the classification to see all sales for that region and quarter:

```
West    Q1  January  Hardware  Nails    120,000
West    Q1  March    Hardware  Nails     98,000
West    Q1  March    Hardware  Screws    97,000
West    Q1  January  Lumber    Studs    137,000
                                        452,000 Total
```

Or for all sales during the month of March:

```
West    Q1  March    Hardware  Nails     98,000
West    Q1  March    Hardware  Screws    97,000
East    Q1  March    Lumber    Plywood   92,000
South   Q1  March    Hardware  Nails     98,000
                                        385,000 Total
```

Or for all West region sales of the Lumber family of products:

```
West    Q1  January  Lumber    Studs    137,000
                                        137,000 Total
```

Note: See Example: Detail of a Cube Chart and Cube Tables in the *Using Dashboard* documentation for details about viewing and working with cube data.

➤ In this Chapter:

Measures

Measures are the central value that are aggregated and analyzed. In the above examples, *Total sales* is the aggregate value. In each example, the Total is measuring the sum of all sales in the set. Measures are built with the C-SQL Set functions, including SUM, AVG, MIN, MAX, STD_DEVIATION, and VARIANCE. For more information about measures, see "Measure Columns" on page 52.

Dimensions

A *dimension* is a ranked order of classifications that from the highest to lowest level each describe smaller, more distinct sets of related data. In the examples present in the introduction to this chapter, the business region is one level of a geographical dimension, the quarter and month columns are each levels of a time dimension, and the product family and product name are part of an inventory dimension. In the time dimension, months are smaller sets of fiscal quarters, just as product name is a smaller set of the product family level. The following tables presents some examples of dimensions.

Time	Geography	Inventory	Security	Taxonomy
year	continent	classification	type	kingdom
quarter	country	type	rating	phylum
month	region	manufacturer	company	class
week	state	model	cusip	order
day	county	configuration		family
hour	city			genus
minute	district			species

For information about dimensions, see "Dimensions" on page 69.

Note: You can limit user access to data in the cube with an access filter. For details, see "Access Filters" on page 15.

Creating Cubes

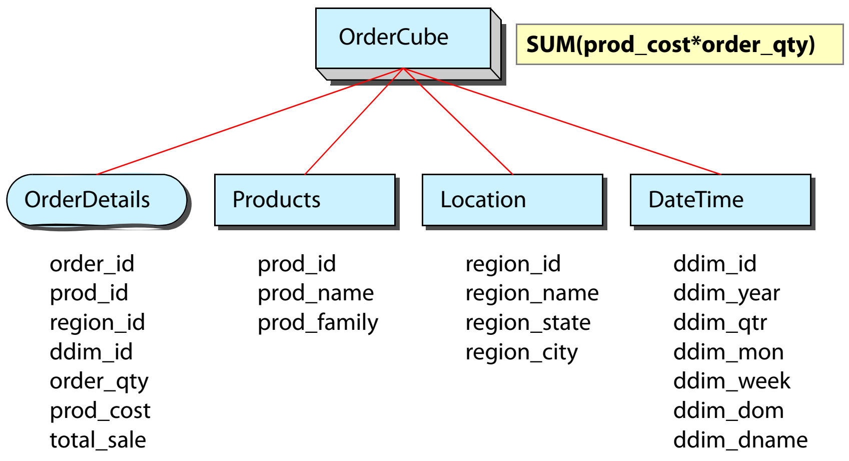
Cubes are similar to business views in that they aggregate event data, but they do so across different dimensions. The view that a cube aggregates is a *fact table*: a view or event table in an event stream that contains one or more columns to measure (aggregate), and which also contains columns that identify the dimensional elements associated with the event. For example, you could imagine a fact table containing an event similar to the following:

```
Cost   Quantity Product State      Month
------ -------- ------- ---------- -------
200.00     1600 Nails   California January
```

However, in practice the dimensional elements are stored in Dimensions (special context tables) and referenced by IDs, like this:

```
Cost   Quantity prod_id region_id ddim_id
------ -------- ------- --------- -------
200.00     1600     100         7      39
```

This illustration shows a cube built from the OrderDetails fact table and which measures total sales across various business regions, products, and time:



➤ **To define a cube:**

1. Choose the fact table and columns to measure.
2. Specify how to measure them (aggregate formulas to use).
3. Choose one or more dimensions that classify the measurements.

Measure Columns

Measure columns define the aggregations that the cube calculates. A cube must have at least one measure column, and may have more. Each measure column defines an expression that contains a C-SQL Set function that aggregates other columns from the fact table. For example, to determine the “total sales” from the OrderDetails fact table, a measure column might be defined as:

```
SUM(OrderDetails.prod_cost*OrderDetails.order_qty) AS TotalOrderSales
```

Dimension Columns

Dimension columns categorize the measurements. A cube must have at least one dimension, and may have more. Further, the data in the fact table must be able to identify a unique element in each associated dimension. For a complete discussion, see "Dimensions" on page 69. (Note that while the mathematical term "cube" implies three dimensions, a database cube can have any number from one or more.)

Prerequisites

Before creating a cube, you need:

- Create permission for Views, Cubes, and Dimensions.
- At least Read-Only access to an existing fact table (business view).
- At least Read-Only access to the Dimensions to include. See "Creating Dimensions" on page 74 for details.
- A dimension with Geo Categories enabled if the cube will be used for Geography charts.

Creation Steps

The following procedure describes how to create a cube.

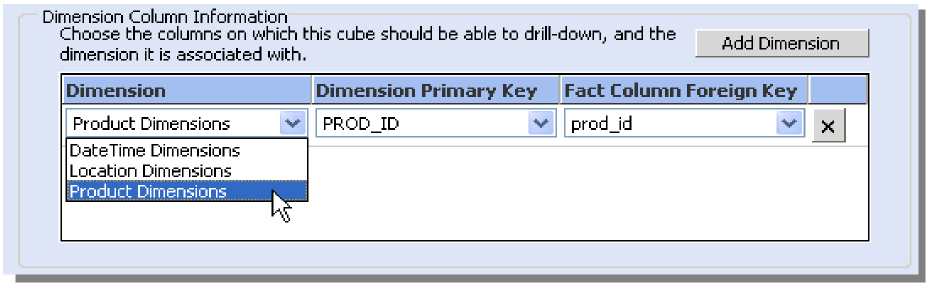
➤ **To create a cube:**

1. Open the Workbench tab of the BAM Workbench, select the Cubes folder, and click New Cube.
2. Identify a name, and optionally provide a description of the cube.
3. Choose the Fact Table that contains the data to measure, and which contains columns that identify the dimension elements.
4. Define one or more Measure Columns.
    - Click Add Measure Column to define a column.
    - Name the column in the Measure Name field.
    - Define the measure formula with a C-SQL Set function in the Aggregate Expression field. The function should reference a column from the fact table. For example, the following SUM() expression totals the product of the cost and quantity columns:

      ```
      SUM(OrderDetails.prod_cost*OrderDetails.order_qty)
      ```

      For more information about C-SQL expressions, see "Formulas" on page 108.
5. Define one or more Dimension Columns.
    - Click Add Dimension to define a column.

- Choose the dimension to include from the Dimension column drop-down list. This list includes all dimensions that you have at least Read-Only access to.



Dimension Column Information
Choose the columns on which this cube should be able to drill-down, and the dimension it is associated with.

Dimension	Dimension Primary Key	Fact Column Foreign Key
Product Dimensions	PROD_ID	prod_id

- Identify the key columns in the dimension and in the fact table. See "Key Columns" on page 73 for more details about the keys.

  The data type for the key in the fact table must be the same for the key in the dimension (context table). You cannot, for example, mix integer and decimal types; both must be either integer or decimal.

6. Save the cube and you can immediately begin building Dashboard Objects on top of it.

8 Data Types

Business Activity Monitoring and C-SQL supports the following SQL-99 data-types and their semantics, each of which is described in detail in the following sections of this documentation:

C-SQL Data Type	Data Type Category
BOOLEAN	Boolean
DECIMAL	Numeric
DOUBLE PRECISION	Numeric
INTEGER	Numeric
INTERVAL	Date-Time
TIMESTAMP	Date-Time
VARCHAR	String

C-SQL provides means for converting data of one type to another type. See "Data Type Conversion" on page 56 for details.

➤ **In this Chapter:**

Data Type Conversion

In Business Activity Monitoring there are two ways to convert values from one data-type to another: *explicit casting* and *implicit casting*.

Explicit Cast

Any C-SQL argument may contain CAST() to convert the data-type of a value. For example, you can cast a character string of numerals into a numeric value and use the result as an argument to FLOOR():

```
FLOOR( CAST( '1234.56' AS DECIMAL) )
```

Implicit Cast

C-SQL automatically attempts to convert a data-type to the correct type for the argument where the value is encountered. For example, if C-SQL encounters the VARCHAR '1234.56' in the FLOOR() argument, it automatically converts the value to a DOUBLE PRECISION numeric before truncating the decimal digits. For example:

```
FLOOR( '1234.56' )     << Implicit cast to DOUBLE PRECISION.
```

Similarly, when a value of one data type is compared to a value of different type, C-SQL first converts one of the values to match the other. In the following example, C-SQL converts the VARCHAR string to a BOOLEAN before evaluating the expression:

```
'true' = TRUE          << Implicit cast to BOOLEAN.
```

Context also affects casting. For example, because the following arithmetic add operator expects numeric arguments, and even though both values are characters, the values are first cast to numeric:

```
'2' + '3'          << Both cast to numeric to match operator data type.
```

Note: The value must be convertible to the required type or the expression will result in an *incorrect data-type* error.

Order of Precedence

The following table shows which types are convertible, and the order of precedence assigned to each possible data type conversion, where zero (0) is the highest precedence and a million (1,000,000) is the lowest:

To \ From	VARCHAR	BOOLEAN	TIMESTAMP	DOUBLE PRECISION	DECIMAL	INTEGER
VARCHAR	0	10	10	10	1,000,000	1,000,000
BOOLEAN	1	0	—	—	—	—
TIMESTAMP	2	—	0	—	—	—
DOUBLE PRECISION	3	—	—	0	1	3

To \ From	VARCHAR	BOOLEAN	TIMESTAMP	DOUBLE PRECISION	DECIMAL	INTEGER
DECIMAL	4	—	—	1	0	1
INTEGER	5	—	—	2	2	0

In the comparison example present in the previous section ("Implicit Cast"), C-SQL converts the VARCHAR to a BOOLEAN because the precedence level for that conversion is 1, as opposed to 10 for converting the BOOLEAN to a VARCHAR. Similarly, in the FLOOR("1234.56") example C-SQL converts the string to a DOUBLE PRECISION because DOUBLE PRECISION has a higher precedence than DECIMAL, even though a decimal might seem to be more appropriate to the value.

See the descriptions of the individual C-SQL data types for the specific details about converting those types.

Numeric

C-SQL has three data types for numeric values.

Type	Description	Bits	Minimum value	Maximum value
INTEGER	Signed integer	32	-2,147,483,648	2,147,483,647
DECIMAL	Decimal	—	1 digitsTotal count of DECIMAL digits, both before and after the decimal separator is 256.	256 digits Total count of DECIMAL digits, both before and after the decimal separator is 256.
DOUBLE PRECISION	IEEE 754 floating point	64	+/-4.94065645841246544E-324	+/-1.79769313486231570E-308

Total count of DECIMAL digits, both before and after the decimal separator is 256.

To express a DOUBLE PRECISION as a literal, use scientific notation, such as *1e24*.

Third party Data Types

The C-SQL numerics map to these data types in other support systems:

Support System	Numeric Data Types		
C-SQL/ JDBC	INTEGER	DECIMAL	DOUBLE PRECISION
Java	int	BigDecimal	double
Oracle	Number(*p*=38)	Number(*p*=38)	Number(*p*=38)

Support System		**Numeric Data Types**	

SQL-Server	Int(32 bit)	Decimal(*p*=38)
		Numeric(*p*=38)
		Money(64bit)
		SmallMoney(32bit)
			double
			real(4 bytes)

Sybase	Int(32 bit)	Decimal(*p*=38)
		Numeric(*p*=38)
		Money(64bit)
		SmallMoney(32bit)
			double
			real(4 bytes)

MySQL	TINYINT SMALLINT MEDIUMINT INT, INTEGER BIGINT	DECIMAL NUMERIC	DOUBLE REAL
PostgreSQL	SMALLINT INT, INTEGER BITINT	DECIMAL NUMERIC	DOUBLE PERCISION REAL
Where p is precision			

Combining Numeric Types

When combining two different numeric types, the result is the type with higher precedence based on the "Order of Precedence" on page 56. For example, adding a INTEGER to a DECIMAL results in a DECIMAL sum.

Casting a fractional number to an integer silently truncates the fraction (rounds down) to fit the target. For example, forcing a DOUBLE PRECISION into an INTEGER truncates the fractional part of the value.

Casting Numeric Types

Casting numerics to types of different storage size is permissible provided that the target is large enough to hold the result; otherwise the conversion fails with an "Number out of range" error. For example, attempting to put a floating-point type of a larger storage size into a location of a smaller size results in an error.

To Strings

When casting numerics to strings, be aware of the following:

- For DECIMAL numbers, the result is zero-padded in the decimal values to match the precision and scale defined for the column. So, for example, if a column is defined as precision 5 and scale 4, a value of 1.1 in the column is cast as '*1.1000*'.
- For DOUBLE PRECISION numbers, the 'e' is cast to upper case. So, for example, +1e11 is converted to '*1.0E11*'.

Decimal Precision Results

All decimal numbers have two components:

- Precision

  The count of digits, both to the left and right of the decimal point. The maximum is 256. The minimum is 1.

- Scale

  The count of digits of the fractional component. It is less than or equal to the precision. When no scale is specified, the default is 2.

In instances where a value has greater scale or precision than the target storage, such as a database field with a smaller precision, Business Activity Monitoring truncates decimals and rounds down the result to make it fit.

Casting

When casting a Decimal value, you can declare the precision and scale as follows:

```
DECIMAL( precision, scale )
```

For example,

```
CAST( '4.012345', DECIMAL(5,4) ) --> 4.0123
```

When casting from a decimal formatted column to a string, the result is zero-padded on the decimals to match the scale. For example, when column is precision 5 and scale 4, implicitly casting a value of 1.1 in the column to a string results in '*1.1000*'.

Multiplication

In multiplication, the resulting precision is the sum of the precisions, and the scale is the sum of the scales.

```
PrecisionResult = MIN( PrecisionLeft+PrecisionRight, 256 )

ScaleResult = MIN( ScaleLeft+ScaleRight, 256)
```

For example, the result of (*4.55*1.414*) is precision **7** (3+4) and scale **5** (2+3).

Division

In division, the results are:

```
ScaleResult = MIN(MAX((ScaleLeft+PrecisionRight-ScaleRight+1),2), 256)

PrecisionResult = MIN((PrecisionLeft + ScaleRight + ScaleResult), 256)
```

For example, the result of (4.55/1.414) is scale **4** (2+4-1) and precision **10** (3+3+4).

Addition and Subtraction

For addition and subtraction, the results are:

```
PrecisionResult = MIN( (MAX( PrecisionLeft - ScaleLeft,
                             PrecisionRight - ScaleRight) +
                       MAX( ScaleLeft, ScaleRight ) + 1), 256)

ScaleResult = MAX( ScaleLeft, ScaleRight )
```

All Other Functions

For other functions and operations, the result is determined by the value with the largest precision and the value with the largest scale — the results may be determined from the same value.

```
PrecisionResult = MIN( MAX( Precision[i] ), 256)

ScaleResult = MIN( MAX( Scale[i] ), 256)
```

String

The C-SQL VARCHAR data type maintains character string values.

Type	**Description**	**Padding**	**Minimum Size**	**Maximum Size**
VARCHAR	Variable length	No	1 character (default), may be null.	Infinite characters. Note that an error occurs if you attempt to store a value into a DBMS that is larger that the size of the column defined in the table.

String Width

Though the maximum size limit for string values is infinite, try not to exceed 255 characters because that is the limit imposed on many DBM systems. However, to improve performance, assist data storage, and aid in string compares, it is good to declare an appropriate maximum width when defining a VARCHAR column. The width should be big enough to hold the maximum length of any string result inserted into the field. Text that is longer than the maximum width will be truncated when the string is stored.

Third Party Data Types

The C-SQL/JDBC string type VARHCAR maps to these data types in other support systems:

Java	**Oracle**	**SQL-Server**	**Sybase**	**MySQL**	**PostGreSQL**
String	Char Varchar Varchar2(4k)	Varchar(8k)	Varchar(8k)	CHAR VARCHAR BINARY VARBINARY BLOB ENUM SET	CHAR[n] VARCHAR[n]

String Concatenation

To concatenate two strings, use either CONCAT() or the **||** operator. See "CONCAT" on page 122 for details.

String Literal

To express a String as a literal, enclose the text in single quotes ('). To include a single quote, include two; for example:

```
'Couldn''t'     Returns: Couldn't
```

Converting Strings to Other Data Types

When combining a string with another data type, or when expressing a string where another data type is expected, automatically converts the string to the new type based on the "Order of Precedence" on page 56. Additionally:

- All leading and trailing spaces are stripped.
- If the string contains an invalid character or invalid formatting, an error occurs. An invalid character is one that is inappropriate for the target data type. For example, 'hello' cannot be converted to an INTEGER.
- Formatting that is not consistent with the definition of a literal data value of the target type is invalid. For example, for a string to implicitly convert successfully to a TIMESTAMP data type, the source string must be contain in the default C-SQL date format. See See "Date-Time" on page 61. for details about formatting strings for date-time types.

Date-Time

Date-time data types store date and time-of-day of that date as a single value (a number). There is no facility for directly accessing a date-time as its internal, numeric representation. Instead, to access date-time values in a meaningful way, C-SQL provides several functions for manipulating the values, and provides literal constructs for representing the values in expressions.

Function	Description
TIMESTAMP Literal	A character string representation of a date-time value. Can be any combination of year, month, day-of-month, hour, minute, second, and fractional seconds.
INTERVAL Literal	A character string representation of an interval: a span of time comprised of years and months, or of days, hours, minutes, and seconds.

Note: Date-time values are in the time-zone of the locale of the server running Business Activity Monitoring.

Converting Between Date-Time and Strings

Convert a date-time to a character string (VARCHAR) with TO_CHAR(), and convert a string to a date-time with TO_DATE(). Both of those functions allow you to specify the format of the string.

Including a TIMESTAMP Literal in a string an argument automatically converts the value to a string using the default date-time format, which is "yyyy-MM-dd hh:mm:ss.SSS". For more information about converting between date-time and string values, see "Data Type Conversion" on page 56.

Comparing Date-Time Values

A date-time is stored internally as a number representing the date-time in milliseconds. As such, you need to be careful when comparing two date-time values. For example, this comparison is only true when both dates have exactly the same milliseconds:

```
first_date = second_date
```

If exact granularity is not important, consider first converting the date-time values to strings that represent just the date portion:

```
TO_CHAR(first_date,"yyyy-MM-dd") = TO_CHAR(second_date,"yyyy-MM-dd")
```

Note that according to the "Order of Precedence" on page 56, comparing a string to a date-time first casts the string to a date-time before the comparison occurs. Consider this example where birth_date is a date-time value. If birth_date has a time associated with it, the comparison will never be true:

```
'2003-02-18' = birth_date
```

A more exact comparison is to first cast birth_date to a string without a time:

```
'2003-02-18' = TO_CHAR( birth_date, "yyyy-MM-dd" )
```

Date-Time Arithmetic

The DATE_ADD() and DATE_DIFF() functions add and subtract intervals of years, months, days, hours, minutes, and seconds on date-time values. See the descriptions of those functions for details. Some query clauses, however, require a INTERVAL Literal (described below).

Third party data types

The C-SQL date-time type maps to these data types in other support systems:

Support System	Date-Time Type
C-SQL/JDBC	Date-time
Java	Date Time Timestamp
Oracle	Date(YMDHMS)
SQL-Server	Datetime(YMDHMS.xx) SmallDateTime (YMDHMS)
Sybase	Datetime (YMDHMS.xx) SmallDateTime (YMDHMS)
MySQL	DATE DATETIME TIMESTAMP TIME YEAR
PostgreSQL	TIMESTAMP

TIMESTAMP Literal

The TIMESTAMP literal represents a date-time value as a character string. To express as date-time as a literal value, prefix the data with the word “TIMESTAMP”, and enclose the entire data in single quotes ('), for example:

```
TIMESTAMP '2003-03-05 19:45:23.123'
```

The format of the string is “yyyy-MM-dd hh:mm:ss.SSS”, where S (the fractional seconds) are optional and may be from zero to nine digits of precision. See “Date-Time Formatting” on page 66 for details about the formatting characters.

INTERVAL Literal

An INTERVAL literal identifies a span of time comprised of years and months (*year-month intervals*) or of days, hours, minutes, and seconds (*day-time intervals*). You cannot combine year-month and day-time in one interval declaration. Intervals are applied to date-time values to calculate the a span of time from that instance. Typically they are used in expressions to offset date-time columns and TIMESTAMP literals, such as when declaring the range from a date or time in Query Windows. For example, the following query window totals of all events arriving in the last hour (implicitly applied to the arrival time of the latest event to arrive):

```
SUM(Qty) AS Total_Of_Qty OVER (RANGE INTERVAL '1' HOUR PRECEDING)
```

When applying an interval to a date-time, the interval is added to or subtracted from the value. For example, if the current date-time is 5 March 2003 at 7:45p.m., adding an interval of 1 year to that date results in 5 March 2004 at the exact same time. Note that calendar arithmetic follows Gregorian calendar rules—see "DATE_DIFF" on page 128 for details.

Year-Month Intervals

A year-month INTERVAL uses either, or combines both, of the date-time fields YEAR or MONTH. The possible definitions are:

```
INTERVAL 'yy'     YEAR[(<precision>)]
INTERVAL 'mm'     MONTH[(<precision>)]
INTERVAL 'yy mm' YEAR[(<precision>)] TO MONTH[(<precision>)]
```

The following examples define intervals of 3 years and of 10 months, respectively:

```
INTERVAL '3'  YEAR
INTERVAL '10' MONTH
```

You can define a fraction year interval by expressing the result in total months, such as 46 months, or by combining the field. For example, to identify an interval of 3 years and 10 months:

```
INTERVAL '3-10' YEAR TO MONTH
```

Note that you may specify a value of zero (0) for either field. The following iintervals are each 2 years:

```
INTERVAL '2-0'  YEAR TO MONTH
INTERVAL '1-12' YEAR TO MONTH
INTERVAL '0-24' YEAR TO MONTH
```

Precision of Year-Month Intervals

The <precision> argument is an ANSI standard that declares the maximum count of digits in the integer. By default, the <precision> is 2. As such, the following two declarations of 100 month intervals each fail:

```
INTERVAL '100' MONTH(2)  << ERROR, precision is less than value size.
INTERVAL '100' MONTH     << ERROR, default precision is 2.
```

To use more than 2 digits, declare a precision in the following manner:

```
INTERVAL '100' MONTH(3)
```

When using both fields, apply the precision on the YEAR field only; the MONTH field uses its default precision of 2. For example, the follow is erroneous because the month is greater than the default.

```
INTERVAL '100-123' YEAR(3) TO MONTH   << ERROR, month is 3 digits
```

Day-Time Intervals

A day-time INTERVAL is comprised of a combination of days, hours. minutes and seconds. The possible definitions are as follows, and where .nn is a fraction of a second:

```
INTERVAL 'dd'                  DAY
INTERVAL 'dd hh'               DAY TO HOUR
INTERVAL 'dd hh:mm'            DAY TO MINUTE
INTERVAL 'dd hh:mm:ss[.nn ]'   DAY TO SECOND
INTERVAL 'hh'                  HOUR
INTERVAL 'hh:mm'               HOUR TO MINUTE
INTERVAL 'hh:mm:ss[.nn ]'      HOUR TO SECOND
INTERVAL 'mm'                  MINUTE
INTERVAL 'mm:ss[.nn ]'         MINUTE TO SECOND
INTERVAL 'ss[.nn ]'            SECOND
```

Here are some examples of day-time intervals:

```
INTERVAL '27 23:59:59.999999999' DAY TO SECOND
INTERVAL '100 10:10' DAY(3) TO MINUTE
```

Precision of Day-Time Intervals

Each of the day-time fields also have a precision argument, such as:

```
MINUTE(<precision>)
```

The <precision> argument is an ANSI standard that declares the maximum count of digits in the integer. By default, the <precision> is 2 (except for fractional seconds whose default is 9, see below for details). As such, the following two declarations of 100 hour intervals each fail:

```
INTERVAL '100' HOUR(2)   << ERROR, precision is less than value size.
INTERVAL '100' HOUR      << ERROR, default precision is 2.
```

To use more than 2 digits, declare a precision in the following manner:

```
INTERVAL '100' HOUR(3)
```

When declaring precision for SECOND with a fractional component, specify two precision values separated by a comma. Consider the following examples:

```
INTERVAL '12.345' SECOND(2, 3)
INTERVAL '12.123456789' SECOND(2, 9)
INTERVAL '12.123456789' SECOND
```

Notice that the last two examples above have the same effect because the default is (2, 9) for SECOND.

When using multiple fields, express the precision on the first field only. The remaining fields use their default. For example, the precision in the following example applies to the minutes only and does not affect the fractional seconds:

```
INTERVAL '100:23.123456789' MINUTE(3) TO SECOND
```

Date-Time Formatting

The TO_CHAR() and TO_DATE() functions both have arguments that define the format of the date-time string. The format date pattern string is identical to the one used by the Java SimpleDateFormat class, which uses these letters in patterns:

Letter	Date-time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Fraction of a second (one S always returns an integer of 0 to 9 digits)	Number	978
'	escape for text	Delimiter	
''	single quote	Literal	'

Some letters have multiple results, depending on the number of consecutive letters in the format. The result will be the value that best fits the pattern. For numbers, if the pattern is bigger than the value, the result is padded with leading zeros (0). See the examples in Table 1: Date and Time Pattern Examples for details.

The examples in the following table, modified from the Java SimpleDateFormat class documentation, show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-08-04 12:08:56 local time in the U.S. Pacific Time time zone.

Table 1: Date and Time Pattern Examples

Date and Time Pattern	Result
'd M yy'	4 8 01
'dd MM yy'	04 08 01
'ddd MMM yyy'	004 Aug 2001
'dddd MMMM yyyy'	0004 August 2001
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.08.04 AD at 12:08:56 PDT
"EEE, MMM d, ''yy"	Sat, Aug 4, '01
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.August.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Sat, 4 Aug 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700

Boolean

C-SQL follows the SQL-99 use of three-valued logic (TRUE, FALSE, and UNKNOWN) to support NULL value semantics. For example,

```
WHERE OnSale IS TRUE
WHERE (Age >= 21) IS UNKNOWN
```

When using Boolean operators to evaluate the truth of an expression, the values are evaluated as described in the following truth tables follows:

Truth Table for NOT

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

Truth Table for AND

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

Truth Table for OR

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Truth Table for IS

IS	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE
UNKNOWN	FALSE	FALSE	TRUE

Note: TRUE is greater than FALSE in comparisons.

9 Dimensions

Dimensions are ranked orders of related data and are used by Cubes to categorize measurements. These measurements are ranked by level and value.

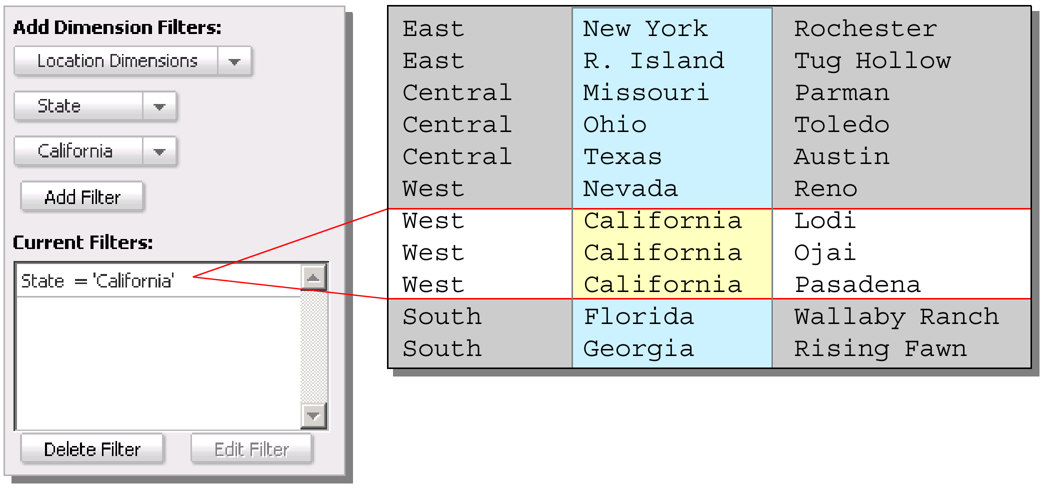
➤ **In this Chapter:**

What are Dimensions?

A *dimension* is a ranked order of classifications that, from highest to lowest level each describe decreasingly smaller sets of related data. The following table presents some examples of dimensions where the top level of each list contains the largest set of related items, while the bottom contains the smallest, most specific set:

Time	Geography	Inventory	Security	Taxonomy
year	continent	classification	type	kingdom
quarter	country	type	rating	phylum
month	region	manufacturer	company	class
week	state	model	cusip	order
day	county	configuration		family
hour	city			genus
minute	district			species

Cubes categorize measurements by dimensions, and within dimensions by levels and values. For example, a location dimension can filter the results to show the measurements for all cities in a state. The illustration below shows several locations filtered by state name, which limits the measurements to just three cities:



East	New York	Rochester
East	R. Island	Tug Hollow
Central	Missouri	Parman
Central	Ohio	Toledo
Central	Texas	Austin
West	Nevada	Reno
West	California	Lodi
West	California	Ojai
West	California	Pasadena
South	Florida	Wallaby Ranch
South	Georgia	Rising Fawn

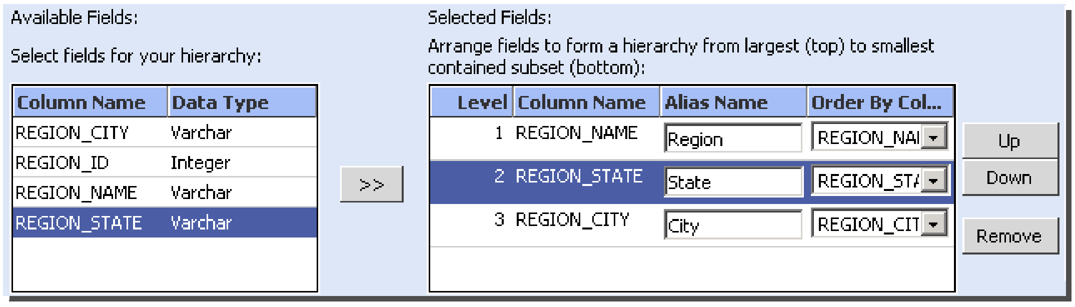
Dimensions draw their values from context tables, where each dimension level is one column in the table, and each row is a unique dimension element. In the example shown in the illustration above, the region names are in one column, states in another, and cities in a third. When you create a dimension, you identify the existing source context table and the columns to include, and you arrange the columns into the level hierarchy.

Level Hierarchy

The level hierarchy is what enables “roll-up” and “drill-down” in cubes. When a user is viewing data for one level, they can “roll-up” to see a higher level of measurements, or “drill-down” to see the data categorized at the next smaller level. For example, when looking at the results for a state, the user might choose to roll-up to see the measurement for all states in the region. Alternatively, the user might click on the results to see the results for each city in the state. The level hierarchy defines the levels in the dimension.

The following illustration shows four columns in the source file, but only three contain dimension level data. These three are selected and arranged in the containing hierarchy of largest to most-specific levels.

Note: If you have selected "Use this for Geography Charts", a Geo Categories column is displayed. For information about Geo categories, see "Geo Categories" on page 72.



Available Fields:

Select fields for your hierarchy:

Column Name	Data Type
REGION_CITY	Varchar
REGION_ID	Integer
REGION_NAME	Varchar
REGION_STATE	Varchar

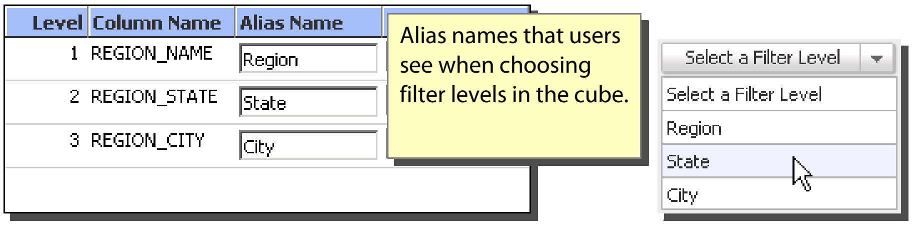
Selected Fields:

Arrange fields to form a hierarchy from largest (top) to smallest contained subset (bottom):

Level	Column Name	Alias Name	Order By Col...
1	REGION_NAME	Region	REGION_NAI
2	REGION_STATE	State	REGION_STA
3	REGION_CITY	City	REGION_CIT

Alias Names

After identifying the columns, you can optionally assign alias names that the users will see when they work with the cube. This illustration shows the filter level alias names as they appear in the Dashboard.

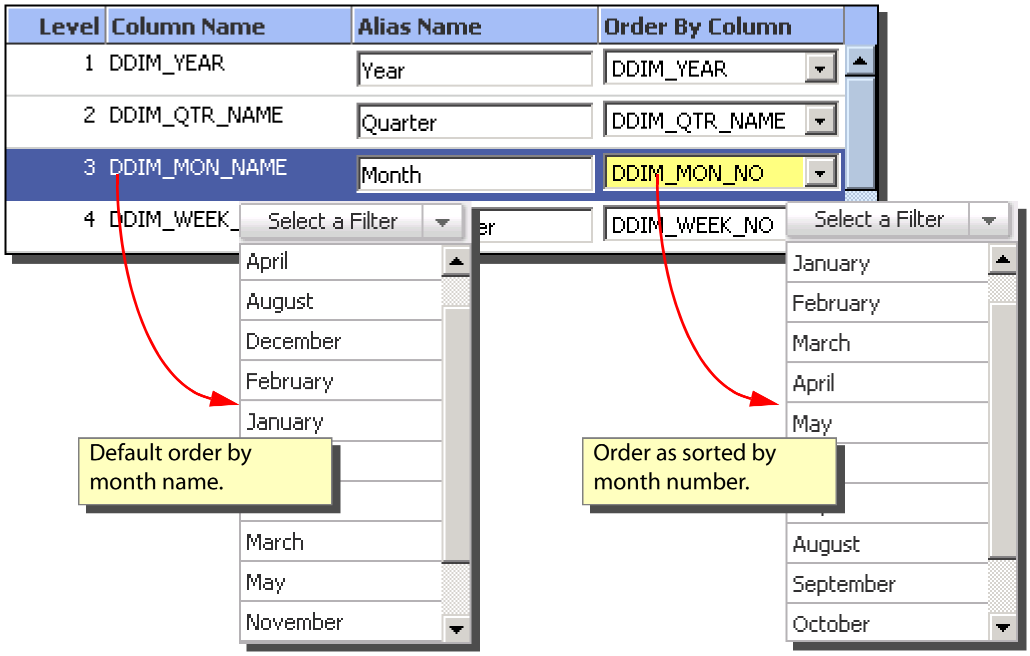


Level	Column Name	Alias Name
1	REGION_NAME	Region
2	REGION_STATE	State
3	REGION_CITY	City

Alias names that users see when choosing filter levels in the cube.
Select a Filter Level
Select a Filter Level
Region
State
City

Order By

In the BAM Dashboard, dimension values are presented in their sort order (as provided by the server). For example, a list of month names appears in alphabetical order, starting with April and ending with September, rather than in the order they occur in a year. To specify another order, use the Order By Column field. This field identifies another column that contains the values to use for sorting. For example, instead of using the “month name” column, use the “month number” column, as shown in the following illustration.



Level	Column Name	Alias Name	Order By Column
1	DDIM_YEAR	Year	DDIM_YEAR
2	DDIM_QTR_NAME	Quarter	DDIM_QTR_NAME
3	DDIM_MON_NAME	Month	DDIM_MON_NO
4	DDIM_WEEK_	er	DDIM_WEEK_NO

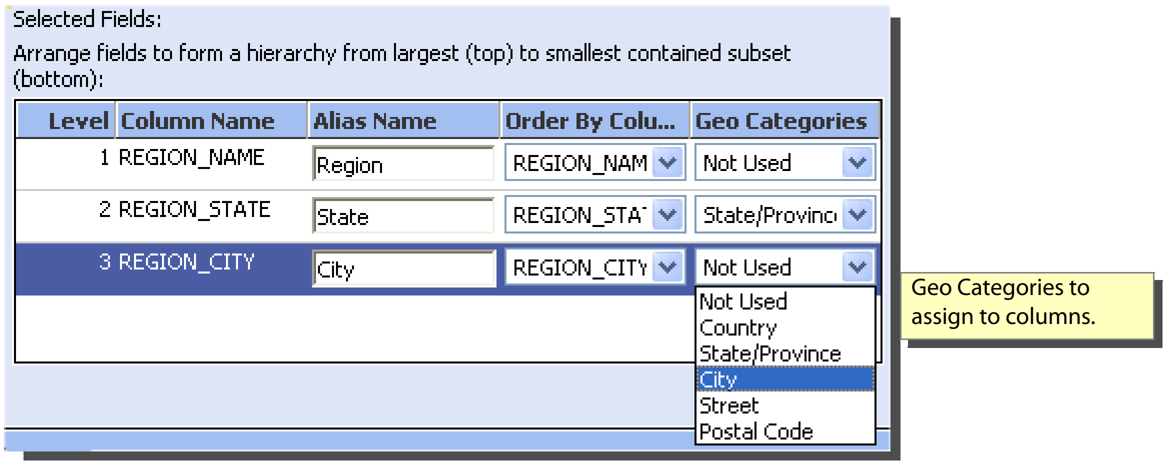
Also, if your Order By column field has multiple values that correspond to a single value in the dimension, the miminum value of the Order By column will be used for the sort order. In such cases, you should be careful when selecting the column to use for sorting values. For example, using month names as the sort order for quarters could produce unpredictable results because the months would be sorted by alphabetical rather than numerical order.

Geo Categories

Geo Categories enable geographic information to be used in cubes. A geo category setting specifies the level of accuracy that the BAM Dashboard should use for the column. For example, if a column contains the names of states or provinces, you would assign it the geo category “State/Province.” The available categories are:

- Not Used
- Country
- State/Province
- City
- Address
- Postal Code

Use the “Not Used” category for a column that you do not want to map on a Geography chart or does not match an available category. For example, a column of REGION_NAME that contains data such as North, South, East, and West cannot be mapped to a geographic location, and you would assign this column to the “Not Used” geo category.



Selected Fields:
Arrange fields to form a hierarchy from largest (top) to smallest contained subset (bottom):

Level	Column Name	Alias Name	Order By Colu...	Geo Categories
1	REGION_NAME	Region	REGION_NAM	Not Used
2	REGION_STATE	State	REGION_STA	State/Provinc
3	REGION_CITY	City	REGION_CITY	Not Used

Key Columns

To use the context as a dimension, your event data must identify the unique dimension element (row) that it belongs to. In the location context data, the most unique value in each element is the city name; however, while that name could be used for identification purposes, for performance reasons it is better to use a number. As such, the locations context data should have an integer ID as follows:

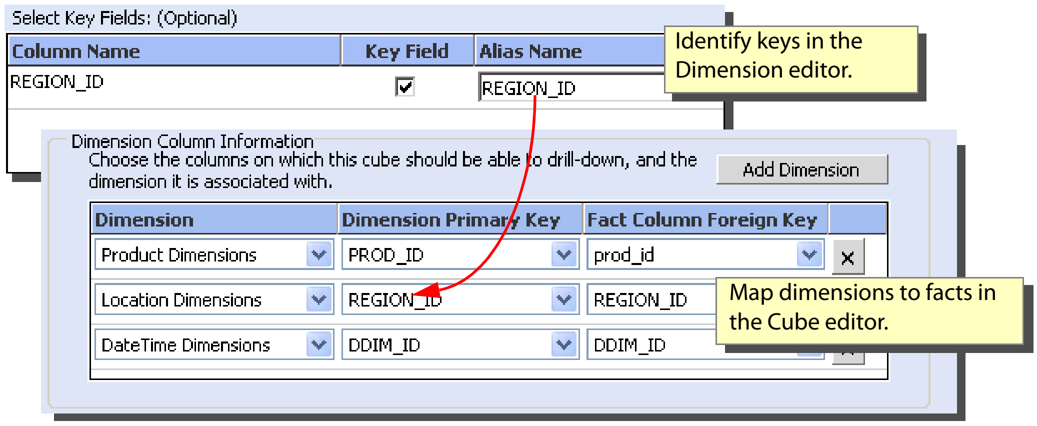
```
region_id  region_name region_state  region_city
---------  ----------- ------------  ------------
        1  West        Nevada        Reno
        2  South       Texas         Austin
        3  East        New York      Rochester
        4  Central     Ohio          Toledo
        5  West        California    Pasadena
      ...
```

Then the fact table that provides the value for the cube to measure also includes the key value to identify the associated dimension. For example, this order record is associated with Ojai, California, in the West region:

```
order_id region_id total_sales ...
-------- --------- -----------
  102341         7      120000 ...
```

Note: The data type for the key in the fact table must be the same for the key in the dimension (context table). You cannot, for example, mix integer and decimal types; both must be either integer or decimal.

When you define the dimension, identify one or more key fields that may be used to identify the specific level. You can choose any column that is not already a dimension level because levels are automatically assumed to be potential keys. The illustration below shows one column, REGION_ID, because all the other columns in the dimension are assigned to levels. Later, when you define the cube, identify the key column in the dimension that maps to the key in the fact table. In this example, they happen to have the same name; however, that is not a requirement. Furthermore, you may assign an alias name to the column to make it easier to identify.


Identify keys in the Dimension editor.
Map dimensions to facts in the Cube editor.
Add Dimension

Select Key Fields: (Optional)

Column Name	Key Field	Alias Name
REGION_ID	✓	REGION_ID

Dimension Column Information
Choose the columns on which this cube should be able to drill-down, and the dimension it is associated with.

Dimension	Dimension Primary Key	Fact Column Foreign Key
Product Dimensions	PROD_ID	prod_id
Location Dimensions	REGION_ID	REGION_ID
DateTime Dimensions	DDIM_ID	DDIM_ID

For information about defining cubes, see "Creating Cubes" on page 52.

Creating Dimensions

Before creating a dimension, you need:

- Create permission for Views, Cubes, and Dimensions.
- At least Read-Only access to the Context table that provides the dimension elements.

  Note: Turn on caching for the context table for optimum performance. When caching is off, performance for cubes can be slowed dramatically. See "Caching Context Queries" on page 48 for information about controlling the cache.

➤ **To create a dimension:**

1. Open the Workbench tab of the BAM Workbench, select the Dimensions folder, and click New Dimension.
2. Identify a name and optionally provide a description of the dimension.
3. Choose the Context Table that contains the dimension elements.
4. (Optional) Select "Use this for Geopgraphy Charts" if you plan to implement geography maps. This adds a Geo Categories column to the Selected Fields list.
5. Define the levels of the hierarchy:
   - Add fields to the hierarchy from the Available Fields list.
   - Order the levels from largest set (top) to smallest (bottom).
   - Identify one or more Key Columns to include.
   - Optionally assign Alias Names to the levels.

- Optionally assign Geo Categories to each level, if you selected the "Use this for Geography Charts" check box. The possible categories are:
  - Not Used
  - Country
  - State/Province
  - City
  - Street
  - Postal Code

Save the dimension, and you can immediately use it in Cubes.

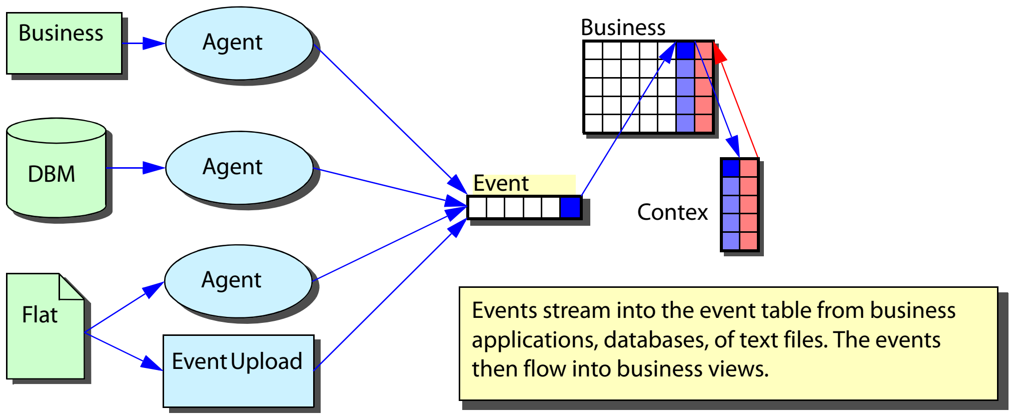
10 Events

Events drive Business Activity Monitoring internal processing. Events are data produced by external business applications that record transactions, identify changes in business state, and synthesize the details about the business activities. Business Activity Monitoring receives events in *event tables*. Business views built on the tables then aggregate the event information and drive the Rules that look for exceptional business conditions.

➤ **In this Chapter:**

How It Works

Events come to Business Activity Monitoring from business applications, databases, and text files. Usually Agents automatically receive or retrieve the event data and load it into an event table. Alternatively, you can manually load events from text files with the BAM Workbench. As events arrive they are processed and their data are passed to the business views. The views then aggregate the data and might retrieve context data relative to the event.


Business
Agent
DBM
Agent
Flat
Agent
Event Upload
Event
Business
Contex
Events stream into the event table from business applications, databases, of text files. The events then flow into business views.

Note: Events are processed in the order that they are received in the system. When one agent has received a large quantity of events, any new events received by other agents are queued behind the first set and are not processed until the first set is completely processed.

External Sources

Business Activity Monitoring provides agents to access several external event sources. Some events stream (are pushed) into the system as they happen. Other events are loaded (pulled) as the result of a request, such as from a database or a text file. The following table summarizes the available sources and identifies the source agent they support. (See "Agents" on page 23 for details about how they retrieve and receive event data.)

External source	Agent	Event push	Event pull
Java Messaging Service (JMS)	Java Messaging Service (JMS) (see page 204)	Yes	No
Text file (XML or flat)	Flat Files (see page 81)	No	Yes
TIBCO Rendezvous (RV)	TIBCO Rendezvous (see page 337)	Yes	No
HTTP Post action	HTTP Post (see page 197)	Yes	No
Salesforce	Salesforce (see page 305)	No	Yes
Web service	None. (See "Web Service Events" on page 363.)	Yes	No

Event Properties

Every event table has a name, description, and status attribute, and most have an agent. The following table describes the attributes.

Attribute	Description
Name	Identifies the table and is the name accessed by the Business Views that depend on this table. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	(*optional*) Description of the table.
Status	Whether or not the object is enabled (able to receive and pass data), or disabled (not receiving or passing data).
Agent	An agent that receives or retrieves the event information, and passes the data to the event table. See "Agents" on page 23 for information about agent types.
Log event data for recovery	When on, logs event data that arrived after the last checkpoint started. This "recovery" log is used to restore the state of the system in the event of an abnormal shutdown of the servers. See Working with Checkpoint and Recovery for complete details.
Process events in the order of arrival	Choose this option when events *must* be processed in the order received. When off, events may be processed out of order. **Note:** To join events in a view, the events must be processed in order: leave this option off to join the events.

Creating Event Tables

Before creating an event table, you must have Create permission for tables (see “Creating Permission” on page 258), and Read only access permission on the agent that will feed the table.

➤ **To create an event table:**

1. In the Workbench tab of the BAM Workbench, click New Event…
2. Choose the source type.
3. Each type has its own specific attributes. For details, see the following related sections:
    - “Flat Files” on page 81
    - “HTTP Post” on page 197
    - “Java Messaging Service (JMS)” on page 204
    - “JDBC” on page 216
    - “Salesforce” on page 305
    - “TIBCO Rendezvous” on page 337
    - “Web Services” on page 362
4. Fill in the fields in the New Event or New Context form.
5. Save the object as enabled.

    The object is immediately ready to receive events or context.

Editing Event Tables

Editing the attributes of an event table causes the object to lose state, and possibly invalidates dependant views. For example, if you remove a column, any view or rule that references that column becomes invalid. (However, if you redefine the column in the table, the dependant views are automatically revalidated.)

Before editing an event table, you must have Read and wrIte permission for tables (see “Accessing Permissions” on page 255), and Read only access permission on the agent that feeds the table.

➤ **To edit an event table:**

1. In the Workbench tab of the BAM Workbench, select the event or context object.
2. Chose Edit This Event or Edit This Context.
3. Change the definitions in the Edit Event form.

    Each type of event has its own specific attributes. For details, see the following sections:
    - “Flat Files” on page 81
    - “HTTP Post” on page 197
    - “Java Messaging Service (JMS)” on page 204
    - “JDBC” on page 216
    - “Salesforce” on page 305
    - “TIBCO Rendezvous” on page 337

- “Web Services” on page 362

Save the object as enabled and it will immediately be ready to receive events or context.

11 Flat Files

A flat file is a text file that contains the information about one or more events. Each line in the file is usually one event record — one row in the event table — and the data in the row map into the columns in the event table. The rows may be formatted as fixed width, delimited, or XML files.

➤ **In this Chapter:**

Note: Fixed-width and delimited files may contain multiple rows for the same event. See "Multi-Row Events" on page 84 for details.

How It Works

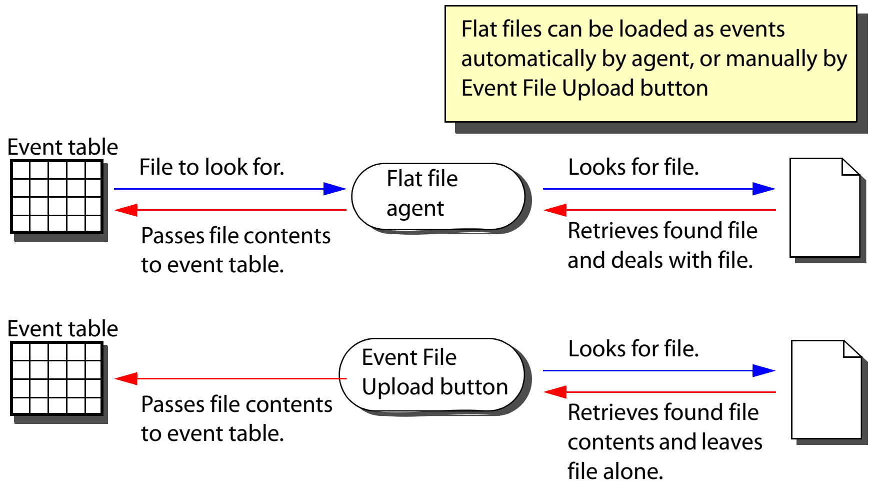
Flat file events are loaded in "batch" mode into the flat-file event table, though events are processed individually as they are loaded into the table. There are two ways to load flat-files into the event tables:

- Automatically

  The Flat File Agents periodically looks to see if the associated file exists. When the file is found, the agent retrieves it and passes it to the event object for event processing.

- Manually

  The Upload Event File button in the event-detail page in the BAM Workbench loads a manually selected file when chosen. For details, see "Upload Event File Option" in . *Using Business Activity Monitoring Workbench*.


Flat files can be loaded as events automatically by agent, or manually by Event File Upload button
Event table
File to look for.
Flat file agent
Looks for file.
Passes file contents to event table.
Retrieves found file and deals with file.
Event table
Event File Upload button
Looks for file.
Passes file contents to event table.
Retrieves found file contents and leaves file alone.

Flat File Event Tables

Flat Files event tables receive event files from Flat File Agents or from the Upload Event File Option in the BAM Workbench of the BAM Workbench. The three flat-file formats are:

- "Delimited Files" on page 93
- "Fixed-Width Files" on page 94.
- "XML Files" on page 95

Before creating an event to a flat-file event table, you need:

- Permissions — Create permission for tables (see "Creating Permission" on page 258), and Read Only access permission on the agent that will feed the table.
- Fixed-width and delimited files — (*optional*) A sample file that contains data in the format of the actual events. Use the sample when you create the event to ensure that the fields map correctly into the event table. Note that this is optional; you can create the event without a source, but having it greatly assists you with event table creation.

- The schema of the XML files to load. Specifically, you need to know the names of the attributes that contain the event column information, and the XML path to the element that contains the columns for each event. See "XPaths" on page 95 for details. The following table describes the attributes.

Attribute	**Description**
Name	Identifies the event object. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the event object is enabled (monitoring for events) or disabled (not monitoring for events).
Log event data for recovery	When on, logs event data that arrived after the last checkpoint started. This "recovery" log is used to restore the state of the system in the event of an abnormal shutdown of the servers. See Working with Checkpoint and Recovery for complete details. **Note:** To improve input/output performance, point the recovery log file directory to a disc different from the one that feeds this agent.
Process events in the order of arrival	**Note:** Choose this option when events must be processed in the order received. Otherwise, if events may be processed out of order, turn this on. To join events in a view, the events must be processed in order: Leave this option off to join the events.
File Agent	An existing file agent that retrieves events and context from a text file. Create an agent with the BAM Workbench Administration tab. See "Flat File Agents" on page 89 for details.
Start import on row	For fixed-width and delimited files, identifies the row of the text in the source file that contains the first data to import. Default is 1, the first row. Use this option if the text contains unnecessary introduction or header information.
Use this row for column names	For fixed-width and delimited files when using a sample, this option identifies a row in the sample that includes the column names. These names identify each column in the Column Information details. When this option is not specified, the default names are Field1, Field2, etc.
Skip rows	For fixed-width and delimited files identifies the rows to ignore in the source file before importing event data. For example, if the file contains some title and header information, the source might actually start on the third row of the file. In such a case, you would specify 2 as the count of rows to skip.
Allow short rows	For delimited files only. See "Delimited Files" on page 93 for details.
Delimiter	For delimited files only. See "Delimited Files" on page 93 for details.
Escape character	For delimited files only. See "Delimited Files" on page 93 for details.
Text qualifier	For delimited files only. See "Delimited Files" on page 93 for details.
Number formats	Number formatting specifications. Default is comma (,) thousands separator and dot (.) decimal separator.

Attribute	Description
Column information	For fixed-width and delimited files details about each column in the table, including the name, data type, and formatting applicable to the type.
Event Key	For fixed-width and delimited files, identifies key field columns for multi-row events. See "Multi-Row Events" on page 84 for details.
Field information	For XML files, identifies the source elements and how they map into the event table columns. See "XML Field Information" on page 96 for details.
Clear State Interval	This tab contains several options for clearing persisted event data that is propagated from the event in the views, objects, and dashboards that depend on it. The three options are: • Do Not Clear State— This is the default. Data persists. • Clear State on a Schedule— Select to clear the state on a schedule. Selecting this option activates the scheduling feature positioned to the right. • Always Clear State (Every Event)— The state is refreshed each time the event is updated.

Creating a Flat-File Source Event

Use the following procedure to create a flat-file source event.

➤ **To create a flat-file source event:**

1. Open the Workbench tab in the BAM Workbench.
2. Choose New Event and select Flat-file as the source type.
3. *(Optional for fixed-width and delimited files.)* Identify a sample file to assist in mapping the columns. This file is a sample of the real data file. Data from this file appears in the next step to assist you as you map the event data into the table.
4. For fixed-width files, define the positions of the data columns with the Set Field Widths dialog.
5. Identify the event Attribute.
6. Define the format-specific Column Information. For details about the source type, see:
   - "Delimited Files" on page 93
   - "Fixed-Width Files" on page 94.
   - "XML Files" on page 95

Save the file source as enabled and it will immediately be ready to receive event messages.

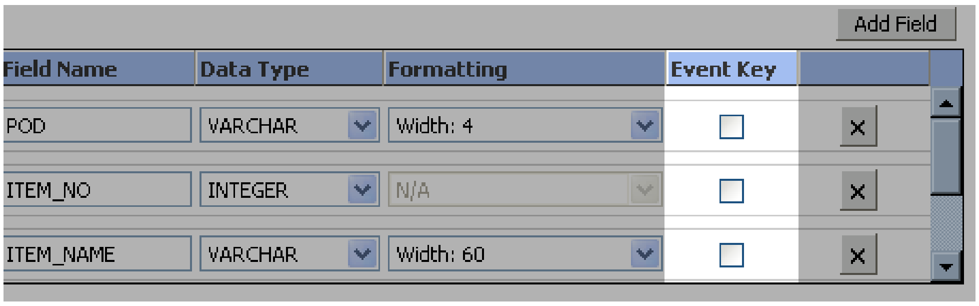
Multi-Row Events

A fixed-width or delimited file may contain multiple rows for the same event. For example, a "purchase order" event might contain one row for each line-item in the order. When this event is loaded into the event table, each row is treated as part of the same event; the system does not treat each row as a new event.

To identify the rows as containing data for the same event, each row must have some identifying data that is unique to the event. For example, this sample data contains line items for 3 purchase orders where each order identified by the POID column. The first order has 3 items, the second has one, and the third has 2:

```
POID,ITEM_NO,ITEM_NAME,ITEM_QTY,ITEM_COST,ITEM_TOTAL
0697,1,Smoke Shifter,100,5.00,500.00
0697,2,Nano Webber,50,6.00,300.00
0697,3,Locking Rail Key,25,7.50,187.50
0698,1,Nano Webber,50,6.00,300.00
0699,1,Foo Bar Stool,100,60.00,6000.00
0699,2,Can of Levers,250,1.50,375.00
```

When defining the column information for this event, you identify the POID column as the key field by choosing Event Key. Each event may have one or more Event Key fields, as shown in the following illustration.



Field Name	Data Type	Formatting	Event Key
POD	VARCHAR	Width: 4	☐
ITEM_NO	INTEGER	N/A	☐
ITEM_NAME	VARCHAR	Width: 60	☐

In the file, the rows for each event must appear together, and the data in the Event Key fields must be unique to the event. As soon as the data in one of the fields is not the same as the previous row, that field's row is a new event. For example, the following sample is treated as three separate events, even though the last row has the same Event Key value as the first two rows:

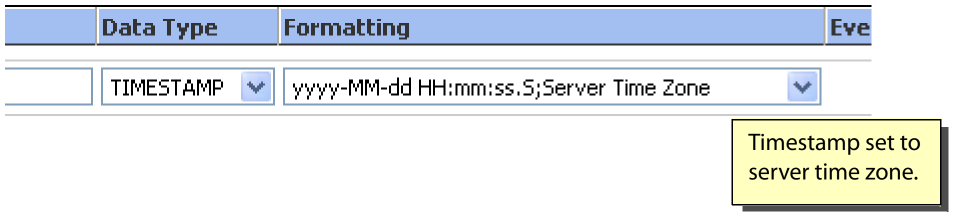
```
POID,ITEM_NO,ITEM_NAME,ITEM_QTY,ITEM_COST,ITEM_TOTAL

0697,1,Smoke Shifter,100,5.00,500.00

0697,2,Nano Webber,50,6.00,300.00

0100,1,Foo Bar Stool,100,60.00,6000.00

0697,3,Locking Rail Key,25,7.50,187.50
```

Note: If any row contains invalid data, that row is discarded and does not affect subsequent rows. For example, the third row in the following sample contains a character ('X') where an integer is expected. In this sample, the third row is discarded, and the fourth *is included* as the third row in the event:

```
POID,ITEM_NO,ITEM_NAME,ITEM_QTY,ITEM_COST,ITEM_TOTAL
0697,1,Smoke Shifter,100,5.00,500.00
0697,2,Nano Webber,50,6.00,300.00
0697,X,Foo Bar Stool,100,60.00,6000.00
0697,4,Locking Rail Key,25,7.50,187.50
```

Time Zones for Flat File Events

Flat File events (delimited and fixed) have the option to specify a date format for a timestamp field. In addition to the date format, you can also specify a time zone. This allows you to specify how the server interprets the timestamp value in the field; that is, the timestamp can be assumed to be in the same time zone as the application server or another specific time zone. This is useful when the application server is a different time zone from the one in which events occur, and you want the timestamp to display in a View for that time zone. The default is the server's time zone.


Data Type
Formatting
Eve
TIMESTAMP
yyyy-MM-dd HH:mm:ss.S;Server Time Zone
Timestamp set to server time zone.

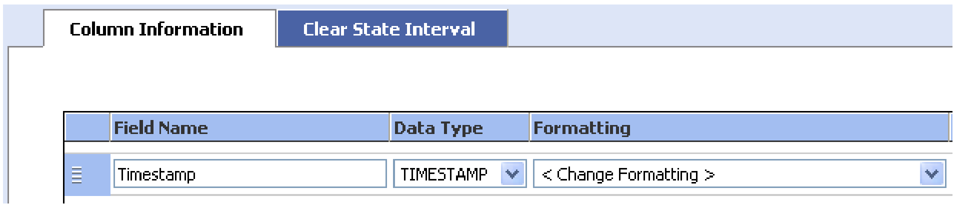
The following table shows examples of a timestamp with different formats and time zones. Note that when the time zone is set to GMT+5:30 and GMT+13:00, the View result for the timestamp shows the date as 2006-10-25 instead of the field value.

Field Value in Flat File	Timestamp Format	View Result
2006-10-26 09:49:12.976	yyyy-MM-dd HH:mm:ss.S; Server Time Zone	2006-10-26 09:49:12.976
2006-10-26 09:49:12.976	yyyy-MM-dd HH:mm:ss.S; Server Time Zone	2006-10-26 09:49:12.976
2006-10-26 09:49:12.976	yyyy-MM-dd HH:mm:ss.S; (GMT+5:30) Chennai, Kolkata,Mumbai ...	2006-10-25 21:19:12.976
2006-10-26 09:49:12	yyyy-MM-dd HH:mm:ss; (GMT+13:00)Nuku'alofa	2006-10-25 13:49:12.0

➤ **To set the format and time zone for a flat-file event:**

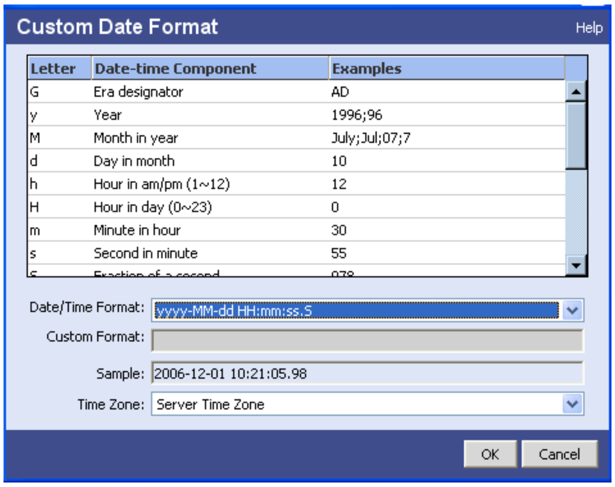
1. Click the Edit This Event... tab for the flat-file event.

2. In the Column Information tab, find the field with a TIMESTAMP data type and select <Change Formatting> under Formatting.



Field Name	Data Type	Formatting
Timestamp	TIMESTAMP	< Change Formatting >

The Custom Date Format window displays.



Letter	Date-time Component	Examples
G	Era designator	AD
y	Year	1996;96
M	Month in year	July;Jul;07;7
d	Day in month	10
h	Hour in am/pm (1~12)	12
H	Hour in day (0~23)	0
m	Minute in hour	30
s	Second in minute	55
S	Fraction of a second	978

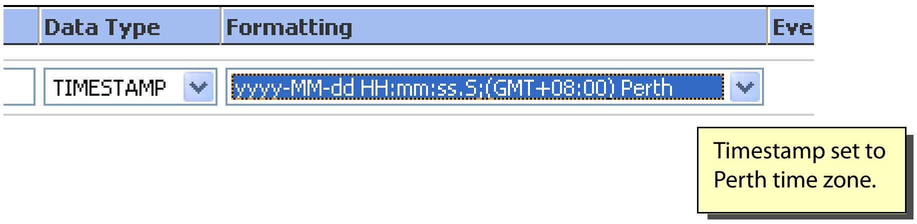
Date/Time Format: yyyy-MM-dd HH:mm:ss.S
Custom Format:
Sample: 2006-12-01 10:21:05.98
Time Zone: Server Time Zone

3. Select a format from the Date/Time Format drop-down menu or create a custom format.

4. Select a time zone from the Time Zone drop-down menu.

5. Click OK.

The timestamp format and time zone displays in the Formatting column of your timestamp field.



Data Type	Formatting	Eve
TIMESTAMP	yyyy-MM-dd HH:mm:ss.S;(GMT+08:00) Perth	

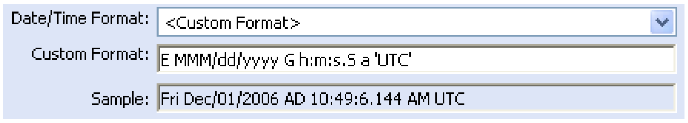
➤ To create a custom time format:

1. Open the Custom Date Format window.

2. Select <Custom Format> from the Date/Time Format drop-down menu.

3. Build your custom format in the Custom Format field by entering date-time component letters.

   Use the list displayed in the date format list for the available definitions. The Sample field will display a sample of your format as you build it.

   Date/Time Format: <Custom Format>
   Custom Format: E MMM/dd/yyyy G h:m:s.S a 'UTC'
   Sample: Fri Dec/01/2006 AD 10:49:6.144 AM UTC

4. Click OK to apply the custom format to your timestamp field.

Flat File Agents

A flat file agent retrieves event messages from a text file.

Note: You *cannot* retrieve context from a file agent because there is no query logic as summarized in the following table

Event push	Event pull	Context pull
No	Yes	No

File Processing

The flat file agent searches for files in a specified location on a defined interval. The name of the files to search for may include * and ? wildcard characters. When the agent locates a file, it retrieves the events and then either deletes, moves, or renames the source file. When multiple files are located in the named location, the agent processes them in filename order.

Prerequisites

Before creating a flat file agent, you need:

- Create permission for agents (see "Creating Permission" on page 258 for details).
- A running File agent program (see "Configuring the File Agent Program" on page 90)

A file agent has the following attributes:

Attribute	Description
Name	Identifies the agent and is the same name as defined by the agentName element in the agent's TestAgent.xml configuration file. See "Configuring the File Agent Program" on page 90 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the agent is enabled (monitoring for events), or disabled (not monitoring for events).

Creating a Flat File Agent

Use the following procedure to create a flat file agent.

➤ To create a flat file agent:

1. Open the BAM Workbench Administration Console.
2. Click New Agent…
3. Choose Flat-file as the source type
4. Fill in the fields that define the agent's attributes.

Save the agent as enabled and it will immediately begin monitoring for events.

Configuring the File Agent Program

The File Agent is a stand-alone Java program that runs on a host (possibly different than the BAM Server host), gathering events from a text file. When it finds event data, it passes the data to the BAM Server for processing.

The agent has two XML configuration files:

- TestAgent.xml (based on VCAgent.xsd) defines the connection information, such as how to locate the BAM Server and how those servers can locate the file agent.
- FileAgent.xml (based on FileAgent.xsd) identifies the text file and what to do with the file when finished uploading its data.

To start the agent, run the cqagent.jar file in Java and pass the TestAgent.xml configuration file as an argument as follows:

```
java -jar …\cqagent.jar TestAgent.xml
```

Optionally, you can identify the logging configuration file directory and logging level by including logging properties. The following example sets the logging level to all messages:

```
java "-Dcom.celequest.property.Logging Directory=C:\logs\agents"
     "-Dcom.celequest.property.Detailed Log File Level=All"
      -jar …\cqagent.jar TestAgent.xml
```

TestAgent.xml

The TestAgent.xml file has the following configuration attributes and elements:

Attribute	Description
serverPort (attribute)	(Optional: default 80) HTTP port on the application server that is running the BAM Server, and which the agent uses to communicate to the server. This is the same port that users use to connect to the BAM Workbench.
pingInterval (attribute)	(Optional: default 20 seconds) How often the agent tests to see if the BAM Server is running. When the server is not running, the agent will not gather events.
agentName	(Required) Identifies this agent and is the same Name to use when creating the agent in the Administration Console. This name must be unique among agents. See "Object Namespace" on page 248 for details.
serverHost	(Required) Name of the host machine running the BAM Server. If they are running on the same machine as the File Agent, specify localhost as the name.
agentImplClass	(Required) Agent implementation class. Do not change this value; currently com.celequest.agent.FileAgent is required.
agentImplConfigFile	(Required) Identifies the configuration file for the implementation (the text file component), usually FileAgent.xml.
agentPort	(Required) Port used to communicate to the agent on the agent's host. Used for communication by the server to the agent for disable and enable status changes. Use any valid port number, such as 5050.

Attribute	Description
pollingInterval	(Required) How frequently (in seconds) to look for new events.
loggingDirectory	(Optional: default is configuration file directory) Directory in which to log file information. The log filename is agentName.log.

Example

The following example names the event agent as “orderStatusEvent”, identifies the implementation configuration file as FileAgent.xml, and sets the server port to 8080:

```
<?xml version="1.0" encoding="UTF-8" ?>
<VCAgent xmlns="http://www.celequest.com/3"
   xmlns:xsi="http://www.w3.org/2001/xmlSchema-instance"
   xsi:schemaLocation="http://www.celequest.com/3 VCAgent.xsd"
   serverPort="80"
>
   <agentName>flatFileEvent</agentName>
   <serverHost>localhost</serverHost>
   <agentImplClass>com.celequest.agent.FileAgent</agentImplClass>
   <agentImplConfigFile>FileAgent.xml</agentImplConfigFile>
   <agentPort>5050</agentPort>
   <pollingInterval>20</pollingInterval>
</VCAgent>
```

FileAgent.xml

This file configures the text file component (the *implementation*) of the File Agent. This file’s actual name and location are identified in the TestAgent.xml file, and it is usually located in the same directory as that file. This configuration file has four configuration elements, though most configurations use the <filename> and <fileDisposal> elements only.

Elements

The FileAgent.xml file has the following configuration elements:

Element	Description
filename	The name and location of the source text file that contains the events. The file is assumed to be in the same directory as the configuration file unless you identify another location in the filename. You may use relative or complete file path specifications. And the filename may include * and ? wildcard characters. On UNIX systems use a slash to separate directory path names, such as events/file*.txt. On Windows systems use two back slashes to separate directory path names, such as events\\file*.txt.
type	Identifies the source as a STREAM or FILE. Use FILE when the entire text file must be uploaded atomically (all or nothing), such as for an XML file. Otherwise, use STREAM to upload lines in batches defined by the buffersize element.
buffersize	(optional: default is 4,000+EOL) Count of characters to buffer or send in batch to the server. The actual size sent is the buffersize plus the remainder of the line of characters that span the limit. Use this setting to avoid uploading excessively large amounts of event text at one time.
fileDisposal	(optional) What to do with the source file after uploading its data. Choices are: delete — (default) Deletes the file after upload. move — Moves the file to a directory specified by the target attribute. rename — Renames the file by adding the extension attribute to the filename. Both move and rename overwrite any existing files of the same name in the target location, without warning or error.

Example

This example identifies the source text file as orderStatusData.txt in the events\ subdirectory on a Windows host and moves the finished file into the ..\done\ sibling directory:

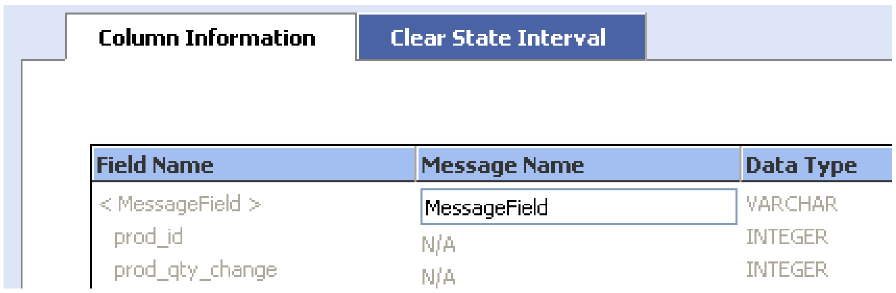
```
<?xml version="1.0" encoding="UTF-8" ?>
<FileAgent xmlns="http://www.celequest.com/3"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:schemaLocation="http://www.celequest.com/3 FileAgent.xsd">
   <fileName>event\\orderStatusData.txt</fileName>
   <fileDisposal>
      <move target="..\\done\\" />
   </fileDisposal>
</FileAgent>
```

Delimited Files

In a delimited file, each field (column) is separated by a character, typically a comma. For example:

```
0703,00001,Assigned,13,2003-03-05 14:23:00,Sridar
0706,00004,Open,13,2003-03-05 19:50:00,
0706,00004,Resolved,13,2003-03-05 19:50:00,Niku
```

Note: Delimited files are also called comma separated value (CSV) files.



Field Name	Message Name	Data Type
< MessageField >	MessageField	VARCHAR
prod_id	N/A	INTEGER
prod_qty_change	N/A	INTEGER

The field separator character, escape character, and text qualifier are each customizable.

- Separator character — Separates each field in the row; usually a comma character (,).
- Escape character — Precedes characters that are not to be used as a separator; usually a back slash character (\). For example, if the separator character is a comma, and the text contains a real comma, then the real comma is “escaped” with a preceding back slash. For example, the comma after “Altadena” is not a field separator:

  ```
  123 Buena Loma Dr,Altadena\, CA,91001
  ```
- Text qualifier — Text strings are further bounded by this character, usually a double quote("). Use this option when text strings are qualified to be different from other data types. For example, this event has text fields that contain numerals, but one of the fields (data value 13) is numeric:

  ```
  "0706","00004","Open",13,2003-03-05 19:50:00,""
  ```

Source rows that do not contain enough data to fill the row generate an error. To permit the data without generating an error, turn on Allow Short Rows. For example, this text generates an error when the third row in imported unless short rows is allowed:

```
1,2,3,4,5,6
1,,,,,
1
```

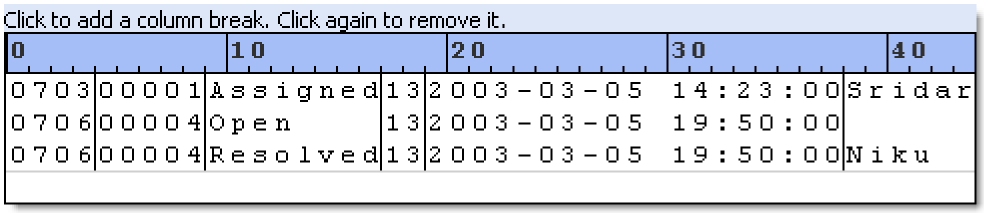
When the First row contains field names option is selected, the names in that row appear as the column names. Otherwise, assign the names manually. Additionally, for each column, assign a data type and optionally declare a data format.

Fixed-Width Files

In fixed-width files, each field (column) is the same predefined width in each file row, similar to a spreadsheet table. For example:

```
070300001Assigned13  2003-03-05 14:23:00Sridar
070600004Open    13  2003-03-05 19:50:00
070600004Resolved13  2003-03-05 19:50:00Niku
```

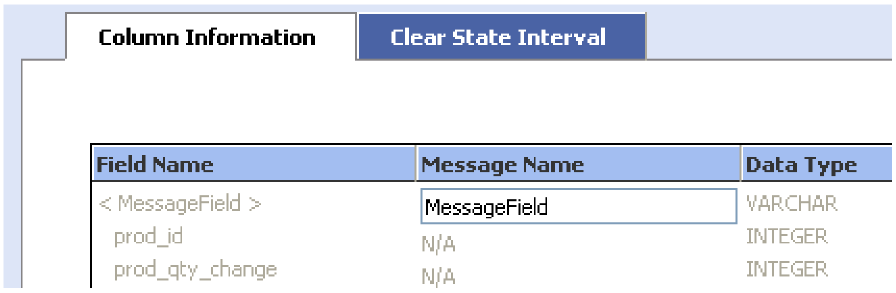
To import a fixed-width file, you need to identify the column positions that begin each field of data with the Set Field Widths dialog. When you provide a sample, the sample data are shown and you click the columns to indicate the start of a field.



Click to add a column break. Click again to remove it.

```
0703|00001|Assigned|13|2003-03-05 14:23:00|Sridar
0706|00004|Open    |13|2003-03-05 19:50:00|
0706|00004|Resolved|13|2003-03-05 19:50:00|Niku
```

If you do not have a sample, you need to identify the starting position of each column in the text. Remember too that the first field starts at position zero (0).



Field Name	Message Name	Data Type
< MessageField >	MessageField	VARCHAR
prod_id	N/A	INTEGER
prod_qty_change	N/A	INTEGER

Once the column positions have been defined, you can assign names and declare their data types and formats.

When the First row contains field names option is selected, the names in that row appear as the column names. Otherwise, assign the names manually. Additionally, for each column, assign a data type and optionally declare a data format.

XML Files

Business Activity Monitoring supports the generation of a data stream from data sources in XML format. However, Business Activity Monitoring views and events are in table format (that is, rows and columns) while XML data is hierarchical. This requires that the flattening of the XML in an event stream, where the hierarchical data is mapped to tabular data. The process for flattening XML is described in the section "How XML Files are Flattened" on page 98.

To define an XML-based event stream in a Business Activity Monitoring application, an application designer must specify the columns that will be output on the event stream. Each column in the stream has a name, a type, and a source. Each column has a source specified as an XPath expression, meaning that the column will be filled by finding elements that match the given XPath, and taking the value of the element in the XML input to use as the value of the column in an output row. Each column XPath should either specify an element or an attribute. Usually, a column XPath that specifies an element matches only atomic elements in the input document; that is, elements that contain only text and no other elements (for details, see "Evaluating Elements" on page 105).

The mapping of the hierarchical XML data to a table is accomplished by having the source for each column in the table specified as an XPath expression. The columns in the table are filled by finding elements that match the given XPath, taking the value of the elements in the XML input and using the value in the column of an output row. Each column XPath specifies an atomic element or an attribute. See the following section, XPaths for more information about how XPaths are used to locate rows and columns.

XPaths

When defining the event's Field Information, XPaths locate the columns and rows in the XML file as follows:

- Schema XPath specifies a compound element; that is, an element that is composed of other elements. All the data for an event row is contained within the elements specified by the schema XPath, and the XPath expressions for the column source are relative to the schema XPath. If a an XML document contains multiple elements that satisfy the schema XPath, a table is generated for each such element and the output of the document is defined as the union of the output of the elements matching the schema XPath. A Schema XPath is an absolute path to the element in the XML structure, and as such always begins with a slash (/) followed by the root element and path to the column element; for example, "/problem_tickets/ticket/ticket_id" is such a schema XPath.
- Relative XPath identifies a column element or attribute relative to the row element. When the column is a child element of the row element, the XPath is either just the element name, or it begins with "child::". For example, these are valid Relative XPaths from the previous example:

  ```
  ticket_id
  child::status
  customer/customer_name
  child::customer/customer_name
  ```

  To locate an attribute, put an at-symbol (@) before the attribute name, like this:

  ```
  customer/@cust_id
  ```

  Note: The XPath standard for locations defines additional XML node mappings not supported by Business Activity Monitoring events.

XML Field Information

Each column in the event table is defined as a field in the XML event editor. Each field has the following attributes:

Attribute	Description
Field Name	Name of the column in the event table.
Relative XPath	Element in the XML file that contains this field's data. See XPaths" above for details.
XML Data Type	Data type of the XML element. See below for details.
Business Activity Monitoring Data Type	Data type of the column in the event table. See "Data Types" on page 55 for details.
Formatting	Formatting of the decimal, string, or date-time value.

XML Data Types

The XML data types map to Business Activity Monitoring Data Types as follows.

XSD	Data Type
anyURI	Varchar
base64Binary	Varchar
Boolean	Boolean
byte	Integer
date	Timestamp (time portion zero'ed out)
dateTime	Timestamp Note the fractional part of a second is supported up to 9 significant digits
decimal	Decimal
double	Double
duration	Varchar (as a string)
ENTITIES	Varchar
ENTITY	Varchar
float	Double
gDay	Varchar (as a string) Defines a part of a date - the day (DD)
gMonth	Varchar (as a string) Defines a part of a date — the month (MM)
gMonthDay	Varchar (as a string) Defines a part of a date — the month and day (MM-DD)

XSD	Data Type
gYear	Varchar (as a string) Defines a part of a date — the year (CCYY)
gYearMonth	Varchar (as a string) Defines a part of a date — the year and month (CCYY-MM)
hexBinary	Varchar
ID	Varchar
IDREF	Varchar
IDREFS	Varchar
int	Integer
integer	Integer
language	Varchar
long	Decimal
Name	Varchar
NCName	Varchar
negativeInteger	Integer
NMTOKEN	Varchar
NMTOKENS	Varchar (as a single string)
nonNegativeInteger	Integer
nonPositiveInteger	Integer
normalizedString	Varchar
NOTATION	Varchar
positiveInteger	Integer
QName	Varchar
QName	Varchar
short	Integer
string	Varchar
time	Varchar
token	Varchar
unsignedByte	Integer
unsignedInt	Decimal

XSD	Data Type
unsignedLong	Decimal
unsignedShort	Integer

How XML Files are Flattened

As stated in the introduction to this section, the mapping of the hierarchical XML data to a table is accomplished by having the source for each column in the table specified as an XPath expression. The columns in the table are filled by finding elements that match the given XPath, taking the value of the elements in the XML input, and using the value in the column of an output row. For example, the following XML document has the XPaths: a/b, a/c, a/d.

Note: The examples in this section always give the absolute XPath for a column. However, when configuring an XML event, you will use an absolute schema XPath and column events will be XPaths that are relative to that schema XPath.

```
<a>1
   <b>0</b>
   <c>1</c>
   <d>2</d>
</a>
```

The resulting table has one row:

0, 1, 2

In this next example, the document is slightly more complicated and has the XPaths: a/b, a/c/d a/c/e/f.

```
<a>
   <b>0</b>
   <c>
       <d>1<d>
        <e>
           <f>2</f>
        </e>
       <d>
   </c>
</a>
```

The resulting table also has one row:

0,1,2

In these examples, both XML documents result in a table with one row because each XPath expression matches exactly one element. The resulting table has one row, where each column is filled by the value of the element that matches XPath that defines the source for a column.

In some cases, more than one element can match an XPath expression for a column source. In the following example, the XPaths are a/b and a/c, which match more than one column.

```
<a>
   <b>0</b>
   <b>1</b>
   <c>2</c>
   <d>3</d>
</a>
```

It looks like the resulting table should be:

```
0,2
0,3
```

However, the resulting table is really:

```
0,2
0,3
1,2
1,3
```

This is because all four elements are children of a single parent element <a> and the source for the columns in each row is specified by the XPaths a/b and a/c. This means that 0,2 is no more correct than 0,3 and both rows should be in the table. In this case, the resulting table is created through a cross product of the of the elements.

To ensure that when XML is flattened the correct result is produced for all cases, Business Activity Monitoring does the following:

- Strip the input XML of all extraneous elements so that values are not duplicated where such duplication is not necessary, which would result in duplicate rows in the output.
- Relationize the document that has been stripped of extraneous elements.

To accomplish the last task, the document is distributed into a number of intermediated of tables, where each table has a primary key (pk), parent foreign key (parent_fk), and a value. The final result table is created through a key join of all these intermediate tables. For example, the following XML document can be relationalized into eight intermediate tables a, a/b, a/b/c, a/b/d, a/b/b/d/e, a/b/d/f, a/b/d/f/g, and a/h.

```
<a>
   <b>
      <c>0</c>
      <d>
         <e>1</e>
         <e>2</e>
            <f>
                <g>3</g>
            </f>
      </d>
    </b>
    <b>
       <c>4</c>
       <d>
          <e>5</e>
          <f>
              <g>6</g>
              <g>7</g>
          <f/>
        </d>
   </b>
   <h>8</h>
   <h>9</h>
</a>
```

Next primary keys (pk) are assigned to each element in a depth-first traversal of the document.

```
<a pk='0'>
   <b pk='1'>
      <c pk='2'>0</c>
      <d pk='3'>
         <e pk='4'>1</e>
         <e pk='5'>2</e>
            <f pk='6'>
                <g pk='7'>3</g>
            </f>
      </d>
    </b>
    <b pk='8'>
       <c pk='9'> 4 </c>
       <d pk='10'>
          <e pk='11'> 5 </e>
          <f pk='12'>
              <g pk='13'>6</g>
              <g pk='14'>7</g>
          <f/>
        </d>
   </b>
   <h pk='15'>8</h>
   <h pk='16'>9</h>
</a>
```

Now the tables are key joined using the primary keys assigned to each element. The SELECT statement that defines the final output table for the event is:

```
SELECT a/b/c.value, a/b/d/e.value, a/b/d/f/g.value, a/h.value
FROM ((((((a/
     FULL OUTER JOIN /a/b on /a/b.parent_fk = /a.pk)
     FULL OUTER JOIN /a/b/c on /a/b/c.parent_fk = /a/b.pk)
     FULL OUTER JOIN /a/b/d on /a/b/d.parent_fk = /a/b.pk)
     FULL OUTER JOIN /a/b/d/e on /a/b/d/e.parent_fk = /a/b/d.pk)
     FULL OUTER JOIN /a/b/d/f on /a/b/d/f.parent_fk = /a/b/d.pk)
     FULL OUTER JOIN /a/b/d/f/g on /a/b/d/f/g.parent_fk = /a/b/d/f.pk)
     FULL OUTER JOIN /a/h on /a/h.parent_fk = a.pk
```

The same join can also be accomplished by using the primary key of each table.

```
SELECT /a.pk, /a/b.pk, /a/b/c.pk, /a/b/d.pk, /a/b/d/e.pk, /a/b/d/f.pk,
/a/b/d/f/g.pk, /a/h.pk
FROM ((((((/a
     FULL OUTER JOIN /a/b on /a/b.parent_fk = /a.pk)
     FULL OUTER JOIN /a/b/c/ on /a/b/c.parent_fk = /a/b.pk)
     FULL OUTER JOIN /a/b/d on /a/b/d.parent_fk = /a/b.pk)
     FULL OUTER JOIN /a/b/d/e on /a/b/d/e.parent_fk = /a/b/d.pk)
     FULL OUTER JOIN /a/b/d/f on /a/b/d/f.parent_fk = /a/b/d.pk)
     FULL OUTER JOIN /a/b/d/f/g on /a/b/d/f/g.parent_fk = /a/b/d/f.pk)
     FULL OUTER JOIN /a/h on /a/h.parent_fk = a.pk
```

The resulting table is:

```
0,1,3,8
0,2,3,8
4,5,6,8
4,5,7,8
0,1,3,9
0,2,3,9
4,5,6,9
4,5,7,9
```

Handling Missing Elements

In cases where elements do not have the same child elements, a full outer join is used to generate the table because defining the output with an inner join would not produce the desired results. For example, the following document does not have a <d >child element in the first <b> element, and the second <b> element has a <d> element but no <c> element.

```
<a>
   <b>
      <c>0</c>
   </b>
   <b>
      <d>1</d>
   </b>
</a>
```

The annotated document is

```
<a pk='0'>
   <b pk='1'>
      <c pk='2'>0</c>
   </b>
   <b>
      <d pk='3'>1</d>
   </b>
</a>
```

The select statement is:

```
SELECT /a/b/c.value, /a/b/d.value
FROM ((/a
     FULL OUTER JOIN /a/b on /a/b.parent_fk = /a.pk)
     FULL OUTER JOIN /a/b/c on /a/b/c.parent_fk = /a/b.pk)
     FULL OUTER JOIN /a/b/d on /a/b/d.parent_fk = /a/b.pk
```

The SELECT statement with the primary key projection is:

```
SELECT /a.pk, /a/b.pk, /a/b/c.pk, /a/b/d.pk
FROM ((/a
     FULL OUTER JOIN /a/b on /a/b.parent_fk = /a.pk)
     FULL OUTER JOIN /a/b/c on /a/b/c.parent_fk = /a/b.pk)
     FULL OUTER JOIN /a/b/d on /a/b/d.parent_fk = /a/b.pk
```

The final result is:

```
  0,null
  null,1
```

Handling Attributes

When elements in a document contain attributes, the attributes are represented as columns in the intermediate table. For example, in the following document, the column XPaths are: a/@b, a/c/@d, and a/c.

```
<a b=‘0’>
   <c d=‘1’>2</c>
   <c d=‘3’>4</c>
</a>
```

The primary keys are assigned to the elements as follows:

```
<a pk=‘0’ b=‘0’>
   <c pk=‘1’ d=‘1’> 2</c>
   <c pk=‘2’ d=‘3’> 4 </c>
</a>
```

The select statement is:

```
SELECT /a/@b, /a/c.d, /a/c.value
FROM /a FULL OUTER JOIN /a/c on /a/c.parent_fk = /a.pk
```

The select statement with primary keys is:

```
SELECT /a/@b, /a/c.d, /a/c.pk
FROM /a FULL OUTER JOIN /a/c on /a/c.parent_fk = /a.pk
```

The resulting table is:

```
    0,1,2
    0,3,4
```

When an XML element has two children with the same tag name, a simple unpredicated XPath expression returns a set of nodes. For example, in the following document the XPath expression /a/b returns two rows:

```
<a>
   <b>0</b>
   <b>1</b>
</a>
```

The rows returned are:

```
0
1
```

In XPath expressions, you can use index predicates to find a node or a specific value in a node. For example, the expression /a/b[1] returns 0, and the expression /a/b[2] returns 1. That is, each XPath returns one row. The expression /a/b[3] returns nothing because there is no third child in the element <a>. In the following example, the XPath /a/b/c[1] returns two rows.

```
<a>
   <b>
      <c>0</c>
      <c>1</c>
   </b>
   <b>
      <c>2</c>
      <c>3</c>
   </b>
</a>
```

The rows returned are:

```
0
2
```

This is because the XPath /a/b/c[1] specifies the first child in the element <b>, and there are two <b> child elements in the parent element <a>.

Ignored Elements

Elements in a document that do not match any column XPath or doe not have any children that match a column XPath are ignored. For example, given the following document and the XPath /a/b/c, only a single row will be returned.

```
<a>
   <b>
      <c>0</c>
      <d>
         <e>1</e>
         <e>2</e>
      </d>
   </b>
<a>
```

Only one row is returned because the /a/b/d element does not match the column XPath /a/b/c, nor does it have any children that match. Therefore, after the first pass the document becomes:

```
<a>
   <b>
      <c>0</c>
   </b>
<a>
```

In the following document also produces one row.

```
<a>
   <b>
      <c>0</c>
   </b>
   <b>
      <d>1</d>
   </b>
</a>
```

The above document produces only one row because the second /ab element matches the column XPath; however it does not have any children that match. Therefore, this document also becomes the following after the first pass:

```
<a>
   <b>
      <c>0</c>
   </b>
<a>
```

The same process also occurs with attributes. Given the XPath /a/b/@id, the following document, produces only one row after the first pass because the second <b> element does not have an id attribute.

```
<a>
   <b id='0' name='oracle'/>
   <b name='ibm'/>
</a>
```

Thus after the first pass, the document becomes

```
<a>
   <b id='0' name='oracle'/>
</a>
```

Index Predicates

Index predicates make it possible to overcome a particular problem where one row needs to be returned but the XML contains two children tags with the same name that contain different values. For example, the following document contains two occurrences of the tag <keyword>:

```
<sales>
   <sale>
      <id>0</id>
      <amount>$125</amount>
      <keyword>Salesforce </keyword>
      <keyword>SAP</keyword>
   </sale>
</sales>
```

For the schema XPath /sales, the column XPaths are:

```
sale/id
sale/amount
sale/keyword
```

This returns two rows:

```
0, $125, Salesforce
0, $124, SAP
```

However, there was only one sale, so only one row should have been returned: 0, $125, Salesforce. That is, you wanted to create a view on top of the event SELECT SUM(amount) FROM sales, and you only needed one keyword. In this case, you could provide the following XPaths:

```
sale/id
sale/amount
sale/keyword[1]
```

This would return the row: 0, $124, Salesforce. However, if you need both keywords, you would need to use a different method than using index predicates.

Index predicates can occur at any level within an XPath expression, not just the last level. For example, the expression /a/b[2]/c returns two rows given the following document:

```
<a>
   <b>
      <c>0</c>
      <c>1</c>
   </b>
   <b>
      <c>2</c>
      <c>3</c>
   </b>
</a>
```

The two rows returned are:

```
2
3
```

Evaluating Elements

In certain cases elements will return a value of null when evaluated by Business Activity Monitoring:

- The element is empty. For example, there is one /a/b element in the following document, which Business Activity Monitoring defines as null:

  ```
  <a>
     <b></b>
  </a>
  ```

- The element has children but no text. For example, the following document contains an element <c>, which has a child element <d> but does not contains any text.

  ```
  <a>
     <b>0</b>
        <c>
           <d>1</d>
        </c>
        <c>2</c>
  </a>
  ```

  The output from this document given the XPaths /a/b, a/b/c, and /a/b/c/d would be:

  ```
  0, null, 1
  0, 2, null
  ```

If an element is not an atomic element and contains a child element, it will return the text from the element as well as the child. For example, the following document contains the element <b>, which is not atomic, and a child element <c>:

```
<a>
   <b>hello
      <c>world</c>
   com</b>
</a>
```

The column XPaths /a/b and /a/b/c would return the following rows:

```
hello
   , com, world
```

XML Flattening Issues

You should be aware of the following when XML is flattened:

- It is illegal to mix predication. For example, the set of XPaths /a/b[2], /a/b are illegal. This is because the semantics that Business Activity Monitoring uses to flatten XML state that as soon as index predicates are used with a given XPath prefix, all elements matching that prefix are implicitly rewritten by appending the index. Each element must be either written with the index predicate or without it. By writing a given XPath prefix in one column with an index predicate and in another column without an index predicate, all elements corresponding to that prefix become ambiguous.
- In a flat file event, every match to the schema XPath defines a new event, which may define any number of rows. For example, a flat file event stream with the schema XPath /a/b and the column XPath c will generate two events from the following document, both with one row.

  ```
  <a>
     <b>
        <c>0</c>
     </b>
     <b>
        <c>1</>
     </b>
  </a>
  ```

  However, for an XML document coming through JMS, one document always generates one event. (This rarely makes a difference.)
- Whitespace is illegal in an XPath expression. The server will not allow you to save an event with whitespace in the XPaths. The following xpaths are illegal:

- b/ c
- b/c [1]
- b/c[1]

- If you specify more than one column with the same XPath, you will always get the same value in both columns, and it will not cause the flattener to generate more rows. For example, The following document returns the result 0,1,1 given the schema /a and the columns b, c, c.

```
<a>
   <b>0</b>
   <c>1</b>
<a>
```

The semantic query is:

```
SELECT /a/b.value, /a/c.value, /a/c.value
FROM
   (/a FULL OUTER JOIN /a/b ON /a/b.parent_fk = /a.pk)
   FULL OUTER JOIN /a/c ON /a/c.parent_fk = /a.pk
```

12 Formulas

All formulas in Business Activity Monitoring are expressions in the C-SQL language, a derivative of ANSI SQL. Some of the formulas are simple expressions, such as field expressions that define the values in business view columns. Other expressions are more complex and represent entire components of the C-SQL query statement (SELECT), such as the WHERE, WINDOW, and JOIN clauses.

All formulas in Business Activity Monitoring accept Operators and Constants that can manipulate the values, and they can accept most C-SQL Functions to further process results.

Note: For detailed descriptions of each of the C-SQL SELECT statement, operators, and functions, see "SELECT" on page 321.

➤ **In this Chapter:**

Functions

Functions return values that are system information, such as the current time, manipulations of data, such as converting a string of characters to upper case, or are evaluations of sets of data, such as the total of all prices in a set of purchase orders.

C-SQL functions can be used in most formulas in Business Activity Monitoring. However, some are limited by the operations allowed in the formula's context. The next section "Function Types", describes the types of functions and tells where they are allowed.

If you are looking for a function for a specific task, see "Function Categories" on page 111 to see what tasks the functions can perform.

For a detailed description of each function, see "Functions" on page 114.

Function Types

C-SQL has five types of functions: Scalar, Set, Rank, Moving Set, and Tumbling Set. The type distinctions determine where you may include the function in a formula.

Scalar

Scalar functions operates on a single item and provide a single result. For example, the ABS() function returns the absolute value of a (single) number. Scalar functions may appear in any C-SQL expression. The scalar functions are:

ABS	DISPLAY_MONEY	LPAD	SIGN
CAST	EXP	LTRIM	SQRT
CEIL	FLOOR	MOD	SUBSTRING
CHARACTER_LENGTH	GREATEST	POSITION	TIMESTAMP_DIFF
CONCAT	IS_RAISED	POWER	TO_CHAR
CURRENT_TIMESTAMP	LAST_DAY	PRIOR_VALUE	TO_DATE
CURRENT_USER	LEAST	ROUND	TRUNC
DATE_ADD	LOG	RPAD	UPPER
DATE_DIFF	LOWER	RTRIM	

Set

Set functions perform aggregations on sets of business view rows and produce a single results for the set. For example, SUM() provides the total of all the rows in a column in a view. A set function *may only be used* in the Select List of a SELECT statement: the field definitions of a view.

Note: A set function may reference another set function, but the results are the same as if only the referenced (inner) function was expressed alone. For example SUM(AVG(Order_Total)) has the same result as AVG(Order_Total).

AVG	MAX	PREV	STD_DEVIATION
COUNT	MIN	SUM	VARIANCE
CURRENT			

Note: NULL is ignored when computing set function, moving set function, and rank function values. For example, the average of (3, NULL, 3) is 3, not NULL, and it is not 2.

Rank

Rank functions compute the scalar result for a column in each row in a set, *with respect to the entire set*. A rank function *may only be used* in the Select List of a SELECT statement.

NTILE	RANK	RATIO_TO_REPORT

Moving Set

Moving set functions are special case set functions that performs calculations on a set of the *latest rows* in a view. The set of rows to include is determined only when a new event arrives. At that time, only the latest rows that meet the set criteria are included in the calculation. Moving set functions are defined by applying "MOV_" to an existing set function. For example, to calculate a moving average, use MOV_AVG(). A moving set can be determined by a count of events or as a duration of time. The following example calculates the mean average of Order_Total for the last twelve hours. As new orders are inserted into the view they are included in the calculation; however, orders older than 12 hours are excluded.

```
SELECT MOV_AVG(Order_Total, HOUR, 12) FROM Purchase_Orders
```

Note: Moving set functions are a shorthand way to express a simple query window. See "MOV_function" on page 150 for a complete description.

MOV_AVG	MOV_MAX	MOV_SUM	MOV_VARIANCE
MOV_COUNT	MOV_MIN	MOV_STD_DEVIATION	

Tumbling Set

Tumbling set functions are special case set functions that perform calculations on a windowed set of the rows in a view. The set of rows to include is determined when a new event arrives, and the set empties when full. Tumbling set functions are a shorthand way to express a tumbling window query. For more information, see “Tumbling Windows” on page 286.

TUMBLE_AVG	TUMBLE_MIN	TUMBLE_STD_DEVIATION
TUMBLE_COUNT	TUMBLE_SUM	TUMBLE_VARIANCE
TUMBLE_MAX		

Function Categories

These are the categories of C-SQL functions:

Alerts

IS_RAISED

Conversion

CAST
DISPLAY_MONEY
TO_CHAR
TO_DATE

Date and time

CURRENT_TIMESTAMP
DATE_ADD
DATE_DIFF
GREATEST
TIMESTAMP_DIFF
LEAST
LAST_DAY
TO_CHAR
TO_DATE

Math

ABS
CAST
CEIL
EXP
FLOOR
LOG
MOD
POWER
ROUND
SIGN
SQRT
SUM
TRUNC

Ranking

NTILE
RANK
RATIO_TO_REPORT

Rules

IS_RAISED
CURRENT_USER

Statistical

AVG
COUNT
GREATEST
LEAST
MAX
MIN
MOV_AVG
MOV_COUNT
MOV_MAX
MOV_MIN
MOV_SUM
MOV_STD_DEVIATION
MOV_VARIANCE
NTILE
RANK
RATIO_TO_REPORT
STD_DEVIATION
TUMBLE_AVG
TUMBLE_COUNT
TUMBLE_MAX
TUMBLE_MIN
TUMBLE_SUM
TUMBLE_STD_DEVIATION
TUMBLE_VARIANCE
VARIANCE

Text and String

CAST
CHARACTER_LENGTH
CONCAT
DISPLAY_MONEY
GREATEST
LEAST
LOWER
LTRIM
LPAD
POSITION
RPAD
RTRIM

SUBSTRING
TO_CHAR
TO_DATE
UPPER

Time-Series and Aggregation

AVG
COUNT
MAX
MIN
MOV_function
MOV_AVG
MOV_COUNT
MOV_MAX
MOV_MIN
MOV_SUM
MOV_STD_DEVIATION
MOV_VARIANCE
STD_DEVIATION
TUMBLE_AVG
TUMBLE_COUNT
TUMBLE_MAX
TUMBLE_MIN
TUMBLE_SUM
TUMBLE_STD_DEVIATION
TUMBLE_VARIANCE
VARIANCE

Views

CURRENT
IS_RAISED
PREV
PRIOR_VALUE

13 Functions

C-SQL functions may appear in commands and rule formulas where an expression is accepted. For a general discussion of functions, a list of the Function Categories, see "Formulas" on page 108. This document describes each of the following C-SQL functions in detail.

C-SQL Function	Description
ABS	Returns the absolute value of a number.
AVG	Returns the average value (arithmetic mean) of a set of numeric values.
CASE	The CASE expression is described in detail in "CASE Expression" on page 323.
CAST	Converts a value from one Business Activity Monitoring type to another Business Activity Monitoring type.
CEIL	Returns the smallest integer, rounded up from zero, greater than or equal to a number.
CHARACTER_LENGTH	Returns the length of a string.
CONCAT	Returns a string that is the concatenation of two characters or strings.
concatList	Returns a string that is the concatenation of a list of characters or strings.
concatSet	Returns an alphabetically ordered set of strings.
COUNT	Returns the count of rows in a view or set.
CURRENT	Returns a value from the latest or last row in a set.
CURRENT_TIMESTAMP	Returns the current date and time in the server time zone.
CURRENT_USER	Returns the login name of the current user.
DATE_ADD	Adds a duration of time to a date-time value.
DATE_DIFF	Subtracts a duration from a date-time value.
DISPLAY_MONEY	Formats a number as a currency value.
EXP	Returns e raised to a specific power.
FLOOR	Returns largest integer less than or equal to an expression.

C-SQL Function	Description
gammaDist	Returns the gamma distribution of a value.
GREATEST	Returns the greatest of a list of expression results.
IS_RAISED	Returns true when the specified alert is in a raised state.
LAST_DAY	Returns the date of the last day of the month that contains a specified date.
LEAST	Returns the least value of a list of expressions.
LOG	Returns the logarithm of a number from a specific base.
logNormDist	Returns the cumulative lognormal distribution of a value.
LOWER	Converts all uppercase characters in a string to lower case.
LPAD	Inserts one or more instances of a string into the start of another string.
LTRIM	Removes characters from the start of a string.
MAX	Returns the maximum value from a set.
median	Returns the median (middle) number in a set.
MIN	Returns the minimum value from a set.
MOD	Returns the modulus (remainder) of a division.
mode	Returns the most frequently occurring number in a set.
MOV_function	Limits the rows used in a set function calculation to a set of the latest rows in the view.
MOV_AVG	Returns the moving average value (arithmetic mean) of a moving window set of numeric values.
MOV_COUNT	Returns the count of rows in a moving window set.
MOV_MAX	Returns the maximum value from a moving window set.
MOV_MIN	Returns the minimum value from a moving window set.
MOV_SUM	Returns the sum of a moving window set of numeric values.
MOV_STD_DEVIATION	Returns sample standard deviation of a moving window set of numbers.
MOV_VARIANCE	Returns the square of the sample standard deviation of a moving window set of numbers.

C-SQL Function	Description
NTILE	Determines the tier rank of each value in a set with respect to the entire set.
POSITION	Returns the position of a character or string within a string.
POWER	Returns a value raised to a specific power.
PREV	Returns a value from the next to last row in a set.
PRIOR_VALUE	Returns the prior value of a column, alias, or expression.
RANK	Determines the rank of each value in a set with respect to the entire set.
RATIO_TO_REPORT	Calculates the ratio of a value to the sum of the values for the entire set.
ROUND	Returns a number rounded up to a specified count of decimal places.
RPAD	Adds one or more instances of a string to the end of another string.
RTRIM	Removes characters from the end of a string.
SAFE_DIVIDE	Returns a quotient of two values, unless the quotient is 0, in which case it returns an alternate quotient value.
SIGN	Identifies the arithmetic sign of a number.
SQRT	Returns the square root of a number.
SUBSTRING	Returns the portion of a string identified by position and length.
SUM	Returns the sum of a set of numeric values.
SUM_OVER_GROUPS	Returns a running sum of the numeric values ordered by the column specified in the arguments.
STD_DEVIATION	Returns sample standard deviation of a set of numbers.
TIMESTAMP_DIFF	Returns the interval of time between two timestamps.
TO_CHAR	Converts a date-time to a character string.
TO_DATE	Converts a character string to a date-time value.
TRUNC	Truncates a number to a specific count of decimal places.
TUMBLE_AVG	Returns the average value (arithmetic mean) of a tumbling window set.

C-SQL Function	**Description**
TUMBLE_COUNT	Returns the count of rows in a tumbling window set.
TUMBLE_MAX	Returns the maximum value from a tumbling window set.
TUMBLE_MIN	Returns the minimum value from a tumbling window set.
TUMBLE_SUM	Returns the sum of a tumbling window set of numeric values.
TUMBLE_STD_DEVIATION	Returns sample standard deviation of a tumbling window set of numbers.
TUMBLE_VARIANCE	Returns the square of the sample standard deviation of a tumbling window set of numbers.
UPPER	Converts all lowercase characters in a string to uppercase.
VARIANCE	Returns the square of the sample standard deviation of a set of numbers.
yield	

ABS

This scalar function returns the absolute value of a number.

Syntax

```
ABS( numeric )
```

Parameters

- numeric— An expression that evaluates to a numeric.

Return Type

Numeric, same data-type as numeric argument.

Example

Return the difference in two persons ages, regardless of which is older.

```
SELECT ABS( father_age - mother_age ) AS "Difference of parents ages"
    FROM Family
```

SIGN() returns the arithmetic sign of a number.

AVG

This set function returns the average value (arithmetic mean) of a set of numeric values.

Syntax

```
AVG( numeric )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view.

Return Type

Numeric, same data-type as numeric argument.

Remarks

Calculates the average of numeric in all rows in the referenced view. When using a GROUP BY Clause, the average applies to the numeric in each group.

```
SELECT AVG( pr_price ) "Average price" FROM Products

Average price
-------------
49.96
```

Example

The following example uses moving averages to produce results *similar* to a Moving Average Convergence/Divergence (MACD) indicator. (This is not a true MACD because it does not use an exponential moving average.) In securities trading, the basic MACD trading rule is to sell when the MACD falls below its 9 day average and to buy when the MACD rises above the 9 day average. You can accomplish this by defining rules similar to this:

- Raise SELL when MACD > Nine_Day_MA
  Lower SELL when MACD < Nine_Day_MA
- Raise BUY when MACD < Nine_Day_MA
  Lower BUY when MACD > Nine_Day_MA

To get these values you need two views:

- MACD_Base_View tracks the moving averages for each security symbol in the event stream. Note that the Nine_Day_MA formula repeats the formulas for the other two averages. This is because you cannot reference an alias in another column of the same view.

```
SELECT
    StockQuotes.SYMBOL AS Symbol,
    MOV_AVG(StockQuotes_Event.CLOSE, Day, 26, StockQuotes.DATE)
        AS Twentysix_Day_MA,
    MOV_AVG(StockQuotes_Event.CLOSE, Day, 12, StockQuotes.DATE)
        AS Twelve_Day_MA,
    MOV_AVG((MOV_AVG(StockQuotes.CLOSE, Day, 12, StockQuotes.DATE) -
             MOV_AVG(StockQuotes.CLOSE, Day, 26, StockQuotes.DATE)),
            Day, 9, StockQuotes.DATE ) AS Nine_Day_MA
    FROM StockQuotes
    GROUP BY StockQuotes.SYMBOL
```

- MACD_View contains the last MACD values for each security stored in the base view:

```
SELECT MACD_Base_View.Symbol AS Symbol,
       MACD_Base_View.Nine_Day_MA AS Nine_Day_MA,
       (MACD_Base_View.Twentysix_Day_MA -
        MACD_Base_View.Twelve_Day_MA
       ) AS MACD
    FROM MACD_Base_View
```

median() returns the median (middle) number in a set.

mode() returns the most frequently occurring number in a set.

MOV_AVG() returns the moving average for a set.

TUMBLE_AVG() returns the tumbling average for a set.

CASE

The CASE expression is described in detail in "CASE Expression" on page 323.

CAST

This scalar function converts a value from one Business Activity Monitoring type to another Business Activity Monitoring type.

Syntax

```
CAST( value AS vcDataType )
```

Parameters

- value— Value to convert.
- vcDataType— One of the C-SQL Data Types to convert to.
  - INTEGER
  - DECIMAL
  - DOUBLE PRECISION
  - VARCHAR
  - TIMESTAMP
  - BOOLEAN

Return Type

Same as vcDataType argument.

Remarks

Types are cast according the Order of Precedence table in "Data Type Conversion" on page 56. CAST() returns an error if a type cannot be cast as specified in an expression. For example, the following is an error because C-SQL attempts to cast '4.5' to an INTEGER, but the decimal is an illegal character for INTEGER types:

```
3 < CAST( '4.5' AS INTEGER )
```

When casting from a decimal formatted column to a string, the result is zero-padded on the decimals to match the scale, just as when casting from a string to a decimal. For example,

```
CAST( '1.1' AS DECIMAL(5,4) )  --> 1.1000
CAST( CAST( '1.1' AS DECIMAL(5,4) ) AS VARCHAR )  --> '1.1000'
```

Example

Cast a date string into a time-stamp:

```
SELECT CAST('1997-10-22' AS TIMESTAMP )
    FROM Foo;
```

"Data Type Conversion" on page 56 provides details about converting types.

TO_CHAR() converts the timestamp to a character string of specified format.

TO_DATE() converts a character string to a date.

CEIL

This scalar function returns the smallest integer, rounded up from zero, greater than or equal to a number.

Syntax

```
CEIL( numeric )
```

Parameters

- numeric— Number to round.

Return Type

Same data type are numeric result.

Example

`CEIL(1234.56)` returns 1235.00.

`CEIL(-2.75)` returns -2.00.

FLOOR() returns the largest value less than or equal to a number.

CHARACTER_LENGTH

This scalar function returns the length of a string.

Syntax

```
CHARACTER_LENGTH( string )
```

Parameters

- string— String or VARCHAR expression result whose length to evaluate.

Return Type

INTEGER.

Remarks

Alternate spelling is:

```
CHAR_LENGTH( string )
```

Returns an integer that is the length of the string. Returns NULL if the string is NULL.

The length of a string is determined by its displayable characters, and not necessarily the storage length of the string. For example, a Unicode character requires 16-bits of storage — which might be considered as 2 characters of storage on some systems — but the actual character length is 1.

CONCAT

This scalar function returns a string that is the concatenation of two characters or strings.

Syntax

```
CONCAT( string1, string2 )
```

Parameters

- string— A character string value or VARCHAR expression result.

Return Type

VARCHAR.

Remarks

Returns string2 appended to the end of string1. Returns NULL if either string is NULL.

The || operator ("String operators" on page 250) is identical to this function.

Examples

`CONCAT('a', 'b')` returns 'ab'.

`'a'||'b'` returns 'ab'.

concatList() returns a string that is the concatenation of a list of characters or strings.

concatSet() returns an alphabetically ordered set of strings.

"String operators" on page 250 describes the || operator.

concatList

This scalar function returns a string that is the concatenation of a list of characters or strings.

> **Note:** This function is provided as a sample UDF. To use it, you will need to load the function from the directory \samples\udf\jar\com\\manifest on the distribution CD. For information on how to load UDFs, see "User-Defined Functions" on page 346.

Syntax

```
concatList( string1, string2 [, … stringN ] )
```

Parameters

- string— An expression that evaluates to a VARCHAR

Return Type

VARCHAR.

Remarks

Returns string2 appended to the end of string1, string3 appended to string2, and so on.

Ignores NULL values unless all values are NULL, in which case returns an empty string.

Examples

`concatList('a','b','c')` returns 'abc'.

CONCAT() returns a string that is the concatenation of two characters or strings.

concatSet() returns an alphabetically ordered set of strings.

"String operators" on page 250 describes the || operator.

concatSet

This set function returns an alphabetically ordered set of strings.

Note: This function is provided as a sample UDF. To use it, you will need to load the function from the directory \samples\udf\jar\com\celequest\manifest on the distribution CD. For information on how to load UDFs, see "User-Defined Functions" on page 346.

Syntax

```
concatSet( stringExp )
```

Parameters

- stringExp— An expression that evaluates to a VARCHAR. Typically the argument is a column in a view.

Return Type

VARCHAR.

Remarks

Returns a string that is the ordered set of all the strings passed into the function.

Ignores NULL values unless all values are NULL, in which case returns an empty string.

Examples

Consider this statement:

```
SELECT concatSet(item) AS item_list FROM GroceryList
```

If the items in GroceryList are presented as follows in this order:

```
'banana'
'egg'
'apple'
'donut'
NULL
'carrot'
```

The order in item_list in the new view is:

```
'apple,banana,carrot,donut,egg'
```

Subsequently, if 'bagel' is added to GroceryList, the new order in the new view is:

```
'apple,bagel,banana,carrot,donut,egg'
```

CONCAT() returns a string that is the concatenation of two characters or strings.

concatList() returns a string that is the concatenation of a list of characters or strings.

"String operators" on page 250 describes the || operator.

COUNT

This set function returns the count of rows in a view or set.

Syntax

```
COUNT( * )
```

Return Type

INTEGER.

Remarks

Returns zero (0) if the view or set is empty.

This is also known as the "count star" function.

Rows that include NULLs are counted.

MOV_COUNT() returns the count of a moving set.

TUMBLE_COUNT() returns the count of a tumbling set.

CURRENT

This set function returns a value from the latest or last row in a set.

Syntax

```
CURRENT( columnName )
```

Parameters

- columnName— Column or alias to retrieve.

Return Type

Same data-type as argument.

Remarks

Returns a value from the latest row in the set based on the event timestamp. When all rows in the set have the same timestamp, returns the value from the last row in the set.

Example

Gather all stock feed bids and group them by stock symbol. The “current” row will always be the last one received, and as such, will contain the current bid price:

```
SELECT symbol, CURRENT(bid) AS Bid, MAX(bid) AS High, MIN(bid) AS LOW
    FROM Stock_feed
    GROUP BY symbol

symbol  Bid    High   Low
------- ------ ------ -----
K       31.25  31.28  30.72
IBM     80.79  80.04  82.55
VCLR    22.60  24.42  22.00
```

Moving set semantics

Cannot be used with a moving or tumbling set.

PREV() returns a value from the row previous to the current one.

CURRENT_TIMESTAMP

This scalar function returns the current date and time in the server time zone.

Syntax

```
CURRENT_TIMESTAMP()
```

Return Type

Date-Time.

Example

`LAST_DAY( CURRENT_TIMESTAMP() )` returns the date of the last day of the current month.

TO_CHAR() converts a timestamp to a character string.

LAST_DAY() returns the date of the last day of a month.

DATE_ADD() adds a duration to a date-time.

DATE_DIFF() subtracts a duration from a date-time.

TIMESTAMP_DIFF() returns a time interval between two timestamps.

CURRENT_USER

This scalar function returns the login name of the current user.

Syntax

```
CURRENT_USER()
```

Return Type

VARCHAR.

Remarks

Returns the user's login name as defined in Business Activity Monitoring, in the same character case, and as it appears in the BAM Workbench. As such, when using in a comparison, be sure to match the character case exactly.

This function is primarily for use in access filters. See "Access Filters" on page 15, especially the section "Users as Context" on page 16, for examples and uses.

DATE_ADD

This scalar function adds a duration of time to a date-time value.

Syntax

```
DATE_ADD( timestamp, [ durationType, ] duration )
```

Parameters

- timestamp— The date-time to adjust.
- durationType— Type of the duration value; one of these literals:
  - SECOND
  - MINUTE
  - HOUR
  - DAY (default)
  - MONTH
  - YEAR
- duration— Duration of time to add; a literal positive integer.

Return Type

Date-Time.

Remarks

Uses Gregorian calendar addition rules.

Example

`DATE_ADD( CURRENT_TIMESTAMP(), 2)` returns a date-time two days in the future from now.

`DATE_ADD( aTimestamp, DAY, 14 )` returns a value 2 weeks after the data.

DATE_DIFF() subtracts a duration from a date-time.

CURRENT_TIMESTAMP() returns the current date and time.

TIMESTAMP_DIFF() returns a time interval between two timestamps.

DATE_DIFF

This scalar function subtracts a duration from a date-time value.

Syntax

```
DATE_DIFF( timestamp, [ durationType, ] duration )
```

Parameters

- timestamp— The date-time from which to subtract some duration of time.
- durationType— Type of the duration value; one of these literals:
  - SECOND
  - MINUTE
  - HOUR
  - DAY (default)
  - MONTH
  - YEAR
- duration— Duration of time to subtract; a literal positive integer.

Return Type

Date-Time.

Remarks

Uses Gregorian calendar subtraction rules.

Durations that span leap year days and seconds generally ignore the leap value. For example, subtracting 1 year from 3 March 1976 results in 3 March 1975 without being affected by the 29 February 1976 leap day. However, subtracting 1 year from 29 February results in a 28 February date.

Examples

`DATE_DIFF( CURRENT_TIMESTAMP(), 2)` returns a date-time two days ago from now.

DATE_ADD() adds a duration to a date-time.

CURRENT_TIMESTAMP() returns the current date and time.

TIMESTAMP_DIFF() returns a time interval between two timestamps.

DISPLAY_MONEY

This scalar function formats a number as a currency value.

Syntax

```
DISPLAY_MONEY( number [, languageCode, countryCode ] )
```

Parameters

- number— Number to format.
- languageCode— A two-letter ISO 639 language code. Helps determine the currency symbol to display.
- countryCode— A two-letter ISO 3166 country code. Specifies the thousands separator, decimal separator, and count of decimal digits to display based on what is appropriate for the country.

Return Type

VARCHAR.

Remarks

Returns a the number formatted as a currency string.

Omitting the languageCode and countryCode uses the symbol and format appropriate for country that your computer is configured to use by default.

Some currency symbols require that the browser be configured to the correct code-page for the language.

Examples

Here are some examples that format the number 12345.678:

Language/Country	Formula	Result
English/USA	`DISPLAY_MONEY(12345.678,'en','us')`	$12,345.68
Swedish/Sweden	`DISPLAY_MONEY(12345.678,'sv','se')`	12 345,68 kr
German/Germany	`DISPLAY_MONEY(12345.678,'de','de')`	12.345,68 dm

Common Codes

The following table lists some of the common ISO 639 two-letter language codes:

Language	Code	Language	Code	Language	Code
Afrikaans	af	Fiji	fj	Dutch	nl
Arabic	ar	Faroese	fo	Norwegian	no
Catalan	ca	French	fr	Punjabi	pa
Corsican	co	Hebrew	he	Polish	pl
Czech	cs	Hindi	hi	Portuguese	pt
Danish	da	Croatian	hr	Russian	ru
German	de	Italian	it	Serbo-Croatian	sh
Greek	el	Inuktitut	iu	Swedish	sv
English	en	Japanese	ja	Turkish	tr
Spanish	es	Korean	ko	Urdu	ur
Persian	fa	Mongolian	mn	Yoruba	yo
Finnish	fi	Nepali	ne	Chinese	zh

The following table lists some of the common ISO 3166 two-letter country codes.

Country	**Code**	**Country**	**Code**
AUSTRALIA	AU	NEPAL	NP
AUSTRIA	AT	NETHERLANDS	NL
BRAZIL	BR	NEW ZEALAND	NZ
CANADA	CA	NORWAY	NO
CHINA	CN	OMAN	OM
CROATIA (local name: Hrvatska)	HR	PAKISTAN	PK
DENMARK	DK	PITCAIRN	PN
FIJI	FJ	POLAND	PL
FINLAND	FI	PORTUGAL	PT
FRANCE	FR	SAUDI ARABIA	SA
GERMANY	DE	SOUTH AFRICA	ZA
GREECE	GR	SPAIN	ES
HONG KONG	HK	SWEDEN	SE
INDIA	IN	SWITZERLAND	CH
IRAN (Islamic republic of)	IR	TAIWAN	TW
ISRAEL	IL	TURKEY	TR
ITALY	IT	UNITED ARAB EMIRATES	AE
JAPAN	JP	UNITED KINGDOM	GB
KOREA (Demo. people's republic of)	KP	UNITED STATES	US
MEXICO	MX	—	—

EXP

This scalar function returns *e* raised to a specific power.

Syntax

```
EXP( power )
```

Parameters

- power— The power (DOUBLE) to which to raise *e*.

Return Type

DOUBLE PRECISION.

Remarks

Returns *e* raised to the n^{th} power, where *e* = 2.71828183...

Example

`EXP(4)` raises *e* to the 4^{th} power and returns 54.59815.

LOG() returns the logarithm of a number from a specific base

POWER() raises a number to a specific power.

FLOOR

This scalar function returns largest integer less than or equal to an expression.

Syntax

```
FLOOR( numeric )
```

Parameter

- numeric— Number to floor.

Return Type

Numeric, same data-type as numeric argument.

Example

`FLOOR('1234.56')` returns the integer 1234.00, after first implicitly casting the string literal to a DECIMAL.

`FLOOR(-2.75)` returns -3.00.

Note: This function behaves identical to the Microsoft Excel INT() function.

CEIL() returns smallest integer rounded up.

gammaDist

This scalar function returns the gamma distribution of a value.

Syntax

```
gammaDist( number, alphaNumber, betaNumber, isCumulative )
```

Note: This function is provided as a sample UDF. To use it, you will need to load the function from the directory \samples\udf\jar\com\celequest\manifest on the distribution CD. For information on how to load UDFs, see "User-Defined Functions" on page 346.

Parameter

- number— Positive number to evaluate, may be zero (0).
- alphaNumber— Alpha parameter (positive number, may be zero) to the gamma distribution equation.
- betaNumber— Beta parameter (positive number, may be zero) to the gamma distribution equation.
- isCumulative— Boolean that determines the form of the function of number based on alphaNumber and betaNumber:
  - TRUE uses the cumulative distribution function.
  - FALSE uses the probability mass function.

Return Type

DOUBLE PRECISION.

Remarks

When alphaNumber is one (1), returns an *exponential distribution*.

When alphaNumber is a positive integer, the result is a *Erlang distribution*.

Example

Populate a view with the probability mass for a gamma distribution of alpha=9 and beta=2:

```
SELECT TimeToFail,
       gammaDist(TimeToFail, 9, 2, FALSE) AS GammaDist
    FROM UnitTests
    ORDER BY TimeToFail
```

logNormDist() returns the cumulative lognormal distribution of a value.

GREATEST

This scalar function returns the greatest of a list of expression results.

Syntax

```
GREATEST( value, value [, value … ] )
```

Parameters

- value— A value to be used for the comparison. All values after the first are converted to the data type of the first.

Return Type

Same data-type as argument.

Example

Selects the string with the greatest value:

```
SELECT Greatest(
        'SCHOLAR',
        'SKYLER',
        'SHUELLER')
    FROM Foo;

Greatest
--------
SKYLER
```

LEAST() determines the least value from a list.

MAX() returns the maximum value from a set.

“Function Types” on page 109 discusses moving sets.

IS_RAISED

This scalar function returns true when the specified alert is in a raised state.

Syntax

```
IS_RAISED('alertName')
```

Parameters

- alertName— Fully qualified name of an alert: the name must include the containing business activity and scenario names, like this 'activityName.scenarioName.alertName'.

Return Type

Boolean.

Remarks

Returns True if the alert exists and is in a raised state; otherwise, if the alert is in a lowered state or does not exist, it returns False.

Note: Because this function returns False when the alert does not exist, there is no test to ensure that the name you entered is a valid alert in the system; therefore, misspelling the name will cause the function to always return False.

Use this function in a rule condition to test the state of an alert and to generate a new alert when the tested alert remains raised for a period of time.

When used in a view definition, the view definition cannot have a set function.

See “Monitoring Alerts” on page 303 for a detailed discussion of using this function.

Alert States.

Monitoring Alerts.

LAST_DAY

This scalar function returns the date of the last day of the month that contains a specified date.

Syntax

```
LAST_DAY( dateTime )
```

Parameters

- dateTime— A valid date (TIMESTAMP).

Return Type

Date-Time.

Example

`LAST_DAY( CURRENT_TIMESTAMP() )` returns the date of the last day of the current month.

CURRENT_TIMESTAMP() returns the current date and time.

LEAST

This scalar function returns the least value of a list of expressions.

Syntax

```
LEAST( value, value [, value … ] )
```

Parameters

- value— A value to be used for the comparison. All values after the first are converted to the data type of the first.

Return Type

Same data-type as argument.

Example

```
SELECT Least(
        'SCHOLAR',
        'SKYLER',
        'SHUELLER')
    FROM Foo;

Least
------
SCHOLAR
```

GREATEST() determines the greatest value from a list.

MIN() returns the minimum value from a set.

“Function Types” on page 109 discusses moving sets.

LOG

This scalar function returns the logarithm of a number from a specific base.

Syntax

```
LOG( numeric [, base ] )
```

Parameters

- numeric— Number (DOUBLE) from which to retrieve the logarithm; must be greater than 1.
- base— Base (DOUBLE) of the logarithm; must be greater than zero (0). Omit this option to use the natural log of numeric.

Return Type

DOUBLE PRECISION.

Remarks

This can be expressed mathematically as "$\log_{\text{numeric}}$ base".

Example

`LOG(8,64)` returns 2.0.

`LOG(2)` returns 0.301029…

EXP() raise *e* to a specific power.

POWER() raises a value to a specific power.

logNormDist

This scalar function returns the cumulative lognormal distribution of a value.

Syntax

```
logNormDist( number, meanNumber, stdNumber )
```

Note: This function is provided as a sample UDF. To use it, you will need to load the function from the directory \samples\udf\jar\com\celequest\manifest on the distribution CD. For information on how to load UDFs, see "User-Defined Functions" on page 346.

Parameters

- number— Value to evaluate.
- meanNumber— Mean average of ln(number).
- stdNumber— Standard deviation of ln(number).

Return Type

DOUBLE PRECISION.

Remarks

Returns the cumulative lognormal distribution of a value, where *ln*(number) is normally distributed with mean and standard deviation.

gammaDist() returns gamma distribution of a value.

LOWER

This scalar function converts all uppercase characters in a string to lower case.

Syntax

```
LOWER( string )
```

Parameters

- string— String to convert.

Return Type

VARCHAR.

Example

`LOWER('Stage Right')` returns 'stage right'.

UPPER() converts to all uppercase.

LPAD

This scalar function inserts one or more instances of a string into the start of another string.

Syntax

```
LPAD( string, length, [ padChar ] )
```

Parameters

- string— Character or string to alter.
- length— The display length of the returned string. Must be zero (0) or greater. When using multi-byte characters, the length is the count of characters that display or print, not the count of multi-bytes.
- padChar— Character or string to insert. Default is a single space or blank character (‘ ’).

Return Type

VARCHAR.

Remarks

Returns a string in the same character type as the string parameter.

When length is smaller than the length of string, returns the string truncated to length.

Examples

`LPAD('ABC',6,'x')` returns ‘xxxABC’.

`LPAD('ABC',6,'xo')` returns ‘xoxABC’.

`LPAD('ABC',4) returns ‘ ABC’.`

`LPAD('ABC', 2, 'x') returns ‘AB’.`

RPAD() adds characters to the end of a string.

LTRIM

This scalar function removes characters from the start of a string.

Syntax

```
LTRIM( wholeString [, setString ] )
```

Parameters

- wholeString— String to trim.
- setString— Characters to remove; default is a single blank space (' ').

Return Type

VARCHAR.

Remarks

Recursively removes all instances of setString from the start of wholeString until wholeString no longer starts with setString, and returns the result.

Examples

`LTRIM(' ZZZ')` returns 'ZZZ'.

`LTRIM('aaaZZZ','a')` returns 'ZZZ'.

`LTRIM('ababaZZZ','ab')` returns 'aZZZ'.

`LTRIM('abcabaZZZ','abc')` returns 'abaZZZ'.

RTRIM() removes characters from the end of a string.

MAX

This set function returns the maximum value from a set.

Syntax

```
MAX( expression )
```

Parameters

- expression— An expression that evaluates to any data type and which cannot reference a rank function function. Typically the argument is a column in a view.

Return Type

Same data-type as expression argument.

Remarks

For Boolean, True is greater than False.

For String, 'z" is greater than 'A'.

Example

Return the maximum price from all the rows in Foo:

```
SELECT MAX( price ) FROM Foo;

PRICE
---------
770.00
```

Moving set semantics

When used as a MOV_function(), returns the maximum value for the moving set.

```
MOV_MAX( numeric, window, size [,timestampColumn] )
```

MOV_MAX() returns the maximum value from a moving window set.

TUMBLE_MAX() returns the maximum value from a tumbling window set.

MIN() returns the minimum value from a set.

GREATEST() returns the maximum value from a list.

median

This set function returns the median (middle) number in a set.

Note: This function is provided as a sample UDF. To use it, you will need to load the function from the directory \samples\udf\jar\com\celequest\manifest on the distribution CD. For information on how to load UDFs, see "User-Defined Functions" on page 346.

Syntax

```
median( numericExp )
```

Parameters

- numericExp— An expression that evaluates to numeric and which cannot reference a rank function function. Typically the argument is a column in a view.

Return Type

INTEGER when all results of numericExp are integer; otherwise DOUBLE PRECISION when any of the results are decimal.

Remarks

This function sorts the values in the set and then returns the median of the ordered set.

When the count of values is odd, the median is the middle number of the set. For example, the median of 2,1,5 is 2: the middle value of the ordered set.

Otherwise, when the count is even, the median is the average value of the two middle numbers in the set. For example, the median of 2,1,5,4 is 3: the average of 2 and 4. Further, when the result of the average is a decimal value, the result is "floored" to the integer: the median of 2 and 3 is 2, which is floor(2.5).

Ignores NULL values.

Examples

Consider this statement:

```
SELECT median(Value) AS MedianV FROM NumberList
```

The result is 1.5 (the average of 1.0 and 2.0 after ignoring the NULLs) when the items in NumberList are presented in this order:

```
3.0
NULL
0.0
2.0
1.0
NULL
NULL
```

The result is 1 if the set is

```
1
2
```

Because the values are integers, the result must also be an integer. As such the average which is 1.5 is floored to 1.

AVG() returns the mean average value of a set.

mode() returns the most frequently occurring number in a set.

MIN

This set function returns the minimum value from a set.

Syntax

```
MIN( expression )
```

Parameters

- expression— An expression that evaluates to any data type and which cannot reference a rank function function. Typically the argument is a column in a view.

Return Type

Same data-type as expression argument.

Remarks

For Boolean, True is greater than False.

For String, ‘z” is greater than ‘A’.

Example

Return the minimum price from all the rows in set Foo:

```
SELECT MIN( price ) FROM Foo;

PRICE
---------
100.00
```

Moving set semantics

When used as a MOV_function(), returns the minimum value for the moving set.

```
MOV_MIN( numeric, window, size [,timestampColumn] )
```

MOV_MIN() returns the minimum value from a moving window set.

TUMBLE_MIN() returns the minimum value from a tumbling window set.

MAX() returns the maximum value from a set.

LEAST() returns the smallest value from a list.

MOD

This scalar function returns the modulus (remainder) of a division.

Syntax

```
MOD( dividend, divisor )
```

Parameters

- dividend— Numeric to divide.
- divisor— Numeric to divide by.

Return Type

INTEGER.

Remarks

When divisor is zero (0), returns dividend.

This function behaves differently from the classical mathematical modulus function when dividend is negative. The classical modulus can be expressed with this formula:

```
dividend - divisor * FLOOR(dividend/divisor)
```

This function uses this updated formula:

```
SIGN(dividend) *
    (ABS(dividend) - ABS(divisor * FLOOR(ABS(dividend/divisor)) ) )
```

This table illustrates the difference between the MOD function and the classical modulus formula:

Dividend	Divisor	MOD (Dividend,Divisor)	Classical Modulus
11	4	3	3
11	-4	3	-1
-11	4	-3	1
-11	-4	-3	-3

Note: Most database management systems use the same formula as Business Activity Monitoring, while spreadsheet applications like Microsoft Excel use the classical modulus.

Example

Return the remainder of dividend divided by divisor:

```
SELECT MOD(11,4) "Modulus" FROM Foo;

Modulus
-------
3
```

mode

This set function returns the most frequently occurring number in a set.

Note: This function is provided as a sample UDF. To use it, you will need to load the function from the directory \samples\udf\jar\com\celequest\manifest on the distribution CD. For information on how to load UDFs, see “User-Defined Functions” on page 346.

Syntax

```
mode( numericExp )
```

Parameters

- numericExp— An expression that evaluates to numeric and which cannot reference a rank function function. Typically the argument is a column in a view.

Return Type

Same type as numericExp result.

Remarks

When multiple different values occur with the same frequency, mode() returns the first one it encountered. See the example.

Ignores NULL values.

Examples

Consider this set of numbers, fed into mode() in this order:

```
1
3
4
1
3
```

The mode() function returns 1 because it occurs with the most frequency, and is encountered before 3, which occurs with the same frequency. Had the set been fed into mode() in reverse order, it would have returned 3.

For this set of numbers, mode returns 1.0:

```
1
3
NULL
2.0
NULL
NULL
1
```

The NULLs are ignored, and the 2.0 causes mode() to return a DOUBLE PRECISION value.

See Also

AVG() returns the mean average number in a set.

median() returns the median (middle value) for a set.

"C-SQL Function" on page 114 for a complete list of functions.

MOV_function

This scalar function limits the rows used in a set function calculation to a set of the latest rows in the view.

Moving Set Functions

The moving window set functions are:

Argument	Description
MOV_AVG	Returns the moving average value (arithmetic mean) of a moving window set of numeric values.
MOV_COUNT	Returns the count of rows in a moving window set.
MOV_MAX	Returns the maximum value from a moving window set.
MOV_MIN	Returns the minimum value from a moving window set.
MOV_SUM	Returns the sum of a moving window set of numeric values.
MOV_STD_DEVIATION	Returns sample standard deviation of a moving window set of numbers.
MOV_VARIANCE	Returns the square of the sample standard deviation of a moving window set of numbers.

Note: Moving set functions are shorthand for simple query windows. For a complete discussion, see "Query Windows" on page 272.

Syntax

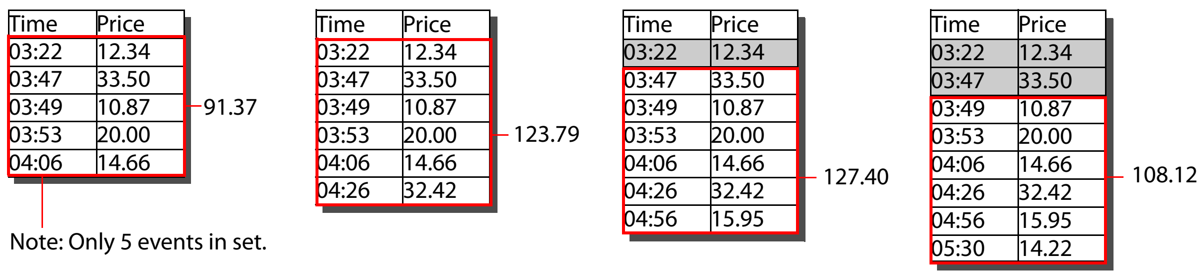
All moving set functions have this syntax:

```
mov_function( numeric, window, size [,timestampColumn] )
```

where `function` is an existing set function. The return type of the moving function is the same as that of the named function. The numeric is typically a column in the view, but may contain other functions and operators, though it cannot reference a rank function.

The window and size arguments specify which rows are included in the set. The window argument determines if size is the count of rows in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR). For example a set of the last 6 events limits the set to no more than 6 events (per group when using the GROUP BY Clause). Note that some events generate multiple rows; do not confuse events with rows.

`MOV_SUM(Price, EVENT, 6)`



Time	Price
03:22	12.34
03:47	33.50
03:49	10.87
03:53	20.00
04:06	14.66

Time	Price
03:22	12.34
03:47	33.50
03:49	10.87
03:53	20.00
04:06	14.66
04:26	32.42

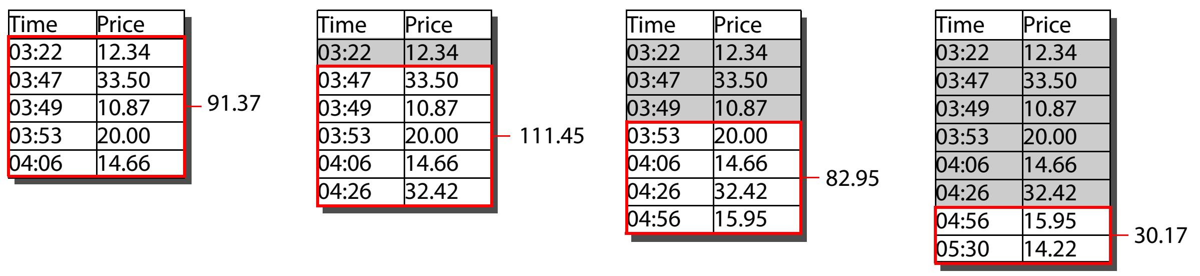
Time	Price
03:22	12.34
03:47	33.50
03:49	10.87
03:53	20.00
04:06	14.66
04:26	32.42
04:56	15.95

Time	Price
03:22	12.34
03:47	33.50
03:49	10.87
03:53	20.00
04:06	14.66
04:26	32.42
04:56	15.95
05:30	14.22

Complete window expression:
`SUM(Price) OVER (EVENT '5' PRECEDING REFERENCE FRAME)`

When using time-series span (instead of event span), the size of the set varies depending on when the events were recorded in the view. For example when using a time-series of 1 hour, only those rows that entered the view in the last hour are used in the calculation.

`MOV_SUM(Price, HOUR, 1)`



Time	Price
03:22	12.34
03:47	33.50
03:49	10.87
03:53	20.00
04:06	14.66

Time	Price
03:22	12.34
03:47	33.50
03:49	10.87
03:53	20.00
04:06	14.66
04:26	32.42

Time	Price
03:22	12.34
03:47	33.50
03:49	10.87
03:53	20.00
04:06	14.66
04:26	32.42
04:56	15.95

Time	Price
03:22	12.34
03:47	33.50
03:49	10.87
03:53	20.00
04:06	14.66
04:26	32.42
04:56	15.95
05:30	14.22

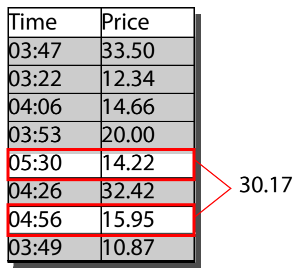
Complete window expression:
`SUM(Price) OVER (RANGE '1' HOUR PRECEDING REFERENCE OPERATOR)`

Note: It is possible, but highly unlikely, for an event to arrive in the system in time to be included in a view, but to be discarded because by the time it reaches the view, it is no longer in the view's time span. For example, if the event enters the event table a few milliseconds before it would excluded from a derived view, it might be included or excluded depending on how long it takes to process and propagate the event in base views.

The optional timestampColumn argument instructs the system to use the value of a field in the view as the reference point for starting the time-series span. When you omit this option, the system calculates the time-series based on the system clock, such as the last hour from now. When you name a timestamp column instead, the calculation is based on the time span from the value in the most recent value in the column from any row in the view.

`MOV_SUM(Price, HOUR, 1, Time)`

Events of the last hour from the most recent event based on its *Time* column value, not necessarily the order that the events entered the view.



Time	Price
03:47	33.50
03:22	12.34
04:06	14.66
03:53	20.00
05:30	14.22
04:26	32.42
04:56	15.95
03:49	10.87

Complete window expression:
```
SUM(Price) OVER (ORDER BY Time
                 RANGE '1' HOUR PRECEDING
                 REFERENCE OPERATOR)
```

Time-Series Spans

Time span calculations use the Gregorian calendar and are calculated to the second that the event was recorded in the system (recorded in the vc_timestamp column). For example, if the span is 1 day and an event arrives just before midnight, it excludes almost all events on the previous day; however, an event arriving just after midnight includes almost all of the events on the previous day. Here are some additional semantics:

- In locales where daylight savings time is observed, durations of days, months, and years are adjusted accordingly. As such, while 1 day is typically 24 hours long, it may be 23 or 25 hours depending on the time of year.
- Month calculations are based on the day of the month: a 1 month span on 5 April includes all dates after 5 March. When the day of the month does not exist at the start of the window, the end of the month is used. For example, a 1 month span on 31 May starts after 30 April.
- Similarly, year calculations are based on the day of the year, and adjust as necessary for leap years.
- For the purposes of parallel execution, you may choose to not process events in the order in which they are timestamped. In this case the order of the data within a group is arbitrary and will only produce approximate Moving Set values that may not be reproducible for the same input events during a subsequent evaluation of the same set.
- The set of events included in a moving window view are determined when a new event enters the view. Events that are filtered out of a view before they enter the view, such as when excluded by a WHERE Clause, do not affect the view and do not cause the view to update.

View Warning

Do not use a moving set function in a derived view to perform a calculation on a moving set function result in a base view. This is because the derived function will always return the current value in the base view, regardless of the span of the window. If you need such a aggregation, place the functions in the same view. See the example in "AVG" on page 118 for details.

Interacting with GROUP BY

Using a moving set function on a view defined with a GROUP BY Clause populates the groups as follows:

Time-series spans

Time-series spans apply to all events in all groups. Only events that fall within the time span are included in the groups. Events that do not meet time span definition are excluded. When all events have been removed from a group set, the group is empty. If no other columns retain the group, it is removed from the view. Consider this example where average prices are tracked in groups by product for the last hour. When a Product no longer has events in the last hour, that Product's group is removed.

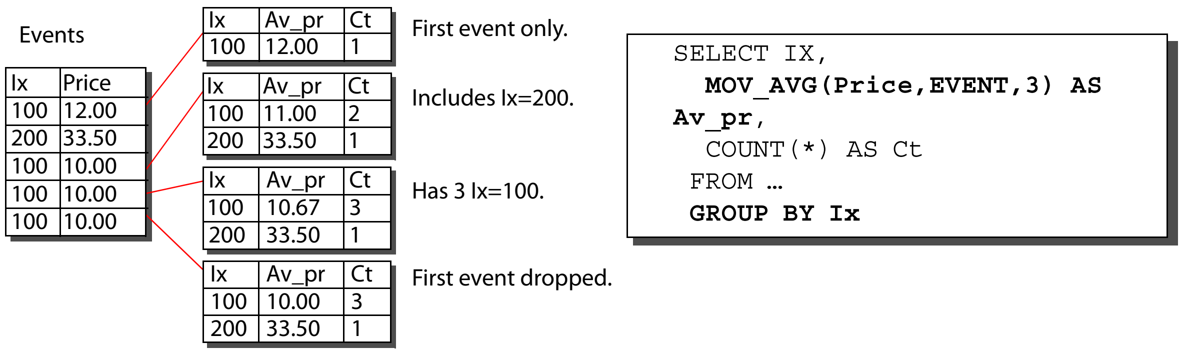
```
SELECT MOV_AVG(Price,HOUR,1) AS Av_pr
  FROM ...
  GROUP BY Product
```

However, in this variation the presence of the SUM() function causes the view to retain every event group, but the average price for a group of the last hour may be empty. Querying an empty group returns NULL.

```
SELECT MOV_AVG(Price,HOUR,1) AS Av_pr,
       SUM(Price) AS Total
  FROM ...
  GROUP BY Product
```

Event spans

Event spans apply their size to each group in the view; each group tracks a count of events determined by the size of the span. Groups are never removed from the view, and events are removed from the sets only when they are pushed out by a newer event. Consider the view in the example shown in the illustration below, where MOV_AVG() tracks up to 3 events per group. When the fourth event whose Ix value is 100 is inserted, the first is dropped from the moving average calculation of the Ix=100 group. However, within this example, once the 200 group is created, its set remains constant with the one event:



Ix	Price
100	12.00
200	33.50
100	10.00
100	10.00
100	10.00

Ix	Av_pr	Ct
100	12.00	1

Ix	Av_pr	Ct
100	11.00	2
200	33.50	1

Ix	Av_pr	Ct
100	10.67	3
200	33.50	1

Ix	Av_pr	Ct
100	10.00	3
200	33.50	1

```
SELECT IX,
  MOV_AVG(Price,EVENT,3) AS
Av_pr,
  COUNT(*) AS Ct
 FROM ...
 GROUP BY Ix
```

For event spans that have events with multiple rows in the view, the entire event is treated as one item in the set based on the timestamp (vc_timestamp) and event ID (vc_event_id).

Note: All columnNames referred directly by a rank function or scalar function must appear in the set of columns listed in the GROUP BY list.

MOV_AVG

This moving set function returns the moving average value (arithmetic mean) of a moving window set of numeric values.

Syntax

```
MOV_AVG( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

Return Type

Numeric, same data-type as numeric argument.

Remarks

Returns NULL if the group is empty.

Example

Return the average price of all the events that arrive within a 7 day interval:

```
MOV_AVG(price, DAY, 7, trade_time) AS Avg_7_day_price
```

"Function Types" on page 109 discusses moving sets.

AVG() returns the mean average for a set.

TUMBLE_AVG() returns the tumbling average for a set.

MOV_COUNT

This moving set function returns the count of rows in a moving window set.

Syntax

```
MOV_COUNT( *, window, size [,timestampColumn] )
```

Parameters

- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

INTEGER.

Remarks

Returns zero (0) if the set is empty.

Rows that include NULLs are counted.

When used with a GROUP BY returns the count of rows in the group set. See the discussion and example in "Interacting with GROUP BY" on page 153 for more information.

Example

Return the count of all the events that arrive within the current 8 hour interval:

```
MOV_COUNT(*, HOUR, 8, trade_time) AS Total
```

The above function is shorthand for this in-line window expression:

```
COUNT(*) AS Total OVER (ORDER BY trade_time '8' HOUR)
```

This expression is equivalent to the following after filling in all default values:

```
COUNT(*) AS Total OVER ( ORDER BY trade_time
    RANGE INTERVAL '8' HOUR PRECEDING
    REFERENCE OPERATOR)
```

Note: The eight-hour window begins when the first event arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE '2003-03-05 00:00:00.000'.

"Function Types" on page 109 discusses moving sets.

COUNT() returns the count of a view or set.

TUMBLE_COUNT() returns the count of a tumbling window set.

MOV_MAX

This moving set function returns the maximum value from a moving window set.

Syntax

```
MOV_MAX( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

Same data-type as expression argument.

Remarks

For Boolean, True is greater than False.

For String, 'z" is greater than 'A'.

Example

Return the maximum price of all the events that arrive within a 7 day interval:

```
MOV_MAX(price, DAY, 7, trade_time) AS Max_7_day_price
```

MAX() returns the maximum value from a moving window set.

TUMBLE_MAX() returns the maximum value from a tumbling window set.

MIN() returns the minimum value from a set.

GREATEST() returns the maximum value from a list.

"Function Types" on page 109 discusses moving sets.

MOV_MIN

This moving set function returns the minimum value from a moving window set.

Syntax

```
MOV_MIN( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

Same data-type as expression argument.

Remarks

For Boolean, True is greater than False.

For String, 'z" is greater than 'A'.

Example

Return the minimum price of all the events that arrive within a 7 day interval:

```
MOV_MIN(price, DAY, 7, trade_time) AS Min_7_day_price
```

MIN() returns the minimum value from a view or set.

TUMBLE_MIN() returns the minimum value from a tumbling window set.

MAX() returns the maximum value from a set.

LEAST() returns the smallest value from a list.

"Function Types" on page 109 discusses moving sets.

MOV_SUM

This moving set function returns the sum of a moving window set of numeric values.

Syntax

```
MOV_SUM( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

Numeric, same data-type as numeric argument. Returns NULL if the set is empty.

Example

Total the price of all events that arrive in current hour:

```
MOV_SUM(Price, HOUR, 1)
```

Which is shorthand for this in-line window expression:

```
SUM(Price) OVER (RANGE '1' HOUR PRECEDING
                     REFERENCE OPERATOR)
```

SUM() returns the sum of a view or set.

TUMBLE_SUM() returns the sum of a tumbling window set.

"Function Types" on page 109 discusses moving sets.

MOV_STD_DEVIATION

This moving set function returns sample standard deviation of a moving window set of numbers.

Syntax

```
MOV_STD_DEVIATION( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the most recent value in the column from any row in the view. Omit this option to use the system clock as the time basis.

Return Type

DOUBLE PRECISION.

Remarks

Returns 0 when there is only a single row of input. Returns NULL if the set is empty.

The result is computed using the formula $\sqrt{\sum\left[\frac{(X_i - X)^2}{n}\right]}$, where *n* is the number of elements in the sample and *X* is the sample mean.

Example

Return the standard deviation in salaries for the last year’s worth of events:

```
SELECT MOV_STD_DEVIATION(salary, YEAR, 1) AS "Dev. for the last year"
    FROM employees;
```

Which in turn is equivalent to the following after filling in all default values:

```
SELECT STD_DEVIATION(salary) AS "Dev. for the last year" OVER (
    RANGE INTERVAL '1' YEAR PRECEDING
    REFERENCE OPERATOR)
```

STD_DEVIATION() returns the standard deviation of a view or set.

TUMBLE_STD_DEVIATION() returns the standard deviation of a tumbling window set.

VARIANCE() returns the square of the standard deviation.

“Function Types” on page 109 discusses moving sets.

MOV_VARIANCE

This moving set function returns the square of the sample standard deviation of a moving window set of numbers.

Syntax

```
MOV_STD_DEVIATION( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

DOUBLE PRECISION.

Remarks

Returns zero (0) when the expression set contains only one element. Returns NULL if the set is empty.

The result is computed using the formula $\sum\left[\frac{(X_i - X)^2}{n}\right]$, where *n* is the number of elements in the sample and *X* is the sample mean.

Example

Return the variation in salaries for each calendar year:

```
SELECT MOV_STD_VARIATION(salary, YEAR, 1) AS "Variation for last year"
    FROM employees;
```

Which in turn is equivalent to the following after filling in all default values:

```
SELECT VARIATION(salary) AS "Variation for last year" OVER (
    RANGE INTERVAL '1' YEAR PRECEDING
    REFERENCE OPERATOR)
```

VARIANCE() returns the variance of a view or set.

STD_DEVIATION() returns a standard deviation.

TUMBLE_VARIANCE() returns the variance of a tumbling window set.

“Function Types” on page 109 discusses moving sets.

NTILE

This rank function determines the tier rank of each value in a set with respect to the entire set.

Syntax

```
NTILE( toRank, tiers )
```

Parameters

- toRank— A expression of any data type, and which typically references a column of values to rank.
- tiers— Count of tiers in which to partition the results; an integer greater than zero (0). When this value is greater than the count of items to rank, all items are given the same rank.

Return Type

INTEGER.

Remarks

Returns an integer for each row in the set that represents the tier that the row belongs to, where one (1) is the highest tier holding the greatest value. When toRank results in NULL, that result is assigned to the lowest rank.

Note: This function *cannot* be used as an argument in a set function, moving set function, or rank function. For example, SUM(NTILE()) is illegal.

An *n*tile function ranks rows by attempting to evenly distribute them throughout a fixed set of tiers. For example, when there is a set of six expression results {D, B, E, C, A, and B} to rank into two tiers, NTILE() assigns each a rank of either 1 (for C, D, and E) or 2 (for A, B, and B). Results with the same value are always placed in the same tier.

When a set of values is not divisible by the tiers, the function evenly distributes any leftover rows into higher-level groups. For example, the following table demonstrates how the letter items are distributed into various counts of tiers:

tiers:	1	2	3	4	5	6	7
A	1	2	3	3	4	5	1
B	1	2	2	2	3	4	1
B	1	2	2	2	3	4	1
C	1	1	2	1	2	3	1
D	1	1	1	1	1	2	1
E	1	1	1	1	1	1	1

Example

The following query ranks sales of coffee and tea products into six tiers by their sales rankings. The ranking is in sixths, so each product name receives a value from 1 to 6. This example requires that there be just one unique entry for each product:

```
SELECT prod_name, NTILE( dollars, 6) AS sales_rank
    FROM (lineitem INNER JOIN product
          ON lineitem.item_id=product.productid)
    WHERE product.classkey IN (1, 2, 4, 5);

PROD_NAME           SALES_RANK
Demitasse M         1
Xalapa Lapa         1
Cafe Au Lait        2
Aroma Roma          2
Veracruzano         3
Colombiano          3
Darjeeling Special  4
Irish Breakfast     4
English Breakfast   5
Earl Grey           5
Gold Tips           6
```

RANK() ranks rows within the entire set.

"Function Types" on page 109 discusses moving sets.

POSITION

This scalar function returns the position of a character or string within a string.

Syntax

`POSITION( sourceForString , searchInString )`

Alternate form: `POSITION( sourceForString IN searchInString )`

Parameters

- sourceForString— String to search for.
- sourceInString— String expression result in which to search.

Return Type

INTEGER.

Remarks

Returns the position, starting from 1, of the 1st instance of sourceForString in the sourceInString result. When CHARACTER_LENGTH(sourceForString) is zero (0), returns 1. Returns NULL when either argument is NULL.

Examples

`POSITION( 'a' IN 'banana')` returns 2.

`POSITION( 'ana' IN 'banana')` returns 2.

`POSITION( 'A' IN 'banana')` returns 0.

`POSITION( 'M' IN 'banana')` returns 0.

`POSITION( '' IN 'banana')` returns 1.

`POSITION( NULL IN 'banana')` returns NULL.

SUBSTRING() extracts a character or substring from a string.

POWER

This scalar function returns a value raised to a specific power.

Syntax

```
POWER( numeric, power )
```

Parameters

- numeric— Number to raise.
- power— Power to raise numeric. Must be an integer when numeric is negative.

Return Type

Same data-type as numeric argument.

Remarks

This can be expressed mathematically as "number^{power}".

Example

`POWER(3,5)` returns 243.

EXP() raise *e* to a specific power.

LOG() returns the logarithm of a number from a specific base.

PREV

This set function returns a value from the next to last row in a set.

Syntax

```
PREV( columnName )
```

Parameters

- columnName— Column or alias of any data type to evaluate.

Return Type

Same data-type as argument.

Remarks

Returns values from the row before the “current” row in a set, where the current row is the latest row in the set based on the event timestamp, or when all have the same timestamp, is last row in the set.

Example

Gather all stock feed closing prices and group them by stock symbol. The “current” row will always be the last one received, and as such, will contain the current closing price. The previous row will be the previous day’s close:

```
SELECT symbol, CURRENT(close) AS "Last Trade",
        PREV(close) AS "Prev Cls",
        (CURRENT(close) - PREV(close)) AS Change
    FROM Stock_feed
    GROUP BY symbol

symbol  Last Trade  Prev Cls  Change
------- ----------- --------- -------
K       31.25       31.28     -0.03
IBM     80.79       80.04     0.75
VCLR    24.42       22.60     1.82
```

Moving set semantics

Cannot be used with a moving or tumbling set.

CURRENT() returns the value from the latest or last row in a set.

PRIOR_VALUE() returns the prior value of a column, alias, or expression.

“Function Types” on page 109 discusses moving sets.

PRIOR_VALUE

This scalar function returns the prior value of a column, alias, or expression.

Syntax

```
PRIOR_VALUE( columnName )
```

Parameters

- columnName— Column or alias of any data type to evaluate.

Return Type

Same data-type as argument.

Remarks

Returns a NULL if there is no prior value — the first time the function is called on the columnName.

Use PRIOR_VALUE() when the data (events) enter the system grouped and ordered,

This function is not permitted in the WHERE Clause of a view definition.

Example

Consider this query that identifies the how long a task took to complete — as a percentage of an hour — based on minutes since the previous task completed:

```
SELECT Task, CAST(
      (TIMESTAMP_DIFF(PRIOR_VALUE(Completed), Completed, MINUTE ) /60
                  , DECIMAL(5,4)
                 ) AS Hours
    FROM Tasks_Completed
Task               Hours
---------------    ------
Startup
Initialize         0.0887
Begin job          0.1012
Finish job         4.3243
Clean up           0.2500
Shut down          0.1285
Have milk shake    0.6667
```

PREV() returns a value from the next to last row in a set.

RANK

This rank function determines the rank of each value in a set with respect to the entire set.

Syntax

```
RANK( expression )
```

Parameters

- expression— A expression of any data type, and which typically references a column.

Return Type

INTEGER.

Remarks

Returns an integer for each row in the set that is the row's ranking within the entire set, where the greatest value is ranked 1. When expression results in NULL it is ranked last in the result list. For example, the ranking of (10, NULL, 20) ranks the 10 as 2, the 20 as 1, and NULL as 3.

When the values to be ranked are equal, they are assigned the same rank, and the next rank is skipped. For example values 4.5, 4.5, 1.0 will be assigned rank values of 1, 1, and 3 respectively.

Note: This function *cannot* be used as an argument in a set function, rank function, or moving set function. For example, SUM(RANK(…)) is illegal. Nor can RANK() be used on an stateless view.

When the set contains only one row, RANK() returns 1. For example, RANK(SUM(sales)) = 1.

Examples

Rank product sales by region:

```
SELECT RANK( SUM( sales ) ) AS R, SUM(sales) AS S, region
    FROM product_orders
    GROUP BY region
R   S        region
1   100000   north
2   50000    south
```

Rank product sales by product:

```
SELECT prod_name, SUM(dollars) AS prod_sales,
    RANK( SUM(dollars) ) AS prod_rank
    FROM product, lineitem
    WHERE lineitem.classkey = product.classkey
        AND lineitem.prodkey = product.prodkey
    GROUP BY prod_name;
PROD_NAME       PROD_SALES   PROD_RANK
Demitasse Ms    656401.50    1
Xalapa Lapa     577450.00    2
Aroma Roma      479330.25    5
Verona          467234.00    6
NA Lite         557655.00    3
Lotta Latte     533454.50    4
```

NTILE() ranks rows and places them in a finite set of tiers.

"Function Types" on page 109 discusses moving sets.

RATIO_TO_REPORT

This rank function calculates the ratio of a value to the sum of the values for the entire set.

Syntax

```
RATIO_TO_REPORT( numeric )
```

Parameters

- numeric— Any numeric data type expression, typically a reference a numeric column.

Return Type

DOUBLE PRECISION.

Remarks

Returns an number for each row in the set that is the row's ratio to the sum of the entire set. When expression results in NULL, the function returns NULL. When the sum of the set is zero (0), the ratio is also zero.

Note: This function *cannot* be used as an argument in a set function, moving set function, or rank function. For example, SUM(RATIO_TO_REPORT(...)) is illegal.

Example

Determine what percentage each product sales is to the total sales of all products, for the last 20 weeks:

```
SELECT prod_description DESC,
        SUM(dollars) as sales,
        RATIO_TO_REPORT( SUM( li_amount )) * 100 AS ratio_dollars
    FROM lineitem, product
    WHERE lineitem.li_prod_id = product.prod_id
    GROUP BY prod_description;

DESC        SALES          RATIO_DOLLARS
Widget      896931.15      12.88
Basket      514830.00      7.39
Football    507022.35      7.28
Oil Drum    503493.10      7.23
Computer    437863.00      6.29
Chair       429637.75      6.17
Desk        424215.00      6.09
Mesh Bag    421205.75      6.05
Shoelace    417261.00      5.99
Powder      397102.50      5.70
Telephone   394086.50      5.66
Cord        392377.75      5.64
Mouse       389378.25      5.59
Monitor     305859.75      4.39
Case        294982.75      4.24
Cup         236772.75      3.40
```

ROUND

This scalar function returns a number rounded up to a specified count of decimal places.

Syntax

```
ROUND( number, [ places ] )
```

Parameters

- number— The numeric expression to round
- places— Count of decimal places to round to. A negative integer rounds to whole number digits. Default is zero (0) to remove any fractional components.

Return Type

Numeric, same data-type as number argument.

Examples

`ROUND(1294.5078)` returns 1294.

`ROUND(1294.5078, 0)` returns 1294.

`ROUND(1294.5078, 1)` returns 1294.5.

`ROUND(1294.5078, 2)` returns 1294.51.

`ROUND(1294.5078, -2)` returns 1300.

TRUNC() removes digits from a number.

RPAD

This scalar function adds one or more instances of a string to the end of another string.

Syntax

`RPAD(` string`,` length`,` `[` padChar `] )`

Parameters

- string— Character or string to alter.
- length— The *display length* of the returned string. When using multi-byte characters, the length is the count of characters that display or print, not the count of multi-bytes.
- padChar— Character or string to append. Default is a single space or blank character (‘ ’).

Return Type

VARCHAR.

Remarks

When length is smaller than the length of string, returns the string truncated to length.

Examples

`RPAD('ABC', 6, 'x')` returns ‘ABCxxx’.

`RPAD('ABC', 6, 'xo')` returns ‘ABCxox’.

`RPAD('ABC', 4)` returns ‘ABC ’.

`RPAD('ABC', 2, 'x')` returns ‘AB’.

LPAD() inserts characters to the start of a string.

RTRIM

This scalar function removes characters from the end of a string.

Syntax

```
RTRIM( sourceString [, setString ] )
```

Parameters

- sourceString— String to trim.
- setString— Characters to remove; default is a single blank space (' ').

Return Type

VARCHAR.

Remarks

Recursively removes all instances of setString from the end of sourceString until sourceString no longer ends with set, and returns the result.

Examples

`RTRIM('ZZZ ')` returns 'ZZZ'.

`RTRIM('ZZZaaa', 'a')` returns 'ZZZ'.

`RTRIM('ZZZababab', 'ab')` returns 'ZZZ'.

`RTRIM('ZZZababc', 'abc')` returns 'ZZZab'.

LTRIM() removes characters from the start of a string.

SAFE_DIVIDE

This scalar function returns a quotient determined by a dividend, a divisor, and an alternate divide-by-zero quotient. If the divisor is non-zero, this function returns the quotient of the dividend divided by the divisor. If the divisor is zero, or there is some other runtime data error while calculating the quotient, such as a DECIMAL out of range, the alternate quotient is returned.

Note: The recommended practice is use decimals for both dividend and divisor values.

Syntax

`SAFE_DIVIDE(numberDividend,numberDivisor,numberAlternateQuotient)`

Parameters

- numberDividend— The numerical value to be divided.

  The data type is determined by how the value is enter. If the value is entered without a decimal (e.g. 19), the value is cast as an integer. If the value is entered with (e.g. 19.00), the value is cast as an integer.
- numberDivisor— The numerical vlaue by which the divident is divided.

  This value is automatically cast as the same data type as the numberDividend.
- numberAlternatQuotient— The numerical value to be returned if the resulting quotient is 0 or if t

Return Type

Numeric, same data-type as number argument.

Examples

`SAFE_DIVIDE(100.20,4321)` returns 5.

`SAFE_DIVIDE(100.19.5,4321)` returns 5. The 19.5 is cast as an integer with a value of 19.

`SAFE_DIVIDE(100.00,19.5,4321)` returns 5.12. The quotient is carried to the same number of decimal points as the numberDividend, in this case two decimal points.

`SAFE_DIVIDE(SAFE_DIVIDE(100.000,19.5,4321)` returns 5.128. The quotient is carried to the same number of decimal points as the numberDividend, in this case three decimal points.

`SAFE_DIVIDE(100.0,4321)` returns 4321. Because the quotient of the numberDividend and numberDivisor has a value of 0, the numberAlternateQuotient value is returned.

SIGN

This scalar function identifies the arithmetic sign of a number.

Syntax

```
SIGN( number )
```

Parameters

- number— The numeric value to evaluate.

Return Type

INTEGER.

Remarks

Returns 1 if the number is positive, 0 if the number is zero, and -1 if the number is negative.

Note that these expressions return identical results:

```
 (number * SIGN(number) )
ABS(number)
```

ABS() returns the absolute value of a number.

SQRT

This scalar function returns the square root of a number.

Syntax

```
SQRT( number )
```

Parameters

- number— The number (DOUBLE) to evaluate. Must be greater than zero (0).

Return Type

DOUBLE PRECISION.

Example

`SQRT(42)` returns 6.480…

SUBSTRING

This scalar function returns a substring of a specified string.

Syntax

```
SUBSTRING( string, position, [ length ] ).
```

Alternate forms:

```
SUBSTR( string, position, [ length ] )
SUBSTRING( string FROM position [ FOR length ]  )
```

Parameters

- string— Character string to search.
- position— Starting position of the substring, where 1 is the first character at the start of the string, and -1 is the last character. Negative values count backwards from the end of the string. Using zero (0) is the same as using 1. A position not within string returns an empty string.
- length— Length of the substring to extract. Omitting length returns all characters from position through the end of the string. Specifying a value greater than the remainder of the string returns all characters from position through the end of the string, and pads the difference with space characters to achieve the specified length. A negative value or zero (0) returns an empty string.

Return Type

VARCHAR.

Examples

`SUBSTR('breakfast', 1)` returns 'breakfast'.

`SUBSTR('breakfast', 0)` returns 'breakfast'.

`SUBSTR('breakfast', 30)` returns '' (empty string).

`SUBSTR('breakfast', 1, 2)` returns 'br'.

`SUBSTR('breakfast', CHARACTER_LENGTH('breakfast'), 2)` returns 't '.

`SUBSTR('breakfast', 3, 4)` returns 'eakf'.

`SUBSTR('breakfast', 3, 8)` returns 'eakfast'.

`SUBSTR('breakfast', -5, 4)` returns 'kfas'.

`SUBSTR('breakfast', 1, -1)` returns '' (empty string).

CHARACTER_LENGTH() returns the length of a character string.

POSITION() locates a character within a string.

SUM

This set function returns the sum of a set of numeric values.

Syntax

```
SUM( numeric )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view.

Return Type

Numeric, same data-type as numeric argument. Returns NULL if the set is empty.

Example

Total the invOnHand column for all rows in the stock table:

```
SELECT SUM(invOnHand) "Total on hand"
    FROM stock;

Total on hand
-------------
2
```

Moving set semantics

When used as a MOV_function(), returns the moving sum for the moving set.

MOV_SUM(*numeric*, *windowwindow*, *size* [,*timestampColumn*])

Last value in the set

When the moving set size is a single event, MOV_SUM() maintains the sum of the last order prices for each customer, for all the customers that have placed an order since the system startup time.

```
SELECT os.os_cust_id, MOV_SUM(os.os_price, EVENT, 1)
    FROM order_status os
    GROUP BY os.os_cust_id
```

MOV_SUM() returns the sum of a moving window set.

TUMBLE_SUM() returns the sum of a tumbling window set.

SUM_OVER_GROUPS

This is rank function that is passed two parameters, a numeric value followed by an ordering parameter. When the groups are ordered by the ordering parameter, the function returns a running sum of the numeric values.

Syntax

```
SUM_OVER_GROUPS([numeric value],[ordering parameter])
```

Parameters

- numeric value—
- ordering parameter—

Return Type

Running sum of the numeric values provided in the arguments.

Example

Consider the following data set

```
grp  data
----------
1    1
2    3
3    10
4    200
```

If the following was issued:

```
SELECT grp, SUM_OVER_GROUPS(data, grp) as running_sum GROUP BY grp
```

The result would be:

```
grp  running_sum
----------
1    1
2    4
3    14
4    214
```

STD_DEVIATION

This set function returns sample standard deviation of a set of numbers.

Syntax

```
STD_DEVIATION( number )
```

Parameters

- number— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view.

Return Type

DOUBLE PRECISION.

Remarks

Returns 0 when there is only a single row of input. Returns NULL if the set is empty.

The result is computed using the formula $\sqrt{\sum\left[\frac{(X_i - X)^2}{n}\right]}$, where *n* is the number of elements in the sample and *X* is the sample mean.

Example

```
SELECT STD_DEVIATION(salary) "Deviation"
    FROM employees;

Deviation
----------
3909.36575
```

MOV_STD_DEVIATION() returns the standard deviation of a moving window set.

TUMBLE_STD_DEVIATION() returns the standard deviation of a tumbling window set.

VARIANCE() returns the square of the standard deviation.

TIMESTAMP_DIFF

This scalar function returns, as an absolute value, the interval of time between two timestamps.

Syntax

```
TIMESTAMP_DIFF( startTime, endTime, unit )
```

Parameters

- startTime— Start Date-Time.
- endTime— End Date-Time.
- unit— Type of the time interval to return; one of these literals:
  - SECOND
  - MINUTE
  - HOUR
  - DAY
    MONTH (30 days)
    YEAR (12 months or 360 days)

Return Type

INTEGER.

Remarks

Uses absolute time difference rounded up to the nearest whole value; *does not use* Gregorian calendar arithmetic.

Rounds the result to the nearest integer. For example, the difference between 10:00 and 10:29 in HOUR units is zero (0), but 10:00 and 10:30 return one (1).

Returns NULL if either timestamp is NULL.

Example

Return the count of days from now until the end of the month.

```
TIMESTAMP_DIFF( CURRENT_TIMESTAMP(),
                LAST_DAY( CURRENT_TIMESTAMP() ), DAY)
```

Return True when a problem ticket is open for more than 30 minutes:

```
TIMESTAMP_DIFF( ticket_opened, CURRENT_TIMESTAMP(), MINUTE) > 30
```

Return the number of days between two dates as a positive number, regardless of the which date is oldest:

```
ABS( TIMESTAMP_DIFF( father_birthdate, mother_birthdate, DAY) )
```

DATE_ADD() adds a duration to a date-time.

DATE_DIFF() subtracts a duration from a date-time.

CURRENT_TIMESTAMP() returns the current date and time.

TO_CHAR

This scalar function converts a date-time to a character string.

Syntax

```
TO_CHAR( date, [ format ] )
```

Parameters

- date— Date-Time value to convert.
- format— Date pattern of string identical to the one used by the Java SimpleDateFormat class, and is described in "Date-Time Formatting" on page 66. Omit this option to convert using the default format, which is "yyyy-MM-dd HH:mm:ss.SSSSSSSSS".

Return Type

VARCHAR.

Remarks

See "Converting Between Date-Time and Strings" on page 62 for a complete discussion about the conversion.

Examples

`TO_CHAR(CURRENT_TIMESTAMP(), 'd MMMM yy')` returns '5 March 03'.

CAST() converts one data type to another.

TO_DATE() converts a character string to a date.

CURRENT_TIMESTAMP() returns the current date and time.

TO_DATE

This scalar function converts a character string to a date-time value.

Syntax

```
TO_DATE( string, [ format ] )
```

Parameters

- string— Date string (VARCHAR) or literal to convert.
- format— Date pattern of string identical to the one used by the Java SimpleDateFormat class, and is described in "Date-Time Formatting" on page 66. Omit this option to convert using the default format, which is "yyyy-MM-dd HH:mm:ss.SSSSSSSSS".

Return Type

Date-Time.

Remarks

Omitting the time values from the pattern zero-fills (0) the portion of the TIMESTAMP thereby setting the time to midnight.

Examples

`TO_DATE('2003-02-18')` is identical to `TO_DATE('2003-02-18', "yyyy-MM-dd").`

`TO_DATE('2003-02-18 12:00:00', 'yyyy-MM-dd HH:mm:ss')` assigns noon as the time.

To strip the time portion off a TIMESTAMP, convert it to character and back to date:

```
TO_DATE( TO_CHAR( a_timestamp, 'yyyy-MM-dd' ) )
```

CAST() converts one data type to another.

TO_CHAR() converts a date to a character string.

CURRENT_TIMESTAMP() returns the current date and time.

TRUNC

This scalar function truncates a number to a specific count of decimal places.

Syntax

```
TRUNC( decimalNumber [, places ] )
```

Parameters

- decimalNumber— Number to truncate.
- places— Count of decimal places to truncate to. When omitted, truncates all decimals and returns an integer. When negative converts digits to zero.

Return Type

Numeric, same data-type as decimalNumber argument.

Examples

`TRUNC(1234.567)` returns 1234.

`TRUNC(1234.567, 1)` returns 12345.6.

`TRUNC(1234.567, -2)` returns 1200.

ROUND() rounds the number up to a specified count of decimal places.

TUMBLE_AVG

This tumbling set function returns the average value (arithmetic mean) of a tumbling window set.

Syntax

```
TUMBLE_AVG( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be a positive integer.
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

Numeric, same data-type as numeric argument.

Remarks

Returns NULL if the group is empty.

Example

Return the average price of all the events that arrive within a 7 day interval:

```
TUMBLE_AVG(price, DAY, 7, trade_time) AS Avg_7_day_price
```

The above function is shorthand for this in-line window expression:

```
AVG(price) AS Avg_7_day_price OVER (
    ORDER BY trade_time RANGE INTERVAL '7' DAY PRECEDING SLIDE)
```

To determine the average price of the previous 7 days, not including the current, use a window instead of a TUMBLE_AVG(), like this:

```
AVG(price) AS Avg_prev_7_day_price OVER (
    ORDER BY trade_time
    RANGE BETWEEN INTERVAL '8' DAY PRECEDING
          AND INTERVAL '1' DAY PRECEDING
    SLIDE INTERVAL '7' DAY
    INITIALIZE TIMESTAMP '1963-02-18 00:00:00.000'
    REFERENCE OPERATOR)
```

AVG() returns the mean average for a set.

MOV_AVG() returns the average for a moving window set.

Tumbling Windows discusses tumbling window sets.

TUMBLE_COUNT

This tumbling set function returns the count of rows in a tumbling window set.

Syntax

```
TUMBLE_COUNT( *, window, size [,timestampColumn] )
```

Parameters

- window— Determines if size is the count of events in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be a positive integer.
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

INTEGER.

Remarks

Returns zero (0) if the set is empty.

When using an EVENT window, this function returns an integer less than or equal to the window value.

Rows that include NULLs are counted.

When used with a GROUP BY returns the count of rows in the group set.

Example

Return the count of all the events that arrive within an 8 hour interval:

```
TUMBLE_COUNT(*, HOUR, 8, trade_time) AS Total
```

The above function is shorthand for this in-line window expression:

```
COUNT(*) AS Total OVER (ORDER BY trade_time
    RANGE INTERVAL '8' HOUR PRECEDING SLIDE)
```

Which in turn is equivalent to the following after filling in all default values:

```
COUNT(*) AS Total OVER ( ORDER BY trade_time
    RANGE INTERVAL '8' HOUR PRECEDING
    SLIDE INTERVAL '8' HOUR
    REFERENCE OPERATOR)
```

Note: The eight-hour window begins when the first event arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE '2003-03-05 00:00:00.000'.

COUNT() returns the count of a view or set.

MOV_COUNT() returns the count of a tumbling window set.

TUMBLE_MAX

This tumbling set function returns the maximum value from a tumbling window set.

Syntax

```
TUMBLE_MAX( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be a positive integer.
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

Same data-type as expression argument.

Remarks

For Boolean, True is greater than False.

For String, 'z" is greater than 'A'.

Example

Return the maximum price of all the events that arrive within a 1 hour interval:

```
TUMBLE_MAX(price, HOUR, 1, trade_time) AS Max_price
```

The above function is shorthand for this in-line window expression:

```
MAX(price) AS Max_price OVER (ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING SLIDE)
```

Which in turn is equivalent to the following after filling in all default values:

```
MAX(price) AS Max_price OVER ( ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING
    SLIDE INTERVAL '1' HOUR
    REFERENCE OPERATOR)
```

Note: The one-hour window begins when the first event arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE TIMESTAMP '2003-03-05 00:00:00.000'.

Similarly, the function TUMBLE_MAX(price, EVENT, 5) is the shorthand for this complete window:

```
MAX(price) OVER ( ORDER BY trade_time
    EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT
    SLIDE 5
    REFERENCE OPERATOR)
```

MAX() returns the maximum value from a moving window set.

MOV_MAX() returns the maximum value from a tumbling window set.

MIN() returns the minimum value from a set.

GREATEST() returns the maximum value from a list.

Tumbling Windows discusses tumbling window sets.

TUMBLE_MIN

This tumbling set function returns the minimum value from a tumbling window set.

Syntax

```
TUMBLE_MIN( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be a positive integer.
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

Same data-type as expression argument.

Remarks

For Boolean, True is greater than False.

For String, 'z" is greater than 'A'.

Example

Return the minimum price of all the events that arrive within a 1 hour interval:

```
TUMBLE_MIN(price, HOUR, 1, trade_time) AS Min_price
```

The above function is shorthand for this in-line window expression:

```
MIN(price) AS Min_price OVER (ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING SLIDE)
```

Which in turn is equivalent to the following after filling in all default values:

```
MIN(price) AS Min_price OVER ( ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING
    SLIDE INTERVAL '1' HOUR
    REFERENCE OPERATOR)
```

Note: The one-hour window begins when the first event arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE TIMESTAMP '2003-03-05 00:00:00.000'.

Similarly, the function TUMBLE_MIN(price, EVENT, 5) is the shorthand for this complete window:

```
MIN(price) OVER ( ORDER BY trade_time
    EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT
    SLIDE 5
    REFERENCE OPERATOR)
```

MIN() returns the minimum value from a moving window set.

MOV_MIN() returns the minimum value from a tumbling window set.

MAX() returns the maximum value from a set.

LEAST() returns the smallest value from a list.

Tumbling Windows discusses tumbling window sets.

TUMBLE_SUM

This tumbling set function returns the sum of a tumbling window set of numeric values.

Syntax

```
MOV_SUM( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be a positive integer.
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the first event in the view as the basis.

Return Type

Numeric, same data-type as numeric argument. Returns NULL if the set is empty.

Example

This tumbling sum expression sums all the events that arrive within a 1 hour interval:

```
TUMBLE_SUM(price, HOUR, 1, trade_time) AS Total
```

The above function is shorthand for this in-line window expression:

```
SUM(price) AS Total OVER (ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING SLIDE)
```

Which in turn is equivalent to the following after filling in all default values:

```
SUM(price) AS Total OVER ( ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING
    SLIDE INTERVAL '1' HOUR
    REFERENCE OPERATOR)
```

Note: The one-hour window begins when the first event arrives in the view. To begin the window at the top of the hour instead, include INITIALIZE TIMESTAMP '2003-03-05 00:00:00.000'.

Similarly, the function TUMBLE_SUM(price, EVENT, 5) is the shorthand for this complete window:

```
SUM(price) OVER ( ORDER BY trade_time
    EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT
    SLIDE 5
    REFERENCE OPERATOR)
```

SUM() returns the sum of a view or set.

MOV_SUM() returns the sum of a moving window set.

Tumbling Windows discusses tumbling window sets.

TUMBLE_STD_DEVIATION

This tumbling set function returns sample standard deviation of a tumbling window set of numbers.

Syntax

```
TUMBLE_STD_DEVIATION( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT), or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be a positive integer.
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

DOUBLE PRECISION.

Remarks

Returns 0 when there is only a single row of input. Returns NULL if the set is empty.

The result is computed using the formula $\sqrt{\sum\left[\frac{(X_i - X)^2}{n}\right]}$, where *n* is the number of elements in the sample and *X* is the sample mean.

Example

Return the standard deviation in salaries for each calendar year:

```
SELECT TUMBLE_STD_DEVIATION(salary, YEAR, 1) AS "Deviation per year"
    FROM employees;
```

Which in turn is equivalent to the following after filling in all default values:

```
SELECT STD_DEVIATION(salary) AS "Deviation per year" OVER (
    RANGE INTERVAL '1' YEAR PRECEDING
    SLIDE INTERVAL '1' YEAR
    REFERENCE OPERATOR)
```

Note that you can use INITIALIZE to declare a fiscal year. And consider using PARTITION BY to get the deviations for different pay grades. For example:

```
SELECT STD_DEVIATION(salary) AS "Deviation per year" OVER (
    PARTITION BY pay_grade
    RANGE INTERVAL '1' YEAR PRECEDING
    SLIDE INTERVAL '1' YEAR
    INITIALIZE '1963-07-01 00:00:00.000'
    REFERENCE OPERATOR)
```

STD_DEVIATION() returns the standard deviation of a view or set.

MOV_STD_DEVIATION() returns the standard deviation of a tumbling window set.

VARIANCE() returns the square of the standard deviation.

Tumbling Windows discusses tumbling window sets.

TUMBLE_VARIANCE

This tumbling set function returns the square of the sample standard deviation of a tumbling window set of numbers.

Syntax

```
TUMBLE_STD_DEVIATION( numeric, window, size [,timestampColumn] )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view. See individual function descriptions for additional restrictions.
- window— Determines if size is the count of events in the set (EVENT) or a duration of time (SECOND, MINUTE, HOUR, DAY. MONTH, or YEAR).
- size— Duration or count of window to use for determining the size of the set. Must be an integer greater than zero (0).
- timestampColumn— (optional) Use the value of the field as the starting point for the time-series span. The calculation is based on the time span from the *most recent value in the column from any row in the view*. Omit this option to use the system clock as the time basis.

Return Type

DOUBLE PRECISION.

Remarks

Returns zero (0) when the expression set contains only one element. Returns NULL if the set is empty.

The result is computed using the formula $\sum\left[\frac{(X_i - X)^2}{n}\right]$, where *n* is the number of elements in the sample and *X* is the sample mean.

Examples

Return the variation in salaries for each calendar year:

```
SELECT TUMBLE_STD_VARIATION(salary, YEAR, 1)
        AS "Variation for last year"
    FROM employees;
```

Which in turn is equivalent to the following after filling in all default values:

```
SELECT VARIANCE(price) AS "Variation for last year"
    OVER ( ORDER BY trade_time
        RANGE INTERVAL '1' YEAR PRECEDING
        SLIDE INTERVAL '1' YEAR
        REFERENCE OPERATOR)
    FROM employees;
```

MOV_VARIANCE() returns the variance of a moving window set.

STD_DEVIATION() returns a standard deviation.

VARIANCE() returns the variance of a view or set.

“Function Types” on page 109 discusses moving sets.

UPPER

This scalar function converts all lowercase characters in a string to uppercase.

Syntax

```
UPPER( string )
```

Parameters

- string— Character string (VARCHAR) to convert.

Return Type

VARCHAR.

Example

`UPPER('Volta')` returns ‘VOLTA’.

LOWER() converts to all lowercase.

VARIANCE

This set function returns the square of the sample standard deviation of a set of numbers.

Syntax

```
VARIANCE( numeric )
```

Parameters

- numeric— An expression that evaluates to a numeric and which cannot reference a rank function function. Typically the argument is a column in a view.

Return Type

DOUBLE PRECISION.

Remarks

Returns zero (0) when the expression set contains only one element. Returns NULL if the set is empty.

The result is computed using the formula $\sum\left[\frac{(X_i - X)^2}{n}\right]$, where *n* is the number of elements in the sample and *X* is the sample mean.

Example

```
SELECT VARIANCE(salary) "Variance"
    FROM employees;

Variance
----------
15283140.5
```

MOV_VARIANCE() returns the variance of a moving window set.

STD_DEVIATION() returns a standard deviation.

TUMBLE_VARIANCE() returns the variance of a tumbling window set.

yield

This function computes the yield given a set of test results (fail=0, pass=1) and calculates the yield; that is, total passed/total units. Null values are ignored.

Syntax

```
yield(pass, unit)
```

Parameters

- pass— Indicates whether to calculate the yield based on the number of units that pass or the number of units that failed. Specify pass to use the number of units that pass the test to calculate the yield. Specify fail to use the number of units that fail the test to calculate the yield.
- unit— Specifies the total number of units to test.

Return Type

Returns NULL if the yield is less than zero; otherwise, DOUBLE PRECISION.

Example

If SELECT YEILD(pass, 4) FROM E is passed the following values:

0, 1
1, 0
0, 1
NULL, NULL

The result is 0.5 because test 1 and 3 pass but test 2 fails. Test 4 is ignored.

If SELECT YEILD(fail, 4) FROM E is passed the same values, the result is 0.25 because only one of the four tests passes.

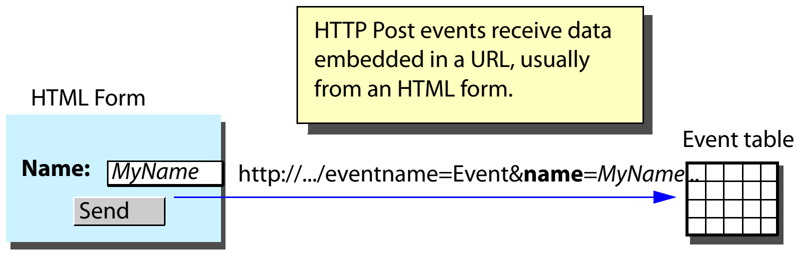
14 HTTP Post

A HTTP Post event tables receive events from a HTTP Post action, either as the result of an HTML form sent from a browser or from data encoded in a URL that connects to the table.

➤ **In this Chapter:**

How It Works

HTTP Post event data arrives embedded in an URL. The internal agent extracts the fields from the URL and puts the data in the event table. The URL may be formed as the result of an HTML form containing <INPUT> fields, or it may be created by some other application that communicates in the HTTP protocol.


HTTP Post events receive data embedded in a URL, usually from an HTML form.
HTML Form
Name:
MyName
Send
http://.../eventname=Event&name=MyName&...
Event table

HTTP Post Event Tables

A HTTP Post event table receives new events from an HTTP Post action, which is usually the result of an HTML form sent from a browser. In an HTML form, each <INPUT> element maps to a column in the event table. Event data can also be published in the URL that passes the fields to the system. See "Posting to an HTTP post event" on page 201 for examples.

Before creating an event to a HTTP Post, you must have Create permission for tables (see "Creating Permission" on page 258). The following table lists the attributes for an HTTP Post event table.

Attribute	**Description**
Name	Event table name. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the table is enabled (monitoring for events) or disabled (not monitoring for events).
Post To URL	URL in which to send the posted information. See "Posting to an HTTP post event" on page 201 for examples.
Log event data for recovery	When on, logs event data that arrived after the last checkpoint started. This "recovery" log is used to restore the state of the system in the event of an abnormal shutdown of the servers. For complete details, see "Working with Checkpoint and Recovery" in *Using Business Activity Monitoring Workbench*.
Process events in the order of arrival	Choose this option when events must be processed in the order received. Otherwise, if events may be processed out of order, turn this option on. **Note:** To join events in a view, the events must be processed in order. Leave this option off to join the events.

Attribute	**Description**
Disable event after this number of consecutive errors	Disables the event when a consecutive count of errors occur. For example, if set to 5, the event is disabled after 5 consecutive errors. However, if 4 errors occur, then no errors are followed by 2 more errors, the event remains enabled. The default is *off*: Do not disable.
Column Information	The Column Information fields define how to map the fields from the JMS message into columns in the event table. There is one column for every field in the event table. See "HTTP Post Column Information" on page 200 for details.
Clear State Interval	This tab contains several options for clearing persisted event data that is propagated from the event in the views, objects, and dashboards that depend on it. The three options are: • Do Not Clear State— This is the default. Data persists. • Clear State on a Schedule— Select to clear the state on a schedule. Selecting this option activates the scheduling feature positioned to the right. • Always Clear State (Every Event)— The state is refreshed each time the event is updated.

Creating an HTTP Post event table

Use the following procedure to create an HTTP post event table.

➤ **To create a HTTP Post event table:**

1. Open the BAM Workbench.
2. Click the New Event... button
3. On the New Event dialog page, select Single Event.
4. Select HTTP Post as the event source type.
5. Assign a name and define the columns of the event table in the Column Information fields.

Save the HTTP Post table as enabled and it will immediately be ready to receive event messages.

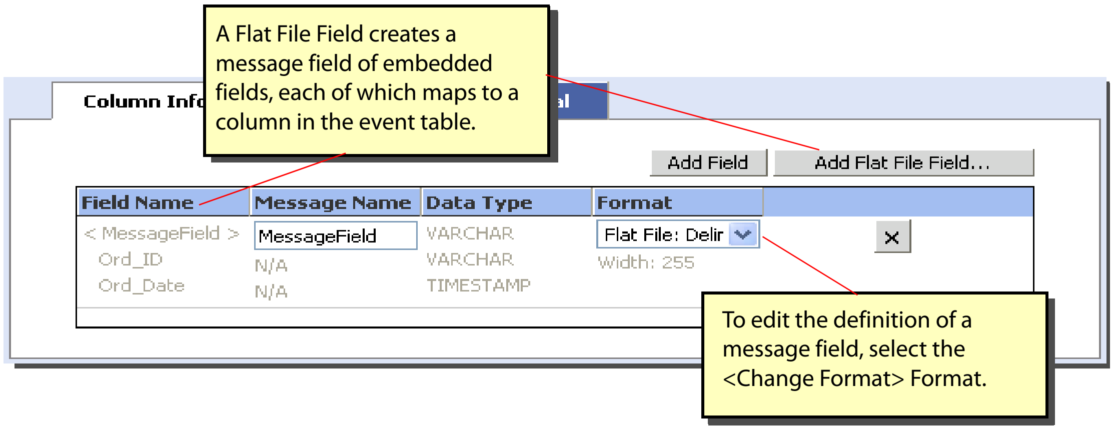
HTTP Post Column Information

The Column Information fields define how to map the fields from the HTTP Post message into columns in the event table. There is one field for every column in the event table, each with the following attributes:

Attribute	Description
Field Name	Name of the column in the event table.
Message Name	Name of the field in the message. On an HTML form, this is the NAME attribute assigned to each form element. See "Posting to an HTTP post event" on page 201 for examples. When mapping a Flat File field, the name for each embedded field is N/A and uneditable.
Data Type	Data type of the column in the event table.
Format	(optional) Format of the event column for VARCHAR (string) and DECIMAL values.

Each field in the message can be a simple field that maps directly into a event column, or it can be a complex field (a *flat file field*) that contains several fields that each map into columns in the table. Complex fields are treated as Flat Files in either delimited (CSV), fixed-width, or XML formats. See "Flat Files" on page 81 for detailed descriptions of these file types.

Note: Message fields can contain more than one row of data; however, each row is part of the same event. This is different from flat file imports that treat each row in the file as a unique event.


Column Info
Add Field
Add Flat File Field...
A Flat File Field creates a message field of embedded fields, each of which maps to a column in the event table.
To edit the definition of a message field, select the <Change Format> Format.

Field Name	Message Name	Data Type	Format
< MessageField >	MessageField	VARCHAR	Flat File: Delir
Ord_ID	N/A	VARCHAR	Width: 255
Ord_Date	N/A	TIMESTAMP	

➤ To add a message field:

1. Click Add Flat File Field.
2. Choose the flat file type of the message field.

   (Optional for fixed-width and delimited files.) Identify a sample file to assist in mapping the columns. This file is a sample of the real data file. Data from this file appears in the next step to assist you as you map the event data into the table.
3. For fixed-width files, define the positions of the data columns with the Set Field Widths dialog.
4. Identify the flat-file attributes. See "Flat File Event Tables" on page 82 for details.

5. Define the format-specific Column Information. For details about the source type, see:
   - "Fixed-Width Files" on page 94.
   - "Delimited Files" on page 93
   - "XML Files" on page 95

   See "Multiple Lines (Events) of Input" on page 203 for details about sending data to flat file fields.
6. Click Save Event to save the message field definition.

➤ **To edit the definition of a message field:**

- In the field's Format column, change the value from "Flat File: *file type*" to "<Change Formatting>".

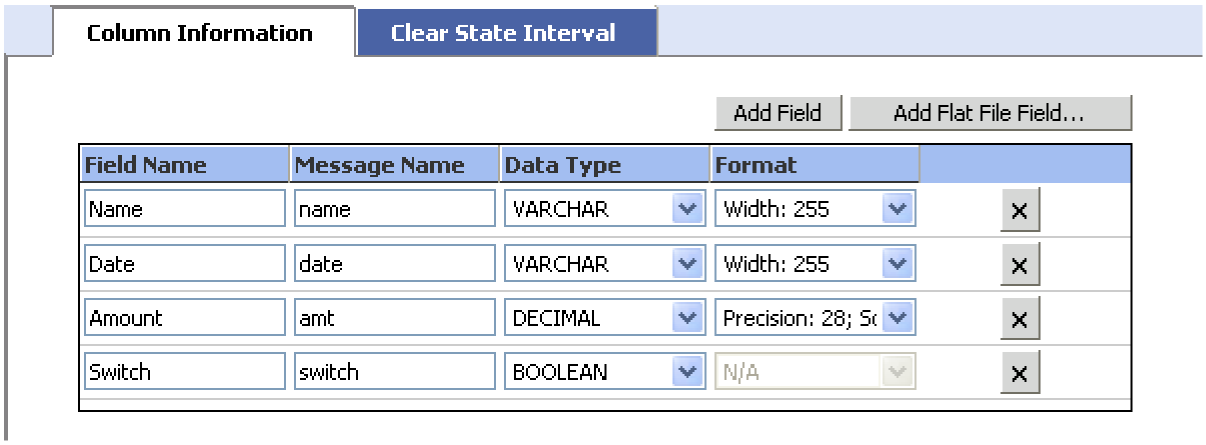
  Note: When editing a message field, the sample file option for delimited and fixed-width file types is not available.

Posting to an HTTP post event

Most HTTP Post event are generated from an HTML form. When defining the event, define one column for each named <INPUT> element. For example, consider this HTML form:

```
<FORM action="http://.../bam/postservlet?eventname=Example"
      method="post">
<P>
    <LABEL for="name">Name: </LABEL>
        <INPUT type="text" name="name"><BR>

    <LABEL for="name">Date (yyyy-mm-dd): </LABEL>
        <INPUT type="text" name="date"><BR>

    <LABEL for="amt">Amount: </LABEL>
        <INPUT type="text" name="amt"><BR>

    <LABEL for="switch">Switch: </LABEL>
        <INPUT type="radio" name="switch" value="FALSE" checked >Off
        <INPUT type="radio" name="switch" value="TRUE">On<BR>

    <INPUT type="submit" value="Send">
    <INPUT type="reset">
    </P>
</FORM>
```

The four form fields map to these four columns in the event table:



Field Name	Message Name	Data Type	Format	
Name	name	VARCHAR	Width: 255	×
Date	date	VARCHAR	Width: 255	×
Amount	amt	DECIMAL	Precision: 28; Sc	×
Switch	switch	BOOLEAN	N/A	×

Notice that the date field maps to a VARCHAR, not a TIMESTAMP. In the views that are derived from this event, *cast* the values to a date-time, similar to the following:

```
CAST(httpEvent."Date" AS TIMESTAMP)
```

Posting to Message Fields

To pass data into a message field, you can either embed the information in the URL (as described below in Posting Values in the URL") or use an HTML <TEXTAREA> element and enter the flat-file data into that field. For example, your HTML form might have the following declaration:

```
<LABEL for="flatfile">Flat file text: </LABEL>
    <TEXTAREA name="MessageField" rows="20" cols="80"></TEXTAREA>
```

In the browser, you can either enter the data manually or copy the data from a flat file and manually past it into the field. Remember that the data must be in the format of the declared Flat File Field, such as delimited.

Note the following:

- When the event table contains more than just the Message Field column, you can enter only one row's worth of data into the <TEXTAREA> field. If the Message Field is the only column, then you may enter multiple event row's worth of data.
- Multiple rows passed through a <TEXTAREA> element are considered part of the same event, unlike a text file where each row is a unique event.

Posting Values in the URL

When passing the field values directly in the URL, name and assign a value to each, separating them with ampersands (&); for example,

```
...?eventname=Example&name="Skyler"&date=2003-03-05&amt=9.21&switch=TRUE
```

However, if passing the values to a delimited flat file field, just name the field and separate the values with the separator character (which is usually a coma); for example,

```
...?eventname=Example&msgFile="Skyler",2003-03-05,9.21,TRUE
```

Multiple Lines (Events) of Input

To send multiple events to a flat file field, separate them with the %0D%0A (the MIME transmission for an end-of-line: "CR LF"); for example,

```
...="Skyler",2003-03-05,9.21,TRUE%0D%0A"Mike",1963-02-18,9.01,FALSE
```

Note that if you intend to send multiple lines, the Flat File field must be the only field in event column list. When the list includes other columns, only one "line" of input is permitted.

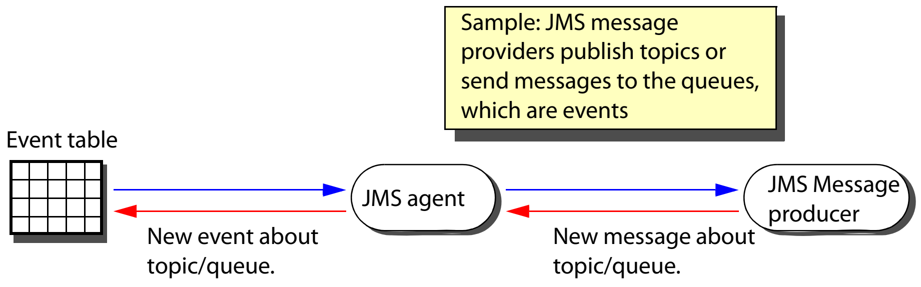
15 Java Messaging Service (JMS)

Java Messaging Service (JMS) provides access to messages produced by Java applications. The producer application publishes messages to *topics* or *queue* destinations that the Business Activity Monitoring agent *subscribes* to. Each new published topic or queue message is mapped to a new event in the associated event table.

➤ **In this Chapter:**

How It works

Business Activity Monitoring JMS agents communicate with JMS topic factories or Queue factories managed by Web application servers. When you define the agent, you tell it how to connect to the factory. When you define a JMS event table, you tell it to subscribe to a JMS topic or queue managed by the factory that the agent talks to. Then, when the topic publishes a new message, or when the queue receives a new message, the agent receives it and passes it to the event table.


Sample: JMS message providers publish topics or send messages to the queues, which are events
Event table
JMS agent
JMS Message producer
New event about topic/queue.
New message about topic/queue.

Note: Business Activity Monitoring JMS agents support JMS *MapMessage* **and** *TextMessage* body types for both topic and queue messages. This JMS Map Message type consists of name-value pairs, where the names are strings and the values are wrappers to Java types. See "Mapping JMS Data Types" on page 209 for details.

JMS Event

A Java Messaging Service (JMS) event receives new event data from a Java application that publishes messages to the topic or sends messages to the queue that the table subscribes to. Each new topic or queue message is a new event in the table.

Limitations

The JMS agent supports JMS *MapMessage* and *TextMessage* body types only. MapMessage consists of name-value pairs, where the names are strings and the values are wrappers to Java types. TextMessage consists of a single unnamed text string, which can be interpreted as a fixed-width or delimited file or an XML file. See "Mapping JMS Data Types" on page 209 for details.

Prerequisites

Before creating an event to a JMS agent, you must:

- Have create permission for Contexts and Events (see "Creating Permission" on page 258), and Read Only access permission on the agent that will feed the table.
- Obtain access to a JMS Topic or Queue as identified by the address factory's JNDI location.
- A sample file (optional).

  If the message contains a complex string that is CSV (delimited) or fixed-width text, it is helpful to have a sample file that contains data in the format of the actual event string. You can use this sample when you create the event to ensure that the fields map correctly into the event table by seeing how the data lines up in the columns.

The following table describes the attributes for a JMS event.

Attribute	Description
Name	Event table name. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the table is enabled (monitoring for events) or disabled (not monitoring for events).
Log event data for recovery	When on, logs event data that arrived after the last checkpoint started. This "recovery" log is used to restore the state of the system in the event of an abnormal shutdown of the servers. See Working with Checkpoint and Recovery for complete details.
Process events in the order of arrival	Choose this option when events must be processed in the order received. Otherwise, if events may be processed out of order, turn this on. **Note:** To join events in a view, the events must be processed in order. Leave this option off to join the events.
JMS Agent	An existing JMS agent that connects to the JMS message stream. Create an JMS Queue or JMS Topic agent with the BAM Workbench Administration tab. See "JMS Queue Agents" on page 210 or "JMS Topic Agents" on page 213 for details.
Topic/Queue	Identifies the *topic/queue* on which the message is being sent and defined by the message publisher. This is a JNDI address similar to com.celequest.myjmstopic on BEA WebLogic and topic/com.celequest.myjmstopic on JBoss.
Message selector	A Boolean expression that puts a filter condition on the messages the publisher sends. The syntax of the condition is the same as that of the SELECT command's WHERE Clause. For example, this filter only accepts messages where the Supplier property contains one of three values: Supplier IN ('Xyz, Corp', 'Ink, Inc', 'Gizmos')

Attribute	Description
Column Information	The Column Information fields define how to map the fields from the JMS message into columns in the event table. There is one column for every field in the event table. See "JMS Column Information" on page 208 for details.
Clear State Interval	This tab contains several options for clearing persisted event data that is propagated from the event in the views, objects, and dashboards that depend on it. The three options are: • Do Not Clear State— This is the default. Data persists. • Clear State on a Schedule— Select to clear the state on a schedule. Selecting this option activates the scheduling feature positioned to the right. • Always Clear State (Every Event)— The state is refreshed each time the event is updated.

Creating a JMS Event Table

Use the following procedure to create a JMS event table.

➤ To create a JMS event table:

1. Open the BAM Workbench Administration Console.
2. Create a new event.
3. Select JMS as the source type.
4. Define the values of the event table's attributes.
5. Define the columns of the event table in the Column Information fields.

Save the JMS table as enabled and it will immediately be ready to receive event messages.

JMS Column Information

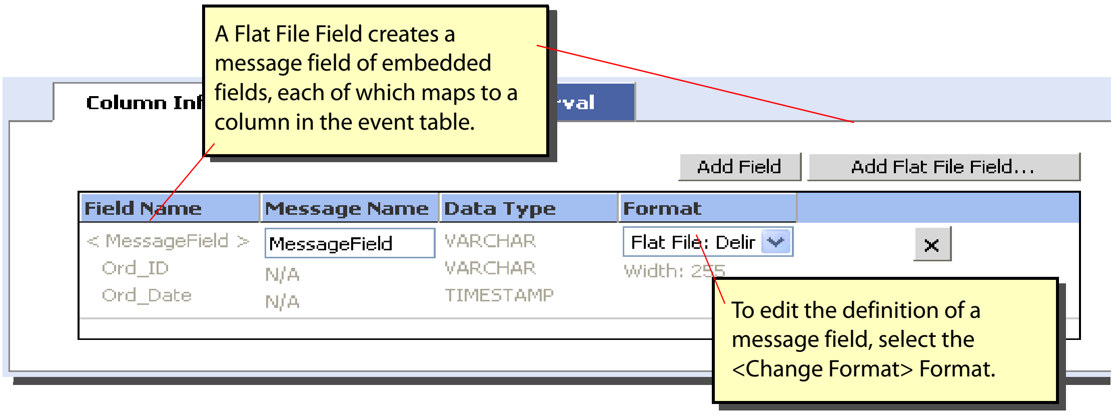
The Column Information fields define how to map the fields from the JMS message into columns in the event table. There is one field for every column in the event table.

Each field in the message can be a simple field that maps directly into a event column, or it can be a complex field (a *flat file field*) that contains several fields that each map into columns in the table. Complex fields are treated as Flat Files in either delimited (CSV), fixed-width, or XML formats. See “Flat Files” on page 81 for detailed descriptions of these file types.

Each column in the event table has the following attributes:

Attribute	Description
Field Name	Name of the column in the event table.
Message Name	Name of the field in the message. When mapping a MessageField, the name for each embedded field is N/A and uneditable.
Data Type	Data type of the event column. See “Mapping JMS Data Types” on page 209 for details.
Format	(optional) Format of the event column for VACHAR (string) and DECIMAL values.

Add columns by clicking Add Field or Add Flat File Field.


A Flat File Field creates a message field of embedded fields, each of which maps to a column in the event table.
Column Inf
val
Add Field
Add Flat File Field...
To edit the definition of a message field, select the <Change Format> Format.

Field Name	Message Name	Data Type	Format
< MessageField >	MessageField	VARCHAR	Flat File: Delir
Ord_ID	N/A	VARCHAR	Width: 255
Ord_Date	N/A	TIMESTAMP	

➤ To add a message field:

1. Click Add Flat File Field.
2. Choose the flat file type of the message field.

   (Optional for fixed-width and delimited files.) Identify a sample file to assist in mapping the columns. This file is a sample of the real data file. Data from this file appears in the next step to assist you as you map the event data into the table.
3. For fixed-width files, define the positions of the data columns with the Set Field Widths dialog.
4. Identify the flat-file attributes. See "Flat File Event Tables" on page 82 for details.
5. Define the format-specific Column Information. For details about the source type, see:
   - "Fixed-Width Files" on page 94.
   - "Delimited Files" on page 93
   - "XML Files" on page 95
6. Click Save Event to save the message field definition.

➤ To edit the definition of a message field:

- In the field's Format column, change the value from "Flat File: *file type*" to "<Change Formatting>".

  Note: When editing a message field, the sample file option for delimited and fixed-width file types is not available.

Mapping JMS Data Types

The JMS mapped message data types map to Business Activity Monitoring Data Types as follows.

Java Data Type	Business Activity Monitoring Data Type	Java Data Type	Business Activity Monitoring Data Type
boolean	Boolean	long	Decimal
byte	Integer	float	Double
short	Integer	double	Double
char	Varchar	String	Varchar
int	Integer	byte[]	Not supported

JMS Queue Agents

A Java Messaging Service (JMS) Queue agent communicates with a JMS message producer through a JMS queue running in the application server environment. The agent tells the producer which messages the JMS Queue Agents event table is interested in receiving. The producer then sends messages to the event table through the agent.

Note: JMS Queue agents are asynchronous, they receive event messages as the events occur, and you *cannot* retrieve context from a JMS Queue agent as summarized in the following table.

Event Push	Event Pull	Context Pull
Yes	No	No

Before creating an agent, you need:

- A Custom Message-driven bean (MDB), which has the Event name hard coded in the properties file and that will receive messages from the Queue, must be deployed in the host application server.

  Note: Application servers do not allow dynamic subscriptions to JMS Queues. A custom MDB must be created for this purpose. For assistance creating and configuring this MDB, refer to the ReadMe files under samples/cqmdb folder from the distribution CD.

- Create permission for agents (see "Creating Permission" on page 258 for details).
- The JNDI location of the topic factory in the application server that is receiving the queue messages.

A JMS Queue agent has the following attributes:

Attribute	Required	Description
Name	Yes	Identifies the agent. This name must be unique among agents. See "Object Namespace" on page 248 for details.
Description	No	Optional description that may contain any text characters.
Status	Yes	Whether or not the agent is enabled (monitoring for events) or disabled (not monitoring for events).
Queue Connection Factory	Yes	Identifies the J2EE connection factory that maintains the desired queue. This string identifies the factory by its JNDI lookup name. For example: jms.ManufacturingQueue In a BEA WebLogic environment, the factory JNDI name is identified on the BEA WebLogic Console, Services > JMS > Connection Factories > *factoryName* > General tab.

Attribute	Required	Description
Acknowledge mode (Auto)	No	Protocol to use when acknowledging receipt of the message. • AUTO — (default) Provider acknowledges message when it is delivered. • CLIENT — Acknowledges the message when the agent receives it. • DUPS OK — Tells the publisher that it is OK to send a message more than once. Note that subsequent receipts of the same message are treated as new and unique events.
User name	No	User name to use to connect to the JMS factory.
Password	No	Password for the User name.
JNDI properties	No	Optional and additional Java Naming and Directory Interface (JNDI) properties necessary to make or maintain the agent to the JMS table. These name-value pairs allow you to specify JMS properties recognized by the JNDI. **Note:** When the JMS Queue is running in a different namespace from the BAM Server, define the properties described in "JNDI Properties For Connecting to a Remote Namespace" on page 211.

JNDI Properties For Connecting to a Remote Namespace

When the JMS Queue is running in a different namespace from the BAM Server, define these JNDI properties to make the connection:

```
java.naming.factory.initial
java.naming.provider.url
```

Further, if you are using security, also define these properties:

```
java.naming.security.authentication
java.naming.security.principal
java.naming.security.credentials
```

Examples

BEA WebLogic JNDI

```
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.provider.url=t3://localhost:9180
```

IBM Websphere

```
java.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFact
ory
java.naming.provider.url=iiop://localhost:9180
```

JBoss

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
```

Sun One Directory Server LDAP

```
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
java.naming.provider.url=ldap://russell:59226/dc=viewceler,dc=com
```

Also, prefix the Queue Connection Factory parameter value with: cn=

Creating a JMS Queue Agent

Use the following procedure to create a JMS Queue agent.

➤ **To create a JMS Queue agent:**

1. Open the Administration Console.
2. Click New Agent…
3. Choose JMS Queue as the source type
4. Fill in the required fields that define the agent's attributes.

Save the agent as enabled and it will immediately begin monitoring for events.

JMS Topic Agents

A Java Messaging Service (JMS) Topic agent communicates with a JMS message producer through a JMS topic running in the application server environment. The agent tells the publisher which messages the JMS Topic Agents event table is interested in subscribing to. The publisher then sends messages to the event table through the agent.

Note: JMS Topic agents are asynchronous, they receive event messages as the events occur, and you *cannot* retrieve context from a JMS Topic agent as summarized in the following table.

Event Push	Event Pull	Context Pull
Yes	No	No

Before creating an agent, you need:

- A Custom Message-driven bean (MDB), which has the Event name hard coded and that will subscribe to the Topic, must be deployed in the host application server.

  Note: Application servers do not allow dynamic subscriptions to JMS Topics. A custom MDB must be created for this purpose. For assistance creating and configuring this MDB, refer to the ReadMe files under samples/cqmdb folder from the distribution CD.
- Create permission for agents (see "Creating Permission" on page 258 for details).
- The JNDI location of the topic factory in the application server that is publishing the topics.

A JMS agent has the following attributes:

Attribute	Required	Description
Name	Yes	Identifies the agent. This name must be unique among agents. See "Object Namespace" on page 248 for details.
Description	No	Optional description that may contain any text characters.
Status	Yes	Whether or not the agent is enabled (monitoring for events) or disabled (not monitoring for events).
Topic Connection Factory	Yes	Identifies the J2EE connection factory that maintains the desired topics. This string identifies the factory by its JNDI lookup name. For example: jms.ManufacturingTopic In a BEA WebLogic environment, the factory JNDI name is identified on the BEA WebLogic Console, Services > JMS > Connection Factories > *factoryName* > General tab.

Attribute	Required	Description
Acknowledge mode (Auto)	No	Protocol to use when acknowledging receipt of the message. • AUTO — (default) Provider acknowledges message when it is delivered. • CLIENT — Acknowledges the message when the agent receives it. • DUPS OK — Tells the publisher that it is OK to send a message more than once. Note that subsequent receipts of the same message are treated as new and unique events.
User name	No	User name to use to connect to the JMS factory.
Password	No	Password for the user name.
JNDI properties	No	Optional and additional Java naming and directory interface (JNDI) properties necessary to make or maintain the agent to the JMS table. These name-value pairs allow you to specify JMS properties recognized by the JNDI. **Note:** When the JMS topic is running in a different namespace from the BAM Server, define the properties described in "JNDI Properties For Connecting to a Remote Namespace" on page 211.

JNDI Properties For Connecting to a Remote Namespace

When the JMS topic is running in a different namespace from the BAM Server, define these JNDI properties to make the connection:

```
java.naming.factory.initial
java.naming.provider.url
```

Further, if you are using security, also define these properties:

```
java.naming.security.authentication
java.naming.security.principal
java.naming.security.credentials
```

Examples

BEA WebLogic JNDI

```
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.provider.url=t3://localhost:9180
```

IBM Websphere

```
java.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFact
ory
java.naming.provider.url=iiop://localhost:9180
```

JBoss

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
```

Sun One Directory Server LDAP

```
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory
java.naming.provider.url=ldap://russell:59226/dc=viewceler,dc=com
```

Also, prefix the Topic Connection Factory parameter value with: cn=

Creating a JMS Topic Agent

Use the following procedure to create a JMS Topic agent.

➤ **To create a JMS Topic agent:**

1. Open the Administration Console.
2. Click New Agent…
3. Choose JMS Topic as the source type
4. Fill in the required fields that define the agent's attributes.

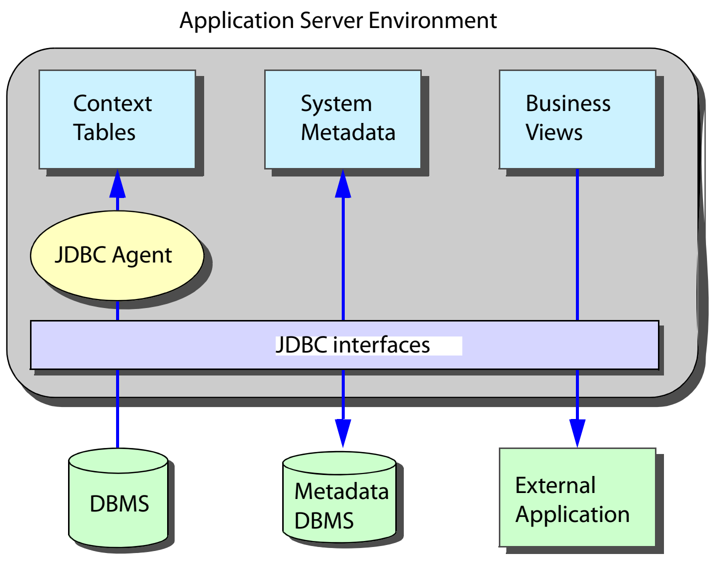
Save the agent as enabled and it will immediately begin monitoring for events.

Message Driven Beans (MDB)

16 JDBC

JDBC (Java Database Connectivity) is a Java application programming interface for accessing standard SQL databases from Java programs. Business Activity Monitoring (a Java application) uses JDBC to:

- Retrieve context data from a relational database (DBMS), as described in "JDBC Tables" on page 217 and "JDBC Agents" on page 224.
- Access the metadata database that Business Activity Monitoring uses to store object and state definitions.
- Allow other Java applications to access the business view data in memory. See "JDBC Access to View Data" on page 228 for details.



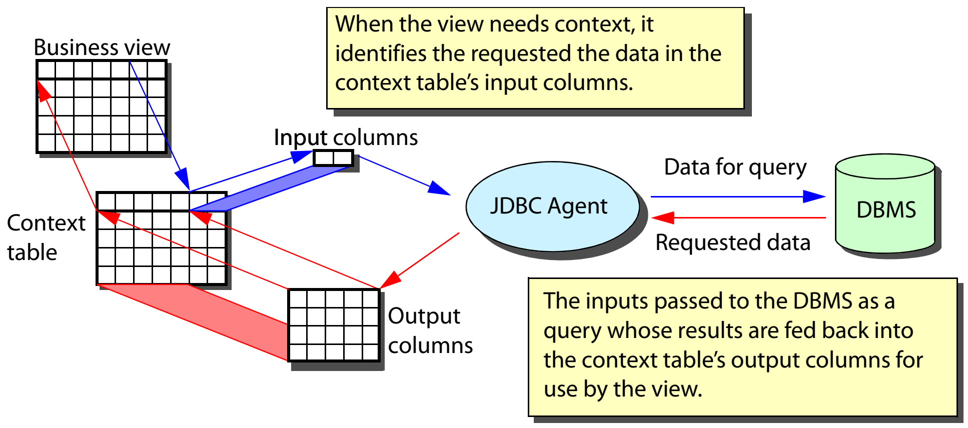
➤ **In This Chapter:**

JDBC Tables

Java database connectivity (JDBC) event and context tables receive their data from external relational database systems (DBMS). The data are retrieved by either making a query on the database or by calling a stored procedure in the DBMS.

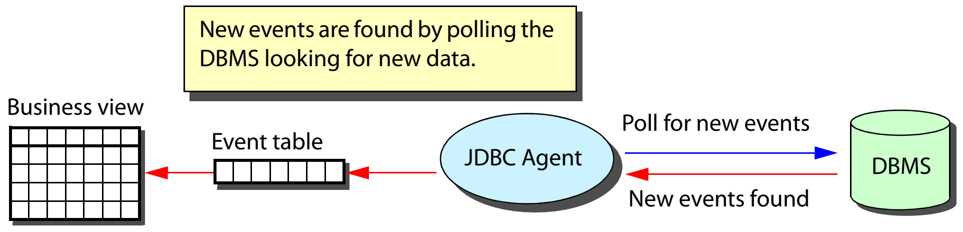
Context Tables

For Context tables, new data is retrieved only when a new event requires it. Then the agent passes the query data to the DBMS, which then returns the result from DBMS.


When the view needs context, it identifies the requested the data in the context table's input columns.
Business view
Input columns
Data for query
JDBC Agent
DBMS
Context table
Requested data
Output columns
The inputs passed to the DBMS as a query whose results are fed back into the context table's output columns for use by the view.

Event Tables

For event tables, the agent periodically polls the DBMS to see if new events are available, then retrieves them for inclusion in the events table. Each event returned is processed individually, regardless of the count of events returned as a result of the polling query.



Before creating an event to a JDBC source table, you need:

- An agent — An existing JDBC agent defined with sufficient access rights to query the database, or call the stored procedure. Create an agent with the BAM Workbench Administration tab. See "JDBC Agents" on page 224 for details.
- For queries — The schemas of the tables to query.
- For query events — A column in the source table must be an incrementing value that identifies when new events are available. See "Polling the JDBC Source" on page 221.
- For stored procedures — To define the procedure in the RDBMS and provide a list of the input and (result set) output fields, and their data types. See "Stored Procedure Source" on page 222," for details.
- Permissions — Create permission for tables (see "Creating Permission" on page 258) and Read Only access permission on the agent that will feed the table.

A JDBC tables has the following attributes:

Attributes	**Description**
Name	Identifies the table. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the event object is enabled (monitoring for events) or disabled (not monitoring for events).
Log event data for recovery	When on, logs event data that arrived after the last checkpoint started. This "recovery" log is used to restore the state of the system in the event of an abnormal shutdown of the BAM Server(s). For complete details, see "Working with Checkpoint and Recovery" in *Using Business Activity Monitoring Workbench*.
Process events in the order of arrival	Choose this option when events must be processed in the order received. Otherwise, if events may be processed out of order, turn this on. **Note:** To join events in a view, the events must be processed in order. Leave this option off to join the events.
JDBC Agent	An existing JDBC source agent that accesses an RDBMS. Create an agent with the BAM Workbench Administration tab. See "JDBC Agents" on page 224 for details.
JDBC Query	SELECT statement made against the database *in the native database language*. For details about the SELECT command, see the reference documentation for the DBMS. If you change the query, click Resubmit Query to validate it; you cannot save this form with an invalid query.
Disable context after errors	Count of consecutive errors to receive before the system disables this context. Once disabled, a context must be re-enabled manually.
Treat all rows in ResultSet as single event	All rows returned in the ResultSet are considered a single event. Otherwise, every row returned from the table is considered a separate event.
Field Information	Columns to populate in the event or context table. The Field Names are derived from the JDBC Query result. When the system validates the query, it populates this field list and identifies the JDBC Data Type of each return value. You specify the associated Data Type of the column in the table. The field names are the same as defined in the DBMS schema unless you alias them with the AS operator in the SELECT statement's select list.
Event Key (event only)	For fixed-width and delimited files identifies key field columns for multi-row events. See "Multi-Row Events" on page 84 for details.

Attributes	Description
Index (context only)	Builds an index for the column when checked. It is critical that you select the right index in order to have good performance when prefetch is enabled for data caching. Select index for those columns you will be using in join conditions. **Note:** If you are not using prefetch caching, selecting index has no effect.
Caching (context only)	Stores query results in memory and future requests retrieve data from memory, thereby lessening the impact on the DBMS by reducing the number of queries. See "Caching Context Queries" on page 48 for details about this feature.
Polling (event only)	How frequently to call the stored procedure or to query the DBMS for new events. See "Polling the JDBC Source" on page 221 for details.
Clear State Interval (event only)	This tab contains several options for clearing persisted event data that is propagated from the event in the views, objects, and dashboards that depend on it. The three options are: • Do Not Clear State—This is the default. Data persists. • Clear State on a Schedule— Select to clear the state on a schedule. Selecting this option activates the scheduling feature positioned to the right. • Always Clear State (Every Event)— The state is refreshed each time the event is updated.

Creating a JDBC Source Event or Context Table

Use the following procedures to create JDBC source event or context table.

➤ To create a JDBC source event table:

1. Open the BAM Workbench tab.
2. From Tables and Views, select **Events** then click **New Event.** (See "JDBC Tables" on page 217.)
3. Select either Single Event
4. Choose JDBC as the event source type.
5. Choose either Query source or Stored Procedure source.
   - Stored Procedure source calls a stored procedure in the DBMS to locate the data. See "Stored Procedure Source" on page 222," for details about this source.
   - Query source makes a SELECT SQL query on the database *in the native database language*. Enter the SELECT statement in the query field. See Query Source," below, for details about this source. When you click Continue, the BAM Workbench issues the query to validate the query and determine the return columns.
6. Save the JDBC source as enabled.

   The source will immediately be ready to receive events or context.

➤ **To create a JDBC source context table:**

1. Open the BAM Workbench tab.
2. From Tables and Views, select **Contexts** then click **New Context**.
3. Choose JDBC as the context source type. (See "Context Tables" on page 217.)
4. Select the JDBC connection
5. Choose either Query source or Stored Procedure source.
   - Stored Procedure source calls a stored procedure in the DBMS to locate the data. See "Stored Procedure Source" on page 222," for details about this source.
   - Query source makes a SELECT SQL query on the database *in the native database language*. Enter the SELECT statement in the query field. See Query Source," below, for details about this source. When you click Continue, the BAM Workbench issues the query to validate the query and determine the return columns.
6. Save the JDBC source as enabled.

   The source will immediately be ready to receive events or context.

Query Source

A query source makes a SELECT query on the database in the source DBMS. Queries are used for both context and events, and are in native syntax used by the DBMS. For specific syntax information, see the DBMS documentation.

Sybase Limitations

When making a query to a Sybase database, be aware of these limitations:

- All names, including tables and columns, are case-sensitive.
- All queries must be in the form SELECT * FROM table only; you cannot include any SELECT clauses. To filter the results, load them into a business view, then filter that view.

Example of Context

Consider a view that joins an event table with a context table, such as the following:

```
SELECT Event.ID, Context.Name, Context.BDate AS Birth_Date
    FROM Event, Context     WHERE Event.ID = Context.ID
```

The context Output for this view might look like this:

```
Field Name       JDBC Data Type    Data Type
------------     --------------    ---------
Name             STRING            Varchar
Birth_Date       DATE              Timestamp
```

Example of Event Using a Polling Query

Consider the following query of an Events table:

```
SELECT * FROM Events
```

The resulting field information might look like this:

```
Field Name       JDBC Data Type    Data Type
------------     --------------    ---------
Event_ID         TINYINT           Integer
Product_ID       TINYINT           Integer
Product_Name     STRING            Varchar
```

Note that the polling Incrementing field is most likely Event_ID. See Polling the JDBC Source, for details.

Polling the JDBC Source

Polling tells the object how frequently to call the stored procedure or to query the DBMS for new events. Polling has the following parameters:

Parameter	Description
Polling interval	How often to call the procedure or issue the query.
Persist state across reboots	After a restart, continue polling using the state of the polling prior to stopping the system. At run-time, the object keeps track of values returned from the last call or query and uses them to determine the starting point of the next call or query. When the option is checked, that information is saved at every check point and when the system is shutdown. See Working with Checkpoint and Recovery for more information. Otherwise, when the option is not checked, polling restarts with the initial values defined for this object.
Disable after errors	Disables the object (stops polling) after consecutive errors occur. By default, the polling stops after five consecutive errors. To re-enable the object, change its Object Status to enabled. Set this option to zero (0) to never disable automatically.
Incrementing field (event query only)	Identifies the column in the source table that contains a value that increments for every event. For example, if the table being queried contains unique, ascending ID values, that field is the one used by the query using the logic *where ID greater that maximum ID from last query"*.
Initial value (event query only)	Value to use the first time the object queries the DBMS for events. For example, you might specify ID values starting with 500. For subsequent queries, the value must be greater than the Incrementing field value returned from the last query. **Note:** If you are using a SQLServer 2005 context database and a 2005 Microsoft driver, intial date values must be in the form *yyyy-mm-dd*; otherwise, an error may be generated.

Stored Procedure Source

The JDBC stored procedure source requires the following attributes:

Attribute	**Description**
Procedure name	Name of the stored procedure in the DBMS. JDBC events do not support *multiple result-set* or *stored procedure output parameters*. For Oracle this means that only Oracle functions are supported because only they return a single result.
Outputs	Columns in the event or context table, their data types, and optional formatting. The procedure returns a result set whose values map to the columns in the order they appear in this list. The data type identifies the type of the column in the table and will automatically be converted from the JDBC type as defined in "Mapping JDBC Data Types" on page 224.
Inputs	(*optional for events*) Parameters passed to the stored procedure and their data types. The parameters contain values to look up in the DBMS table. Inputs are passed as arguments to the procedure in the order they appear in the list. The data type identifies Business Activity Monitoring data type of the value being passed to the procedure. See Mapping JDBC Data Types," below for details. Further, For events, the procedure usually queries the DBMS looking for events inserted since the last time the procedure was called. This is done by identifying fields in the table that contain some incrementing or increasing values. For example, if the table being queried contains unique, ascending ID values, that field is the one used by the procedure using the logic "*where ID greater that maximum ID from last query*". For events, the Initial Polling Value specifies the value to use the first time the procedure queries the DBMS for events. For example, you might specify ID values starting with 500. For subsequent queries, the Subsequent Polling Value identifies a field that contains the maximum value from the last query. This value is an Output field from the previous result.
Polling	How often to call the stored procedure. See "Polling the JDBC Source" on page 221 for details.

Note: When making a query to a Sybase database, be aware that the names, including tables and columns, are case-sensitive.

Example of Receiving Context Using a Stored Procedure

Consider a view that joins an event table with a context table as follows:

```
SELECT Event.ID, Context.Name
    FROM Event, Context
    WHERE Event.ID = Context.ID
```

The context Output for this view is the Name column, and the Input to the procedure is the ID column.

```
Output Field Name  Data Type
-----------------  ---------
Name               Varchar

Input Field Name   Data Type
----------------   ---------
ID                 Integer
```

Example of Receiving Events Using a Stored Procedure

Consider an event with the following fields. Note that Event_Timestamp is the field with the unique and increasing value: each event record has a timestamp assigned by the DBMS.

```
Output Field Name  Data Type
-----------------  ---------
Ticket_ID          Varchar
Cust_ID            Varchar
Status             Varchar
Topic              Integer
When_Opened        Timestamp
Assigned_To        Varchar
Event_Timestamp    Timestamp
```

The event input identifies a parameter that passes the value to query. The name of the input must be unique to the list but is otherwise insignificant. In the following example, the field name is “IN1”.

```
Input Field Name   Initial Polling Value   Subsequent Polling Value
----------------   ---------------------   ------------------------
IN1                2003-03-05 19:45:00     Event_Timestamp
```

The first time the stored procedure queries the DBMS, it issues one semantically similar to the following:

```
SELECT * FROM events
    WHERE event_time >= "2003-03-05 19:45:00"
```

Subsequent queries use the results from the previous query as the starting point for new events.

Mapping JDBC Data Types

The data types of the DBMS columns are displayed as JDBC data types and map to Business Activity Monitoring Data Types as follows:

JDBC data type	Character	Integer	Double	Decimal	Timestamp	Boolean
CHAR	yes	—	—	—	—	as literal
VARCHAR	yes	—	—	—	—	as literal
LONGVARCHAR	yes	—	—	—	—	as literal
NUMERIC	—	yes	yes	yes	—	yes
DECIMAL	—	yes	yes	yes	—	yes
BIT	—	yes	yes	yes	—	yes
TINYINT	—	yes	yes	yes	—	yes
SMALLINT	—	yes	yes	yes	—	yes
INTEGER	—	yes	yes	yes	—	yes
BIGINT	—	yes	yes	yes	—	yes
REAL	—	yes	yes	yes	—	yes
FLOAT	—	yes	yes	yes	—	yes
DOUBLE	—	yes	yes	yes	—	yes
BINARY	—	—	—	—	—	—
VARBINARY	—	—	—	—	—	—
DATE	—	—	—	—	yes	—
TIME	—	—	—	—	yes	—
TIMESTAMP	—	—	—	—	yes	—

Note: Do not use Business Activity Monitoring Boolean data type in a WHERE predicate passed to the JDBC source. Boolean values may be included in the Select list.

JDBC Agents

A Java database connectivity (JDBC) agent communicates with a relational database (DBMS) by either making a query on the database, or calling a stored procedure in the DBMS. The DBMS then returns one or more rows of data, which the agent passes on to the requesting event or context JDBC Tables.

Note: JDBC agents are synchronous, they retrieve event messages and context data as the result of a specific request as summarized in the following table. For context, the agents access the DBMS when a new event requires context data. For events, the agent periodically polls the DBMS to see if new events are available, then retrieves them for inclusion in the events table. Each event returned is processed individually, regardless of the count of events returned as a result of the polling query.

Event push	**Event pull**	**Context pull**
No	Yes	Yes

Before creating a JDBC agent, you need:

- Create permission for agents (see “Creating Permission” on page 258 for details).
- A JDBC data source defined and managed by the application server, preferably one that pools connections. Note, configure the connection pool as documented in your application server's documentation. Additionally, in the pool's definition:
  - Set the maximum number of open connections to the database to be at least 200.
  - Set a refresh rate to be greater than 0, preferably to 1 or 2 minutes. This allows the database to go down and come back up with loosing of the connection from the pool. Further, you should set the pool to test for the existence of a physical table in the database.

Attributes

A JDBC agent has the following attributes:

Attribute	**Description**
Name	Identifies the agent. This name must be unique among agents. See “Object Namespace” on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the agent is enabled (monitoring for events) or disabled (not monitoring for events).
Database type	Identifies the DBMS vendor as Oracle, Sybase, SQL Server, or DB2.
User name	User name to use to connect to the DBMS. Must have *query* access rights.
Password	Password for the User name. If you omit this option, the agent uses the password specified in the JDBC Source configuration definition in the application server.
Max rows per query	Maximum count of rows to return as the query result. Useful to keep users from returning exceptionally large results that impact the DBMS.
Type of JDBC connection	How to connect to the JDBC in the application server. • Datasource-based: Connects to a JDBC database pool through a JNDI connection. Do not use for IBM Websphere. • URL-based: Connects to a JDBC source via a URL.

Attribute	Description
JNDI name for JDBC Source (Datasource only)	Name of the data source to use as a connection to the database. The name is in JNDI form, such as "com.celequest.products.ProductSource." A source gets its connection from a pool of connections maintained by the application server. That server keeps the connections open to reduce delays when establishing a connection. **Note:** The connection pool must be configured as a non-transactional pool; non-TxT on WebLogic.
JNDI properties (Datasource only)	Optional and additional Java naming and directory interface (JNDI) properties necessary to make or maintain the agent to the JDBC source. These name-value pairs allow you to specify JDBC properties. The names are either one of the short cuts listed below or a JNDI recognized property. The agent recognizes the following names as short cuts to JNDI properties: • factory maps to INITIAL_CONTEXT_FACTORY. • provider maps to PROVIDER_URL. • security_credentials maps to SECURITY_CREDENTIALS. • security_principal maps to SECURITY_PRINCIPAL.
JDBC URL (URL only)	URL that maps to the JDBC connection configured in the application server running Business Activity Monitoring. For example, a URL might look like the following: jdbc:oracle:thin:some_context/context@v480:1521:symbols
JDBC driver class (URL only)	JDBC driver to use. This driver must reside in the classpath of the application server running Business Activity Monitoring. Include the complete classname, such as oracle.jdbc.driver.OracleDriver

Creating a JDBC Agent

Use the following procedures to create a JDBC agent.

➤ **To create a JDBC agent:**

1. Open the BAM Workbench Administration Console.
2. Click **New Agent.**
3. Choose JDBC as the source type
4. Fill in the fields that define the agent's attributes.

Save the agent as enabled, and it will immediately be ready to retrieve data.

17 JDBC Access to View Data

Business Activity Monitoring provides an application programming interface (API) that allows JDBC 2.0 applications to retrieve data from a view, and to retrieve the metadata that describes the views in the installation. The data that you can retrieve are:

- View data from the recent view. Note that if the view contains a (moving set) window, the window data is returned.
- Event identifier (VC_EVENT_ID always included) which identifies the event that produced the most recent row included in the view.
- Latest event identifier (VC_LATEST_EVENT_ID always included) which identifies the last event that caused the view to update, though data from that event might not be included in the view.
- Event timestamp (VC_TIMESTAMP always included) which identifies when the last event was included in the view.

Note: This is the same information that is written to a database when persisting views. See "Persisting Views to a Database" on page 359 for more information.

➤ In This Chapter:

Classpath

The classpath to Business Activity Monitoring JDBC driver (cqjdbcclient.jar) needs to be added to the client JDBC application. The client application should also link in the application server (such as weblogic.jar) to have access to the JNDI naming service. For example, when running a program (for example, Test) from a command line, the java call might look like the following on a BEA WebLogic Server:

```
java -classpath .;c:\bea\weblogic700\server\lib\weblogic.jar;
c:\cq\cqjdbcclient\cqjdbcclient.jar JDBCAccessor
```

The rest of this chapter describes the JDBC View Interfaces and provides “JDBC Accessor Examples” on page 235.

JDBC View Interfaces

JDBC 2.0 defines interfaces for accessing data. Business Activity Monitoring implements the interfaces for accessing its data listed in table below. For complete details about the interfaces, see the JDBC documentation at http://java.sun.com/j2se/1.3/docs/api/java/sql/package-summary.html.

Note: All methods return data that meet the JDBC 1.0 standard. Further, methods defined in the JDBC class but which not supported in Business Activity Monitoring API throw an SQLException.

The following are the interfaces that Business Activity Monitoring supports:

Class	Member	Description
Driver	The JDBC SQL database driver. See “Example: Establishing a connection to the BAM Server” on page 236 for an example. The class name is: com.celequest.jdbc.driver.Driver	
	connect	Attempts to make a database connection to the URL.
	acceptsURL	Returns true if this driver understands the specified subprotocol.
	getMajorVersion	Returns the driver’s major revision number.
	getMinorVersion	Returns the driver’s minor revision number.
	jdbcCompliant	Returns false.
Connection		A connection to a specific database. See “Example: Establishing a connection to the BAM Server” on page 236 for an example of using this interface.
	createStatement	Returns a newly created Statement object
	close	Releases a Connection object’s JDBC resources immediately.
	getMetaData	Returns a newly created DatabaseMetaData object.
	isClosed	Returns true if the calling Connection object is closed; otherwise, returns false when it is still open.

Class	Member	Description
Statement		An SQL statement to pass to the database. See "Example: Querying the Contents of a View" on page 238 for an example of using this interface.
	executeQuery	Executes a Business Activity Monitoring C-SQL select query, returns a single ResultSet, and closes the calling Statement object's current ResultSet, if any. The query statement involves a single table only and may contain WHERE, GROUP BY, and ORDER clauses. The viewname references in the query are case insensitive and my be quoted (using the double quote character).
	close	Releases a Statements object's JDBC resources.
DatabaseMetaData		Provides information about the view definitions defined in Business Activity Monitoring installation.
	getTables	Returns view definitions. See "Example: Querying View Metadata" on page 241 for an example of using this method.
	getColumns	Returns the column information for a given view. See "Example: Querying Column Metadata" on page 240 for an example of using this method. See "getColumns() Column Summary" on page 232 for a summary listing of the columns.

Class	Member	Description
ResultSet		A table of data representing a database result set, which is usually generated by executing a statement that queries the database.
	next	Moves the cursor to the next row in the set and fixes the current row.
	close	Immediately releases a ResultSet's JDBC resources.
	wasNull	Returns true if the last value read was SQL NULL.
	getString	Returns the value of a column as a Java String.
	getBoolean	Returns the value of a column as a Java Boolean.
	getInt	Returns the value of a column as a Java int.
	getDouble	Returns the value of a column as a Java double.
	getObject	Returns the value of a column as a Java object (as defined in the default type mapping).
	getBigDecimal	Returns the value of a column as a java.math.BigDecimal object.
	getTimestamp	Returns the value of a column as a Java Timestamp.
	getMetaData	Returns the number, types, and properties of a ResultSet object's columns as a ResultSetMetaData object.
ResultSetMetaData		Provides information about the types and properties of the columns in a ResultSet object.
	getColumnCount	Returns the count of columns in the ResultSet object.
	getPrecision	Returns the designated column's number of decimal digits.
	getScale	Returns the count of digits to the right of the decimal separator.
	getTableName	Returns the table name from which the ResultSet was derived.
	getColumnName	Returns the name of a column.
	getColumnType	Returns the JDBC type for the value stored in a column. See "Data Type Mappings" on page 232 for a summary of the mappings.
	getColumnTypeName	Returns the Business Activity Monitoring type name for the a column.

Data Type Mappings

The “get” functions in this class return values from Business Activity Monitoring columns. This matrix indicates which functions should be used for the various Business Activity Monitoring data types.

	getBoolean	getInt	getDouble	getBigDecimal	getString	getTimestamp
Boolean	X	o	o	o	o	—
Integer	o	X	o	o	o	—
Double	o	o	X	o	o	—
Decimal	o	o	o	X	o	—
Varchar	o	o	o	o	X	x
Timestamp	—	—	—	—	x	X

X indicates that the function returns a value compatible with Business Activity Monitoring data type.

o indicates data types that might be compatible, but whose conversion is not recommended.

getColumns() Column Summary

This table is a summary of the Java documentation for DatabaseMetaData.getColumns(). See the Java documentation for a complete list.

Column	Type	Description
TABLE_CAT	String	Table catalog (may be null)
TABLE_SCHEM	String	Table schema (may be null)
TABLE_NAME	String	Table name
COLUMN_NAME	String	Column name
DATA_TYPE	int	SQL type from java.sql.Types. See DATA_TYPE Return Values,” below, for a summary.
TYPE_NAME	String	Data source dependent type name, for a UDT the type name is fully qualified
COLUMN_SIZE	int	Column size. For char or date types this is the maximum number of characters, for numeric or decimal types this is precision.
BUFFER_LENGTH	String	Not used.
DECIMAL_DIGITS	int	Count of fractional digits
NUM_PREC_RADIX	int	Radix (typically either 10 or 2)

Column	**Type**	**Description**
NULLABLE	int	Is NULL allowed. ● columnNoNulls - might not allow NULL values ● columnNullable - definitely allows NULL values ● columnNullableUnknown - nullability unknown
REMARKS	String	Comment describing column (may be null)
COLUMN_DEF	String	Default value (may be null)
SQL_DATA_TYPE	int	Not used.
SQL_DATETIME_SUB	int	Not used.
CHAR_OCTET_LENGTH	int	For char types the maximum number of bytes in the column.
ORDINAL_POSITION	int	Index of column in table (starting at 1)
IS_NULLABLE	String	"NO" means column definitely does not allow NULL values; "YES" means the column might allow NULL values. An empty string means nobody knows.

DATA_TYPE Return Values

The DATA_TYPE column returns an int value that identifies the Java data type. See the java.sql.Types file for details. The following table summarizes those values.

Type	**Value**
BIT	-7
TINYINT	-6
BIGINT	-5
LONGVARBINARY	-4
VARBINARY	-3
BINARY	-2
LONGVARCHAR	-1
NULL	0
CHAR	1
NUMERIC	2
DECIMAL	3
INTEGER	4

Type	**Value**
SMALLINT	5
FLOAT	6
REAL	7
DOUBLE	8
VARCHAR	12
DATE	91
TIME	92
TIMESTAMP	93
OTHER	1111
JAVA_OBJECT	2000
DISTINCT	2001
STRUCT	2002
ARRAY	2003
BLOB	2004
CLOB	2005
REF	2006

JDBC Accessor Examples

The examples in this section demonstrate how to connect to the BAM Server and query view data, metadata, and metadata about the views and columns defined in the installation. The examples, include:

- "Example: Establishing a connection to the BAM Server" on page 236
- "Example: Querying the Contents of a View" on page 238
- "Example: Querying a View's Column Specifications" on page 238
- "Example: Querying Column Metadata" on page 240
- "Example: Querying View Metadata" on page 241

Java

Access to the JDBC driver depends on the java.sql.* classes. As such, be sure to include the following import in your applications;

```
import java.sql.*;
```

Complete Sample

Included on the product CD-ROM is a sample application that contains the complete code for the snippets listed in these examples. See the \samples\JDBC\ directory for the files. That directory has two files:

- readme.txt describes how to compile and run the application.
- JDBCAccessor.java is the application.

Briefly, to compile the application, use the following command:

```
javac -classpath . JDBCAccessor.java
```

To run the program, load it into the application server, similar to the following:

BEA WebLogic:

```
java -classpath .;c:\bea\weblogic700\server\lib\weblogic.jar;
c:\cq\cqjdbcclient\cqjdbcclient.jar JDBCAccessor
```

JBoss:

```
java -classpath .;C:\jboss\3.2.3\client\jnet.jar;
C:\jboss\3.2.3\client\jboss-net-client.jar;
C:\jboss\3.2.3\client\jnp-client.jar;
C:\jboss\3.2.3\client\jboss-common-client.jar;
c:\cq\cqjdbcclient\cqjdbcclient.jar JDBCAccessor
```

The results from the examples print to the standard output, with errors going to standard error.

Example: Establishing a connection to the BAM Server

This example shows how to establish a connection to the BAM Server's JDBC driver.

Establish the connection by creating a Connection object, similar to the following:

```
Connection connection = null;
connection = DriverManager.getConnection(url, userName, password);
```

The userName and password parameters identify a Business Activity Monitoring account. When querying a specific object (like a view), the user account must have at least Read Only access permission. Otherwise, if the account has No Access to the view, the query fails as if the view does not exist.

The url parameter identifies the factory in the application server that establishes the connection to Business Activity Monitoring JDBC driver. This URL specifies the type of connection, host and port to connect to, and the factory in the application server. Further properties vary among application servers. The sample application names the common properties in variables that you can customize for your installation:

```
String userName    = "system";
String password    = "manager";
```

The sample application that builds the url parameter from the properties specific to the application server as follows:

BEA WebLogic connection properties

```
String hostAndPort = "localhost:80";
String factory = "weblogic.jndi.WLInitialContextFactory";
String url     = "jdbc:celequest:factory="+factory+
                  ";provider=t3://"+hostAndPort+";";
```

JBoss connection properties

```
String hostAndPort = "localhost:1099";
String factory = "org.jnp.interfaces.NamingContextFactory";
String url     = "jdbc:celequest:java.naming.factory.url.pkgs="+
                "org.jboss.naming:org.jnp.interfaces;"+
                "factory="+factory+";provider=jnp://"+hostAndPort+";";
```

Following is a more detailed example. However, to see the complete listing, examine the JDBCAccessor.main() member.

```
// Common connection properties
String userName    = "system";
String password    = "manager";

// JBoss connection properties
String hostAndPort = "localhost:1099";
String factory = "org.jnp.interfaces.NamingContextFactory";
String url     = "jdbc:celequest:java.naming.factory.url.pkgs="+
               "org.jboss.naming:org.jnp.interfaces;"+
               "factory="+factory+";provider=jnp://"+hostAndPort+";";

// Verify the JDBC driver in the application o
try {
    Class.forName("com.celequest.jdbc.driver.Driver");
} catch (ClassNotFoundException e) {
    handleError("Could not find the JDBC driver class.", e);
    return;
}

// Establish the connection to the  JDBC driver
Connection connection = null;

try {
  connection = DriverManager.getConnection(url, userName, password);
} catch (SQLException e) {
    handleError("Could not connect to the JDBC driver.", e);
    return;
}
```

Example: Querying the Contents of a View

This snippet shows how to query the entire contents of a view. The executeQuery() call passes the query to the driver, which returns the view contents in a ResultSet object. All columns, including the internal system columns like VC_TIMESTAMP are included in this list. Additionally, metadata about the view is retrieved in a ResultSetMetaData object to determine the count of columns in the view.

Note: See the JDBCAccessor.PrintViewContents() sample for a complete code listing.

```
/* Query all contents of a view.
 * Connection has already been established, and view name defined.
 */
String queryString = "SELECT * FROM " + VIEW_NAME ;
ResultSet rs;                // Table to hold the query results.
ResultSetMetaData rmd;       // Metadata about the result set.

// Query the view, and get its data and metadata.
Statement stmt = connection.createStatement();
rs = stmt.executeQuery( queryString );
rmd = rs.getMetaData();

// Print the contents of the entire view, row by row.
int columnCount = rmd.getColumnCount();
boolean isEmpty = true;
while (rs.next())  {
  isEmpty = false;
  System.out.print(" Row: ");
  for (int i=0;i<columnCount;i++) {
    // Show the column value, or "NULL"
    String ts = rs.getString(i+1);
    System.out.print( (rs.wasNull() ? "NULL " : ts + " "));
  }
  System.out.println();  // Line break
}
if (isEmpty) {
  System.out.println("\n *** The view is empty ***");
}
```

The result might look like the following:

```
Contents of view [OrderProductTotals]:
Row: Hinges 132300.00 49 130000.00 28 2004-08-17 11:22:06.818 28
Row: Lag bolts 16400.00 41 65000.00 21 2004-08-17 11:22:06.818 28
Row: Nails 129600.00 48 150000.00 26 2004-08-17 11:22:06.818 28
Row: Nuts 337875.00 159 280000.00 27 2004-08-17 11:22:06.818 28
Row: Screws 60000.00 30 80000.00 20 2004-08-17 11:22:06.818 28
Row: Washers 122400.00 72 170000.00 23 2004-08-17 11:22:06.818 28
Row: Chip board 277200.00 126 250000.00 18 2004-08-17 11:22:06.818 28
Row: Plywood 304800.00 127 250000.00 10 2004-08-17 11:22:06.818 28
```

Example: Querying a View’s Column Specifications

This snippet shows how to query the user-defined specifications about the columns in a view. First it shows all of the metadata available for column specifications, then it shows interesting specifications about each column in the view, including the internal system columns. Note that the results appear *in the order that the columns appear in the view*, followed by the internal columns.

Note: See the JDBCAccessor.PrintColumns() sample for a complete code listing.

```
/* Query a view's column specifications.
 * Connection has already been established, and view name defined.
 */
// Query the table to identify the columns to report on. Because the
// view contents are irrelevant here, omit them by declaring
// 'WHERE false' as the query condition.
String queryString = "SELECT * FROM " + VIEW_NAME +" WHERE false";

ResultSet rs;                // Table to hold the query results.
ResultSetMetaData rmd;       // Metadata about the result set.

// Query the view, and then get its metadata.
Statement stmt = connection.createStatement();
rs = stmt.executeQuery( queryString );
rmd = rs.getMetaData();

// Print the metadata about the columns in the view.
System.out.println("Column details for view [" +
                  rmd.getTableName(1) +"]:");

// Walk through and show the interesting metadata available for
// each column in the view. Include labels to identify what we see
// in the result.
for (int i=0;i<rmd.getColumnCount();i++) {
  System.out.println(
     " "               + Integer.toString(i+1) +
     ". Name ["              + rmd.getColumnName(i+1) +
     "] Type ["        + Integer.toString(rmd.getColumnType(i+1)) +
     "] Precision [" + Integer.toString(rmd.getPrecision(i+1)) +
     "] Scale ["       + Integer.toString(rmd.getScale(i+1))+"]");
}
```

The results might look like the following:

```
View: [ORDERPRODUCTTOTALS]
Column details for view [ORDERPRODUCTTOTALS]:
 1. Name [FAMILY] Type [12] Precision [20] Scale [0]
 2. Name [PRODUCT] Type [12] Precision [50] Scale [0]
 3. Name [SALES] Type [3] Precision [25] Scale [2]
 4. Name [QTY] Type [4] Precision [10] Scale [0]
 5. Name [TARGET] Type [3] Precision [15] Scale [2]
 6. Name [TARGET MIN] Type [3] Precision [15] Scale [2]
 7. Name [TARGET MAX] Type [3] Precision [15] Scale [2]
 8. Name [VC_EVENT_ID] Type [4] Precision [10] Scale [0]
 9. Name [VC_TIMESTAMP] Type [93] Precision [0] Scale [9]
 10. Name [VC_LATEST_EVENT_ID] Type [4] Precision [10] Scale [0]
```

Example: Querying Column Metadata

These snippets show how to retrieve metadata about the columns view. (To retrieve the column specifications instead, see "Example: Querying a View's Column Specifications" on page 238.)

Note: See the JDBCAccessor.PrintColumnMeta() sample for a complete code listing.

```
/* Retrieve the metadata about the columns of a defined view.
 */
// NOTE: Specify 'null' instead of 'VIEW_NAME' to get the metadata
//       for ALL columns in the installation.
rs = meta.getColumns(null,null,VIEW_NAME,null);
rmd = rs.getMetaData();

// Show the metadata available for view columns.
int columnCount = rmd.getColumnCount();
for (int i=0;i<columnCount;i++) {
 System.out.print(rmd.getColumnName(i+1) +
      ((i+1)==columnCount ? "" :  ","));
}
```

The results first lists the metadata column names (see "getColumns() Column Summary" on page 232 for a description of the columns):

```
Getting column metadata for [OrderProductTotals]
TABLE_CAT,TABLE_SCHEM,TABLE_NAME,COLUMN_NAME,DATA_TYPE,TYPE_NAME,
COLUMN_SIZE,BUFFER_LENGTH,DECIMAL_DIGITS,NUM_PREC_RADIX,NULLABLE,
REMARKS,COLUMN_DEF,SQL_DATA_TYPE,SQL_DATETIME_SUB,CHAR_OCTET_LENGTH,
ORDINAL_POSITION,IS_NULLABLE
```

Next, show all of the metadata about the columns in a specific view. Note that columns *do not* appear in the order that they appear in the view.

```
// Show the metadata values for the columns in the view.
boolean isEmpty = true;
while (rs.next()) {
 isEmpty = false;
 for (int i=0;i<columnCount;i++) {
      String ts = rs.getString(i+1);   // Metadata value
      if (rs.wasNull())
         System.out.print(",");
      else
         System.out.print(ts +
              ((i+1)==columnCount ? "" :  ","));
 }
 System.out.println();
}
if (isEmpty) {
  System.out.println("\n *** Either there are no columns defined "+
                 "for this view (unlikely), or the view is not "+
                 "defined (probably).");
}
```

The following is a sample listing of the metadata for the OrderProductTotals view. (See "Data Type Mappings" on page 232 for a mapping of the data types from Java). Again, the columns *do not* appear in any particular order.

```
,,OrderProductTotals,VC_EVENT_ID,4,VCInteger,10,,0,10,1,,,,,,1,YES
,,OrderProductTotals,VC_TIMESTAMP,93,VCTimestamp,9,,0,10,1,,,,,,3,YES
,,OrderProductTotals,Target max,3,VCDecimal,15,,2,10,1,,,,,,4,YES
,,OrderProductTotals,VC_LATEST_EVENT_ID,4,VCInteger,10,,0,10,1,,,,,,5,YES
,,OrderProductTotals,Family,12,VCVarchar,20,,0,10,1,,,,,,6,YES
,,OrderProductTotals,Target min,3,VCDecimal,15,,2,10,1,,,,,,7,YES
,,OrderProductTotals,Qty,4,VCInteger,10,,0,10,1,,,,,,9,YES
,,OrderProductTotals,Product,12,VCVarchar,50,,0,10,1,,,,,,10,YES
,,OrderProductTotals,Target,3,VCDecimal,15,,2,10,1,,,,,,11,YES
,,OrderProductTotals,Sales,3,VCDecimal,25,,2,10,1,,,,,,12,YES
```

Example: Querying View Metadata

These snippets show how to query view metadata. First it shows the metadata available for views, then it shows how to find all of the views defined in the system.

Note: See the JDBCAccessor.PrintAllViewsMeta() sample for a complete code listing.

```
/* Retrieve metadata about views.
 */
ResultSet rs;
ResultSetMetaData rmd;
DatabaseMetaData meta = connection.getMetaData();

// Use 'null' for the 3rd parameter to retrieve information about all
// of the views in the system, instead of just one.
rs = meta.getTables(null,null,null,null);
rmd = rs.getMetaData();
for (int i=0;i<rmd.getColumnCount();i++) {
  System.out.println(
     " ["+ rmd.getColumnName(i+1) +
     "] Type ["      + Integer.toString(rmd.getColumnType(i+1)) +
     "] Precision [" + Integer.toString(rmd.getPrecision(i+1)) +
     "] Scale ["     + Integer.toString(rmd.getScale(i+1))+"]");
}
```

The results first describe the metadata that is available:

```
This metadata is available for views.
 [TABLE_CAT] Type [12] Precision [255] Scale [0]
 [TABLE_SCHEM] Type [12] Precision [255] Scale [0]
 [TABLE_NAME] Type [12] Precision [255] Scale [0]
 [TABLE_TYPE] Type [12] Precision [255] Scale [0]
 [REMARKS] Type [12] Precision [255] Scale [0]
```

Next, list the views (table names) defined in the system:

```
// Use the metadata to list all of the views in the system.
final int GT_TABLE_NAME  = 3;  // View name
boolean isEmpty = true;
while (rs.next()) {
  isEmpty = false;
  System.out.println(" " + rs.getString(GT_TABLE_NAME) );
}
if (isEmpty) {
  System.out.println(" *** There are no views defined. ***");
}
```

The results look similar to the following:

```
 OrderChangeDetails
 OrderTotals
 8WeekOrders
 8WeekOrdersAvg
 OrderProductTotals
 OrderSalesGrandTotal
 30DayOrders
 InventoryChangeDetails
 SupplierAlternates
```

18 Objects

Objects manage data in Business Activity Monitoring. Every object has a name, optional description, and a status that determines if it is able to work with its intended data. Further, each object has additional information that you define that tells the object what data to manage, and how to manage it.

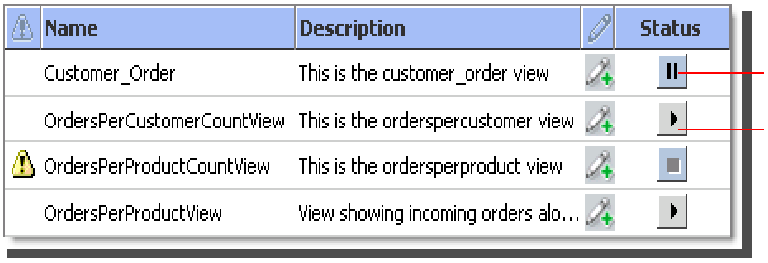
➤ **In this Chapter:**

Object Status

Every object has one of three statuses that determine its ability to operate:

Icon	Description
▶	Enabled — The object is accepting new data and processing them. You can only enable valid objects that do not depend on disabled objects. When you enable a disabled object, you have the choice of enabling just that object, or that object and all objects that depend on that object (*cascade enable*). Further, all of the dependant objects must be capable of being enabled — none may be invalid — or the entire operation fails and no objects are enabled.
■	Disabled — The object is not accepting new data. Disabling an object does not affect the definition or existence of that object; rather, it just keeps new data from flowing into the object and to all objects that depend on the object. For example, disabling a view also disables all rules that monitor the view, and thereby also disables all associated alerts and reportlets.
⏸	Disabled dependant— The object is not accepting new data because an object that this one depends on is disabled. Enabling the referenced object will also enable this object.
⚠	Invalid — The object that has a reference to another object which cannot be satisfied, such as one view that references a column in another view, but that column no longer exists in the referenced view. An object can be invalid because a referenced object does not exist or because some attribute of the object does not match the requirements of the dependent (such as a data type mismatch or a missing column name). Invalidating an object also invalidates all objects that depend on the object. This usually happens when you delete an object that has dependencies, or change an object's definition.

When you view a list of objects, you can see each object's valid/invalid state, and the enabled/disabled status. You can click on an enabled indicator in the Status column to disable it, or click on a disabled indicator to enable it.



⚠	Name	Description		Status
	Customer_Order	This is the customer_order view		⏸
	OrdersPerCustomerCountView	This is the orderspercustomer view		▶
⚠	OrdersPerProductCountView	This is the ordersperproduct view		■
	OrdersPerProductView	View showing incoming orders alo…		▶

Object Names

All object names must be either a *regular identifier* or a *delimited identifier*.

- Regular Identifier — The first character of a regular identifier must begin with a letter from ‘a’-‘z’ or ‘A’-‘Z’, and all subsequent characters can be from ‘a’-‘z’, ‘A’-‘Z’, ‘0’-‘9’, or ‘_’.
- Delimited Identifier — A delimited identifier must start and end with a double quote ("). The body of a delimited identifier must be non-empty and can contain any SQL language characters including: the Regular Identifier characters, underscore (‘_’), space(‘ ’), percent(‘%’), ampersand (‘&’), single quote, left parenthesis (‘(’), right parenthesis (‘)’), asterisk (‘*’), plus sign (‘+’), comma (‘,’), minus sign (‘-’), slash (‘/’), colon (‘:’), semicolon (‘;’), equals operator (‘=’), question mark (‘?’), vertical bar (‘|’) or double quote ("") (escaped with another double quote).

  Note: Names may not contain periods (‘.’), less than (‘<’), or greater than (‘>’) characters.

All names must be unique within their class (such as view or agents; see “Object Namespace” on page 248 for details) and may not be identical to reserved words. For example, you cannot have a view named by the regular identifier select, though you can have a delimited one named "select".

Reserved Words

All identifiers beginning with “VC_” are reserved system names and may not be used. Further, all reserved words in the SQL-99 standard are reserved in C-SQL. The following are Business Activity Monitoring reserved words:

Reserved Words			
abs	delete	local	rollback
absolute	desc	log	round
acked	describe	lower	rows
action	descriptor	lowered	rpad
add	diagnostics	lpad	rtrim
all	disconnect	ltrim	schema
allocate	distinct	match	scroll
alter	domain	max	second
and	double	min	section
any	drop	minute	select
are	else	mod	session
as	end	module	session_user
asc	end-exec	month	set
assertion	escape	mov_avg	sign

Reserved Words			
at	event	mov_count	size
authorization	except	mov_max	smallint
avg	exception	mov_min	some
begin	exec	mov_ntile	space
between	execute	mov_rank	sql
bit	exists	mov_ratio_to_report	sqlcode
bit_length	exp	mov_std_deviation	sqlerror
boolean	external	mov_sum	sqlstate
both	extract	mov_variance	sqrt
by	false	names	std_deviation
cascade	fetch	national	substr
cascaded	first	natural	substring
case	float	nchar	sum
cast	floor	next	system_user
catalog	for	no	table
ceil	foreign	not	temporary
char	found	ntile	then
char_length	from	null	time
character	full	nullif	timestamp
character_length	get	numeric	timestamp_diff
check	global	octet_length	timezone_hour
close	go	of	timezone_minute
coalesce	goto	on	to
collate	grant	only	to_char
collation	greatest	open	to_date
column	group	option	trailing
commit	having	or	transaction
concat	hour	order	translate

Reserved Words			
connect	identity	outer	translation
connection	immediate	output	trim
constraint	in	overlaps	true
constraints	indicator	pad	trunc
continue	initially	partial	union
convert	inner	position	unique
corresponding	input	power	unknown
count	insensitive	precision	update
create	insert	prepare	upper
cross	int	preserve	usage
current	integer	prev	user
current_date	intersect	primary	using
current_time	interval	prior	value
current_timestamp	into	prior_value	values
current_user	is	privileges	varchar
cursor	is_raised	procedure	variance
date	isolation	public	varying
date_add	join	raised	view
date_diff	key	rank	when
day	language	ratio_to_report	whenever
deallocate	last	read	where
dec	last_day	real	with
decimal	leading	references	work
declare	least	relative	write
default	left	restrict	year
deferrable	level	revoke	zone
deferred	like	right	

Object Namespace

The namespace controls how objects are named within Business Activity Monitoring. Generally, object names must be unique among other objects of the same type, within the same container. However, here are some exceptions:

- Alerts, rules, and reportlets *can* share the same name within the containing scenario; you can use the same name for one alert, one rule, and one reportlet within a scenario. Further, each object within a scenario may share the same name as an object of the same type in another scenario.
- Profiles must be unique within a single user; multiple users may share profile names.
- Views, events, context, etc., cannot share the same name; they must be unique within the "tables" class. See the summary below for details.
- Agents must be unique within the "agents" class.
- Users and business activities cannot share the same name; they must be unique within the "containers" class. The following outline summarizes the namespace constraints:

```
/containers
    /Business activities         Unique among /containers
        /Scenarios
            /Alerts
            /Rules
            /Reportlets
    /Users and Roles             Unique among /containers
        /E-mail profiles         Unique within a user
        /RTD (Excel) profiles    Unique within a user
/tables
    /Events                      Unique among /tables
    /Context                     Unique among /tables
    /Consolidated events         Unique among /tables
    /Cube                        Unique among /tables
    /Dimensions                  Unique among /tables
    /Views                       Unique among /tables
/agents
    /Flat (text) files           Unique among /agents
    /SOAP (Web services)         Unique among /agents
    /JMS                         Unique among /agents
    /JDBC                        Unique among /agents
    /Rendezvous (TIBCO)          Unique among /agents
/External actions (processes)
/Join relationships
```

19 Operators and Constants

This chapter describes the operators and constants that Business Activity Monitoring and C-SQL support in expressions and arguments.

➤ **In this Chapter:**

Numeric Operators

There are two classes of numeric operators: prefix and infix.

Prefix operators

Control the arithmetical sign of numeric values.

Prefix Operator	Description
+	Unary plus
-	Unary minus

Infix Operators

Numeric operators perform arithmetical operations on numeric values:

Numeric Operator	Description	Example
+	Addition	11 + 3 returns 14
-	Subtraction	11 - 3 returns 8
*	Multiplication	11 * 3 returns 33
/	Division	11 / 3 returns 3

To perform arithmetic operations on date-time values, use DATE_ADD() and DATE_DIFF().

String operators

Concatenation (||) is the only string operator and it appends the right-side string to the end of the left-side string. For example 'a'||'b' returns 'ab'. The behavior is identical to the CONCAT() function.

Comparison operators

Comparison operators compare two or more values of the same data type and return a Boolean:

Operator	Description	Example
=	Equal	1=1 returns TRUE
<>	Not equal	'A'<>'a' returns TRUE
>	Greater than	CURRENT_DATE()>TO_DATE('02/28/1963') returns TRUE
<	Less than	Age<21 returns UNKNOWN when Age is NULL
>=	Greater than or equal	TRUE>=FALSE returns TRUE
<=	Less than or equal	NULL<=NULL returns NULL
IN	Is a member of a list	Symbol IN ('IBM', 'MSFT', 'VCLR') or Count NOT IN (5, 10, 15, 20)
BETWEEN/AND	Is within a range	SalePrice BETWEEN 50.0 AND (90.0) or NOT BETWEEN 'M' AND 'O'
LIKE	Pattern matching. See below for details.	Title LIKE 'MR_'

LIKE Operator

The LIKE operator matches a pattern of characters. A percent sign (%) in the pattern is a wildcard for zero or more characters, and an underscore (_) is a wildcard for exactly one character.

```
WHERE Title LIKE 'MR_'
WHERE E_Mail NOT LIKE '%.edu'
```

To include either '%' or '_' in the search string, use the keyword ESCAPE to designate an escape character. A '%' or '_' following an escape character is treated as a literal. Note that the escape character may not be used elsewhere in the search string. The following example looks for "10%" anywhere in Discount:

```
WHERE Discount LIKE '%10$%' ESCAPE '$'
```

Note: An escape character prefixing anything other than an escape or special character is ignored.

Be careful about using LIKE when comparing against numeric types. LIKE is a string operator, and as such, searching a numeric first performs an Implicit Cast of the numeric value to a string. When casting numerics to strings, be aware of the following:

- For DECIMAL numbers, casting to string zero-pads the decimal values to match the precision and scale defined for the column. So, for example, if a column is defined as precision 5 and scale 4, a value of 1.1 in the column is cast as '1.1000', and so searches for single digit decimals must be done as LIKE '_._000'.
- For DOUBLE PRECISION numbers, the 'e' is cast to upper case. So, for example, +1e11 is converted to '1.0E11'.

Logical operators

Logical operators compare Boolean values, such as the result of a comparison operation.

Logical Operator	Description	Example
AND	Both true	(SalesPrice>500) AND (OnSale)
OR	One must be true	(ZipCode = '90210') OR (City = 'Lodi')
NOT	Inverse	NOT OnSale
IS	Test of Boolean	IS OnSale or IS NOT OnSale

The truth table for the equal sign (=) operator is equivalent to IS. See "Truth Table for IS" on page 68 for details.

Constants

The C-SQL Boolean constants are TRUE, FALSE, and UNKNOWN. See "Boolean" on page 68 for details about these constants.

Constants	Description
TRUE	True.
FALSE	Not true.
NULL	No data.
UNKNOWN	Test for Boolean value that is NULL, or where a comparison cannot be determined such as when comparing null to null.

Note: NULL is ignored when computing set function, moving set function, and rank function values. For example, the average of (3, NULL, 3) is 3, not NULL and it is not 2.

NULL is a null value. Any non-Boolean column which does not have an entry is considered NULL.

```
WHERE SalesPrice = NULL      Returns TRUE
WHERE SalesPrice = UNKNOWN   Error, cannot cast Numeric to Boolean
```

However, testing a Boolean column returns UNKNOWN when the column is empty.

```
WHERE OnSale = NULL          Returns UNKNOWN (null = null)
WHERE OnSale = UNKNOWN       Returns UNKNOWN (null = unknown)
WHERE OnSale IS NULL         Returns TRUE
WHERE OnSale IS UNKNOWN      Returns TRUE
```

20 Permissions

Permissions control which users may see, create, and edit Business Activity Monitoring objects and user accounts.

➤ **In this Chapter:**

Application of Permissions

Permissions can be applied in two places:

- On a specific object from the Permissions button after selecting the object in a list. When you assign a permission a specific object, it is the *maximum* permission that the user has to *that object*. You cannot set a user's permission to a specific object lower than that user's permission to the class of objects.
- At the class level from the Administration Console > Edit User dialog > Access Permissions tab. When you assign a class level permission, it is the *minimum* permission that the user has to *all objects of this type*. You can also assign class level permissions to a role from the Edit Role dialog > Access Permissions tab. Roles define permissions for set of users over sets of objects.

  When a user belong to one or more roles, the highest level of access between the roles and the user's assigned permissions is the one that applies. For detailed information about roles, see "Roles" on page 296.

By default, every new user has No Access permissions for everything (except user accounts, to which they have Read Only permission). However, even with this minimal set of permissions, a user may receive and view alert notifications and reportlets generated as the result of mandatory subscriptions.

> **Note:** A System User always has full permissions to every object in the installation. For information about the System user, see "Users" on page 332.

The rest of this discussion describes the permissions in detail:

- "Permission Restrictions" on page 259 details the rules for when you are allowed to change permissions.
- "Permission Inheritance and Dependencies" on page 259 describes the permissions you need to create and edit classes, and how some objects affect access to others.

Accessing Permissions

Access permissions specify the level of access a user has to an object. Permissions can be assigned to an entire class of object or to a specific object. The access permissions are:

Permission	**On the class**	**On a specific object**
No Access	Cannot see any objects of this class, unless granted "read" on specific objects.	—
Filtered/Read-Only	—	Limits the rows in a view that the user can see based on an access filter. See "Access Filters" on page 15 for details.
Read-Only	Can see all objects of the class.	Can see the object.
Read-Write	Can see and edit all objects of the class.	Can see and edit the object.

Note: You cannot assign a permission to a specific object that is more restrictive than the user's permission on the class. For example, you cannot assign Read Only when a user has Read-Write on the object's class. Further, you cannot assign a permission to an object that is greater than your own for the same object.

Class Level Access Permissions

All class level permissions are assigned to a user or role. To see or change a class level permission, you must first edit the user's account or role definition.

Note: When viewing the list of user or roles, *do not use* the Permissions button. That button defines which users and roles may access the specific user accounts or role definitions in the list. See "Specific Object Access Permissions" on page 256 for details about this button.

➤ **To change a user's permissions for a class of objects:**

1. In the Administration Console, select the Users folder, and double-click on the user's account in the list. This edit's the user's account.
2. On the Edit User dialog, click the Access Permissions tab.
3. Click Edit next to the permission to change. The permission dialog has three fields:
   - Role-Granted Permissions shows the permission assigned by the roles that the user is a member of. When the user belongs to multiple roles, the greatest level of access among them is applied.
   - Additional User-Specific Permission is the permission that you are assigning for the user for the class. While you can assign a permission lower than the role permissions, doing so does not lower the users permission.
   - Effective Permissions is the greatest level of permission assigned by the other two fields, and is *the permission assigned* to the user for this class of objects.

   For more information about this dialog, see "Access Permissions Tab" on page 335.
4. Save the changes to immediately apply them to the user.

➤ To change a role's permissions for a class of objects:

1. In the Administration Console, select the Roles folder, and double-click on the role in the list. This edit's the role's definition
2. Click the Access Permissions tab.
3. Click Edit next to the permission to change. Set the permission to the class for this role.
4. Save the changes to immediately apply them to the role.

Specific Object Access Permissions

You access the permissions to specific objects by selecting the object in the list of objects, and clicking the Permissions button above the list. Note that this applies to the Users and Roles lists as well. You can assign access permissions to specific user accounts and roles in the same way you assign access to specific views or agents.

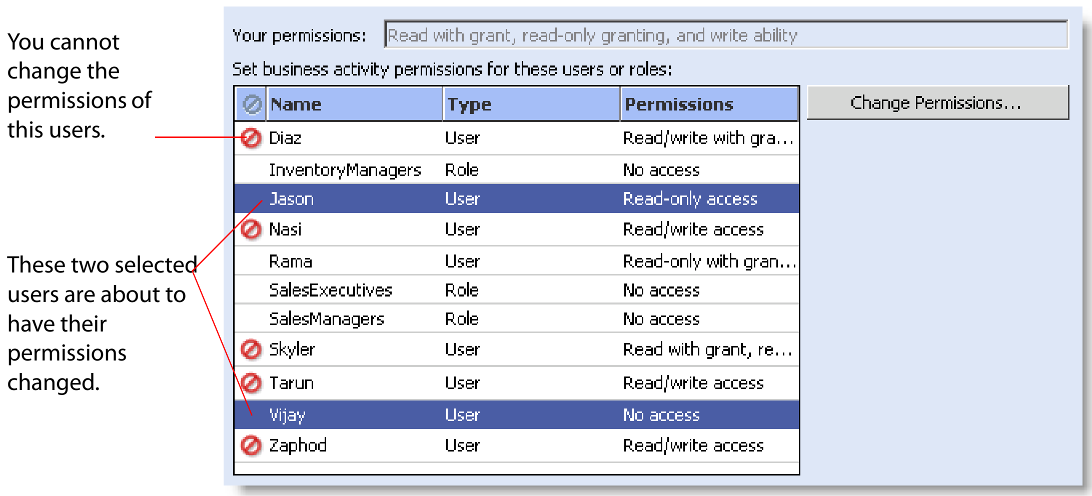
➤ To see user permissions for one or more specific objects:

- Select the objects in a list (such as a specific view in the Views list) and click Permissions. Your permissions to the object are shown at the top of the dialog, and the permissions that each user has to the object are listed below your permissions.

➤ To change user permissions for one or more specific objects:

1. Select one or more objects and click Permissions. (Note that Filtered/Read-Only permission can only be assigned to one view at a time.)
2. Select the users whose access permissions you want to change and click Change Permissions.

   You cannot change the permissions for users that do not meet the criteria listed in "Permission Restrictions" on page 259. For example, the following illustration shows a list of each user's access permissions to the current object, that three users have higher permission than the current user (because their permissions cannot be changed), and that two users are about to have their permissions changed.

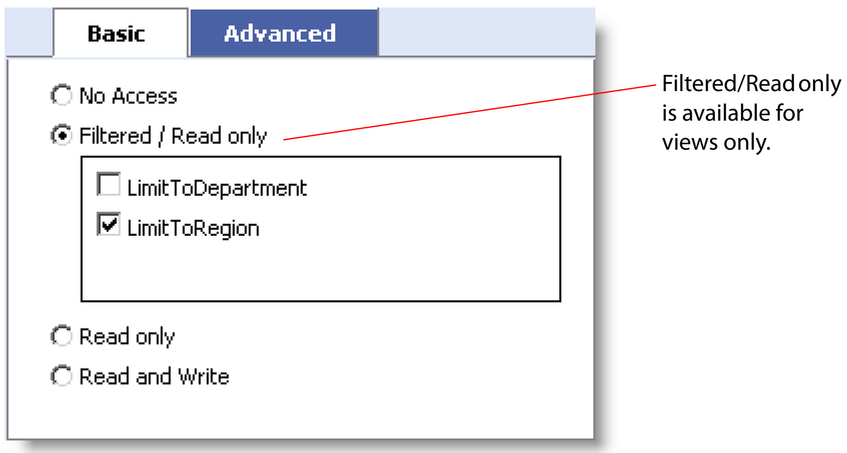

You cannot change the permissions of this users.
These two selected users are about to have their permissions changed.
Change Permissions…

Your permissions: Read with grant, read-only granting, and write ability

Set business activity permissions for these users or roles:

	Name	Type	Permissions
🚫	Diaz	User	Read/write with gra…
	InventoryManagers	Role	No access
	Jason	User	Read-only access
🚫	Nasi	User	Read/write access
	Rama	User	Read-only with gran…
	SalesExecutives	Role	No access
	SalesManagers	Role	No access
🚫	Skyler	User	Read with grant, re…
🚫	Tarun	User	Read/write access
	Vijay	User	No access
🚫	Zaphod	User	Read/write access

3. Set the permissions on the Basic tab. Note that when you assign permissions to a view, you have the option of choosing a Filtered/Read-Only permission, as shown in the following illustration. See "Access Filters" on page 15 for information about this permission.



No Access
[✓ Filtered / Read only]
LimitToDepartment
[✓ LimitToRegion]
Read only
Read and Write

Save the permissions and they are immediately applied to the objects.

Creating Permission

The Create permissions specifies which classes of objects a user may create.

When you create an object, you have Read and Write, and Grant Access permissions to that object. This allows you to grant any Access or Grant permissions to any other users for that object.

Note: Once you have created an object, any other user with Grant permissions on the object can reassign permissions, in effect overriding any permissions you assigned.

When you create a user, the user has "No Access" to everything. You may assign permissions to that user for an entire class of objects on the user's Access Permissions Tab.

➤ **To see which types of objects you may create:**

Click Account Settings and view the Access Permissions Tab for your account.

Granting Permissions

The Grant permissions allow you to assign permissions to other users. The Grant permissions are:

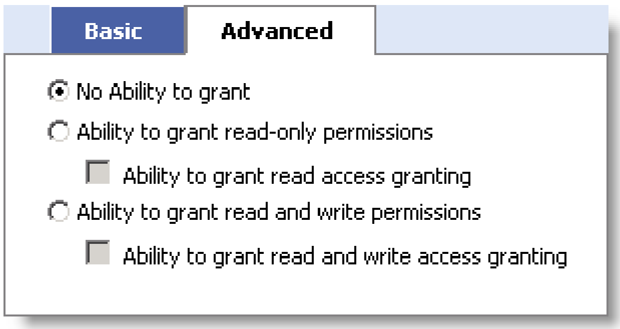
Permission	Description
Grant Read Only	May grant Read Only permission.
Grant Read and Write	May grant Read and Write permission.
Grant Create	May grant Create permission.
Grant ability to grant Read Only	May grant ability to grant Read Only permission.
Grant ability to grant Read and Write	May grant ability to grant Read and Write permission.
Grant ability to grant Create	May grant ability to grant Create permission.

Note: The System User is the only user that is always guaranteed to have full permissions on all objects.

➤ To grant permissions to other users:

1. Select the objects and click Permissions.

2. Select the users to modify and click Change Permissions and choose the Grant permissions on the Advanced tab.



(●) No Ability to grant
( ) Ability to grant read-only permissions
[ ] Ability to grant read access granting
( ) Ability to grant read and write permissions
[ ] Ability to grant read and write access granting

Permission Restrictions

When granting permissions, be aware of these restrictions:

- You cannot lower another user's permission on an object for which they have higher permission than you.
- You cannot raise another user's permission on an object to be higher than your own permission on that object. You will usually encounter this restriction when attempting to assign permissions on multiple objects are once, where your permission on one of the objects is less than your permission for the others.

Permission Inheritance and Dependencies

Objects that track permissions control access to the object, and to objects they may contain. Other objects inherit their permissions from the object that they are contained in. The objects that control permissions are:

- Agents
- Business Activities (controls access to contained scenarios, rules, alerts, and reportlets)
- Roles
- Tables, includes Events, Contexts, and consolidated events
- Users (controls access to user accounts).
- Views

Note: Regardless of what permissions a user has to an alert or reportlet, the user can always see the information in alert notifications and reportlets sent to the user.

Dependencies

When you create or edit objects, you are limited by the permissions of any dependant objects and by the permissions of any containing object. The following are the objects that have dependant requirements for create or edit:

- Table — You must have Read permission to the agent that feeds the table.

- View — You must have Read permission to the event table or base view and any context tables that feed the view.
- Business activity — To see the definitions of any objects contained in a business activity, you need Read permission on that business activity. Further, for contained scenarios, rules, alerts, and reportlets you need:
  - To see the definition of the object you need Read permission on the underlying view.
  - To create a contained object, you need Read and Write permission on the business activity, *and* Read permission on the underlying view.

Note: Permissions on one object can affect a user's access to another, especially for restrictions on views. For example, you might be able to edit an alert but not the alert's reportlet when the reportlet's view is No Access, but the alert's view is Read Only.

21 Portal Server Integration

This chapter describes the requirements and procedures for integrating BAM Server with one of the supported portal servers, in accordance with the JSR-168 specification.

➤ **In this Chapter:**

Before You Start

The following prerequisites must be in place before you begin this procedure:

- BAM Server must be started and running on any of the supported application server implementations.
- One of the following supported portal servers must be installed and running:

Portal Server	Description
BEA WebLogic Portal Server	The version bundled with WebLogic Application Server v 8.1 SP5. **Note:** Be sure to also install the bundled WebLogic Workshop, as this will be required to complete several steps in the implementation.
IBM WebSphere Portal Server	Version 5.1.0.1 This product is bundled with WebSphere Application Server version 5.1.13. **Note:** (DB2 only) If you are using WebSphere Application Server with a DB2 database, you also require the DB2 Universal V 8.1 or 8.2 fix pack 10.
JBoss Portal Server	Version 2.2.1 SP3 This product is bundled with the JBoss 4.0.3 SP1 Application Server

Integrating with BEA WebLogic Portal Server

This section contains the following subsections:

- Converting the lavaJSR168.war
- Deploying the lavaJSR168.war into WebLogic Portal Server

Converting the lavaJSR168.war

To integrate with WebLogic Portal Server, you must first convert the lavaJSR168.war to the WebLogic Portal specification using the Portlet Preparer Tool utility available from BEA (*http://dev2dev.bea.com/codelibrary/code/portletpreparer.jsp*).

➤ **To convert the lavaJSR168.war file:**

1. From the Portlet Preparer Tool utility, extract the portletConverter.jar to a temporary directory; that is, $CONVERTER_ROOT$.
2. Copy the lavaJSR168.war file to the same directory.
3. In the Portlet Preparer Tool utility, set WEBLOGIC_HOME in env.sh or env.bat (depending upon the platform) to the same temporary directory.
4. Run the following command:
   ```
   run ant -Dwar.file=lavaJSR168.war
   ```
   This creates a subdirectory called tempDir.
5. Using WebLogic Workshop, import the tempDir as a portal Web project.

Deploying the lavaJSR168.war into WebLogic Portal Server

After converting the lavaJSR168.war file, you can deploy it.

➤ **To deploy the lavaJSR168.war file:**

1. Install WebLogic Portal Server, if it is not installed already.
2. Create a new portal domain instance and start the new portal domain server.
3. Using WebLogic Workshop, create a new Portal Application.
4. Into the new portal application, import the tempDir portal web project you created in the previous section.

   Note: Before proceeding, ensure that the lavaJSR168.war file is visible under the WEB-INF/lib directory of the web project and that the cqjsr168.jar is visible under the portlet section.
5. Launch the Portal Administration Console and log in.

   Note: Before proceeding, ensure that the correct web application is selected in the console. Ensure also that the dashboard displays under portlets in the modules node.
6. Create a new Page and add the dashboard.
7. Create a new Portal.

8. Under the new portal create, a new desktop.
9. Under the new desktop, create a new blank book and add the page you created above.
10. View the desktop to launch the portal application.
11. In the dashboard portlet, go to Edit mode and log in to the running BAM Server.

    This action displays the available metrics.
12. Select the desired metrics to be displayed in the portal page.
13. Click **OK**.

    This action displays the dashboard objects that are available for the portlet.

Integrating with IBM WebSphere Portal Server

To perform the following procedure, you should access and review the WebSphere Portal documentation, which is available online at *http://www-128.ibm.com/developerworks/websphere/zones/portal/proddoc_v51x.html*.

➤ **To integrate with WebSphere Portal Server:**

1. Install WebSphere Portal Server, if it is not installed already.
2. Log in to the portal administration console.
3. In **Portal Management>Web Modules**, deploy the lavaJSR168.war file.
4. Create a new Virtual Portal Server with an appropriate name.
5. From the newly created Virtual Portal Server, create a new Page.
6. When configuring page layout:
    - Select a two-column layout.
    - Add the dashboard portlet.
7. When completed, the portal displays the page name as a link in the left column. When you click on the **Page** link, the right column displays the dashboard portlet.
8. Click on the **Page** link, and go to the Edit mode of the dashboard portlet.
9. Access BAM Server by entering the username, password, and URL of the running instance.

    This action displays the available metrics.
10. Select the desired metrics to be displayed in the portal page.
11. Click **OK**.

    This action displays the dashboard objects that are available for the portlet.

Integrating with JBoss Portal Server

➤ **To integrate with JBoss Portal Server:**

1. Install JBoss Portal Server, if it is not installed already.
2. Shut down the JBoss Application Server, if it is running.
3. Copy the lavaJSR168.war file to the deploy directory in the JBoss Application Server installation.
4. Restart the JBoss Application Server:

   …\$JBOSS_HOME$\bin\run.bat (Windows)

   …/$JBOSS_HOME$/bin/run.sh (UNIX)
5. Bring up the View page in a browser.

   For example, the URL may be http://[host]:[port]/portal/portal/default/[page name].

   Note: The page name value may be customized by modifying the **<page-name>** element in the celequest-objects.xml file in the WAR package. The default is *celequestpage*.
6. Go to Edit mode of the dashboard portlet.
7. Access BAM Server by entering the username, password, and URL of the running instance.

   This action displays the available metrics.
8. Select the desired metrics to be displayed in the portal page.
9. Click **OK**.

   This action displays the dashboard objects that are available for the portlet.

Creating Multiple Portlet Instances on a JBoss Portal Server

You can create multiple dashboard portlet instances on the JBoss portal server. To create multiple portal pages and have different sets of dashboards displayed on them execute the following procedures.

➤ **To create multiple instances of CelequestDashboardPortletInstance:**

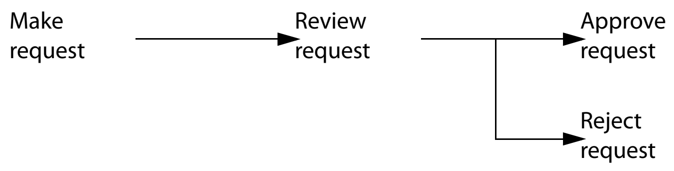
1. Login to JBoss Portal with the user name admin and password admin.
2. Choose the Admin Portal page on the Tab.
3. Click on the Manage Instances Link.
4. Select the LavaDashboardPortletInstance.
5. Give the instance a new name and click create New Instance.
6. Repeat the steps 4 and 5 to create multiple instances of LavaDashboardPortletInstance.

➤ **To create multiple pages:**

1. Click on Manage Portal Link.
2. Click on the root folder in the tree.
3. Click on the default page under the root tree.
4. Create a New Page (for example, celPage1) in the left frame then click on that page.
5. Now associate the new page with a LavaDashboardPortletInstance instance on the center of the page.
6. Click on Preview Link for page in step 5 and edit the dashboard portlet. For example, adding Dashboard objects the page.
7. Repeat steps 4, 5, and 6 to create dashboard portlets for each LavaDashboardPortletInstance instance.

22 Processes

A *process* is the set of steps (actions) that accomplish a task, such as the example below which is a four-step process for making a request for approval. Further, a real transaction through a process is a *process instance*, such as a specific request for approval.

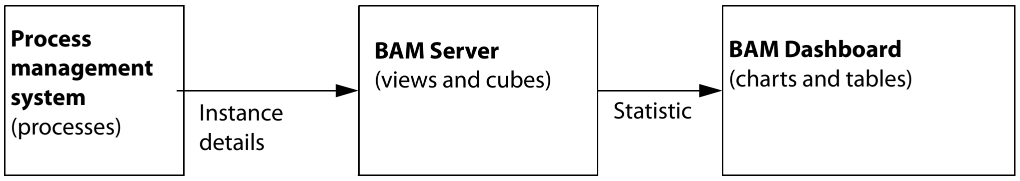


Business process management (BPM) systems create and manage business processes and instances. When a BPM is managing a process instance, it sends details about each step of the transaction to Business Activity Monitoring, which then develops statistics about the entire process. For example, the system might determine how long, on average, it takes to complete the entire process, is the process getting faster over time, what percentage of requests are rejected, or how long just the review step takes.

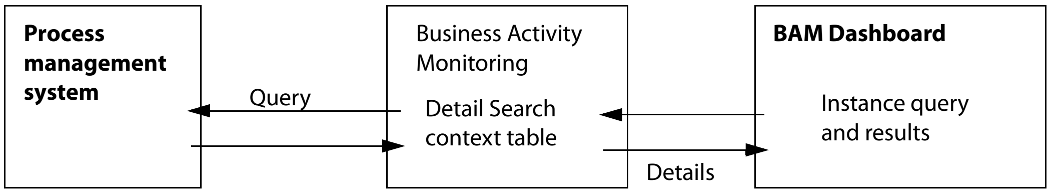
➤ In this Chapter:

How It Works

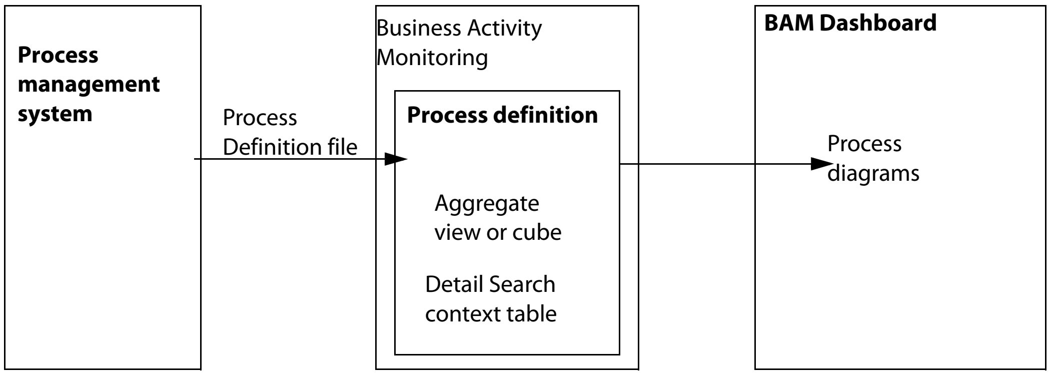
Process instance details arrive in Business Activity Monitoring as events. The BAM Server aggregates the details in views or cubes to generate the statistics. The BAM Dashboard then presents the process as a diagram, and includes the statistics in a table. Other objects can display other metrics, such as charts that present statistics over times.


Process management system (processes)
Instance details
BAM Server (views and cubes)
Statistic
BAM Dashboard (charts and tables)

The BAM Dashboard also provides an optional ad-hoc query to the BPM to get the details about a specific process instance, such as where it is in the process. When a user makes such a request, the BAM Dashboard queries a context table in the BAM Server, and that table queries the BPM for the specific instance details. The results are then returned to the BAM Dashboard for display.


Process management system
Query
Business Activity Monitoring
Detail Search context table
Details
BAM Dashboard
Instance query and results

The process management system generates a *process definition file* that describes the process in XML. A *process definition* in the BAM Workbench then associates that definition file with an aggregate view or cube and optionally with a search context table. The BAM Dashboard uses that object as the source for a process chart and to identify the associated aggregate view or cube and search context table.



Creating and Using Processes

To create and use processes in Business Activity Monitoring, the external business process system must:

- Generate a process definition file in an XML format recognized by Business Activity Monitoring.
- Publish process step statistics as event data into the BAM Server.
- Optionally provide an interface for the process instance queries from Business Activity Monitoring context tables.

➤ **The general steps for creating process diagrams are:**

1. Create an agent (if necessary) and event table to receive the process events from the BPM.
2. Create one view or cube per process.
3. (Optional) Create a context table to query the BPM.
4. Create a process definition object.
5. Create a process diagram.

Event Streams

The event streams receive and aggregate the process instance statistics. Event data are received in an event table, usually arrive through an agent, and the aggregate views are based on the event table. For detailed information about these components, see the following topics:

- "Agents" on page 23
- "Events" on page 76
- "Views" on page 350
- "Cubes" on page 50

The following requirements also apply:

- There should be only one agent and event table per BPM.
- There should be one view or cube per process. Use a Where Clause to distinguish the process events from other processes in the event table, such as "Process Name"='Request Approval'. For details, see "Working with Process Definitions" in *Using Business Activity Monitoring Workbench*.

Context Search Table

The context table generates a query to the BPM whenever a BAM Dashboard user makes an ad hoc query about a specific process instance. For details about context tables, see "Context" on page 43.

When the context receives a query from the BAM Dashboard, it first looks for the details in the context cache. If the instance is not in the cache, the table then queries the BPM for the details. Be sure to define a reasonable invalidation schedule for your business, or disable the cache if the queries need to return the most up-to-date information about the process instance.

Process Definitions

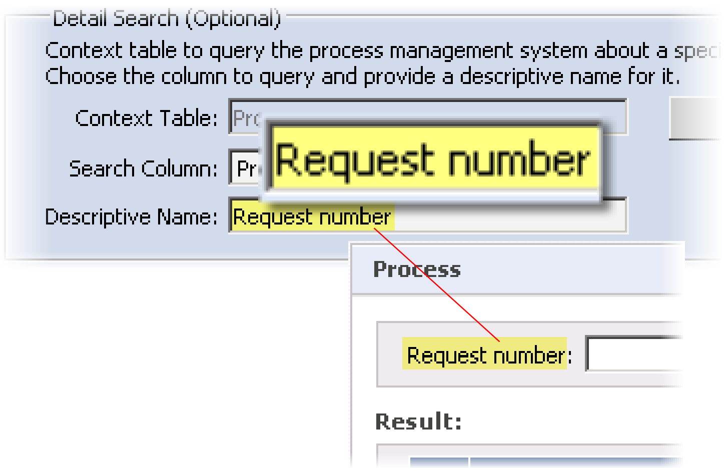
For details about this task, see "Working with Process Definitions" in *Using Business Activity Monitoring Workbench*.

Before creating a process definition, you need:

- Read Only access permission on the view or cube that aggregates the process events.
- (Optional) Read Only access permission on the context search table.

➤ To create a process definition in the BAM Workbench:

1. Open the Process Definitions folder in the BAM Workbench and choose New Process Definition.
2. Name the object and upload the process definition file generated by the BPM.
3. Choose the aggregate view or cube, and choose the column that identifies the steps in the process.
4. (Optional) Choose the detail search context table, choose the column to search, and optionally provide descriptive text to appear in the BAM Dashboard.



Detail Search (Optional)
Context table to query the process management system about a spec
Choose the column to query and provide a descriptive name for it.
Context Table: Pro
Search Column: Pr
Descriptive Name: Request number

Request number

Process
Request number:
Result:

5. Save the process definition.

You can now create process diagrams based on this definition.

Process Diagrams

For details about this task, see "Process Diagrams" in *Using Business Activity Monitoring Dashboard*.

Before creating a process diagram, you need Read Only access permission on the aggregate view or cube that provides the statistics.

➤ To create a process diagram in the BAM Dashboard:

1. Open the Process Diagrams manager and choose Create Diagram.
2. Select the process definition (as defined in the BAM Workbench).
3. Select the source columns to present as statistics.
4. Name the process diagram and save it.

The BAM Dashboard immediately presents the process diagram. The statistics update as events arrive for the specific process.

23 Query Windows

A *query window* specifies a set of rows that are used in calculations with respect to the current row (event) under examination in a view constructed with a C-SQL SELECT statement. (Business views created in the BAM Workbench construct views through a well-formed SELECT statement passed to the servers.) This chapter discusses and describes how to define and use windows.

➤ In this Chapter:

Overview

The calculation using a window may be for computing a moving set function, a join, or expiring rows from a view. All such calculations use a window; however, in the default cases for these operations, you do not need to define the window semantics. For example, in C-SQL, *set functions* perform calculations on sets of rows in a view. The default set of rows for each function is *all events since the view was initiated* (an “unbounded” set). Consider the following view that calculates the total value of a column named Qty for all events that were ever included in the view:

```
SELECT SUM(Qty) AS Total_Qty FROM Orders
```

However, another way to express the SUM() in order to get the same result is the following, which says to *sum over the set of all previous events*:

```
SELECT SUM(Qty) OVER (EVENTS UNBOUNDED PRECEDING) AS Total_Qty
    FROM Orders
```

The OVER clause defines a window that identifies the set of rows to include. With a window, you can limit the set to a specific count of event rows or to those events that occurred within a specific time-span. For example, *to total just the current and last five events*, define a window as follows:

```
SUM(Qty) OVER (EVENTS 5 PRECEDING) AS Total_Qty
```

And, to total *just the current month’s events*, use a time-series window, as follows:

```
SUM(Qty) (RANGE INTERVAL '1' MONTH PRECEDING) AS Total_Qty OVER
```

Window Types

All windows are identified by either the EVENTS or RANGE clause and include an *extent* definition that defines the size of the window.

```
{ RANGE | EVENTS } <window frame extent>
```

The extent syntax is unique to the window type and is described in detail in “Event-Series Windows” on page 276 and “Time-Series Windows” on page 277. Other clauses (not shown) control how the window behaves as new events enter the window, which items to include, and how and when the window updates to include new events and discard old ones.

Window Declarations and References

There are two ways to define windows and associate them with functions:

In-Line

In-line defines the window parameters immediately following the function reference, similar to the following:

```
SELECT PartName, SUM(Qty) OVER (EVENTS 4 PRECEDING) AS Total_Qty,
   FROM Orders
   GROUP BY PartName
```

This format is useful when you have only one window per query, though you can also use it with multiple windows. Note that you cannot share these window definitions among functions in the same query.

Reference by Name

Reference by name to use a window defined with the WINDOW clause, similar to the following:

```
SELECT PartName,
       SUM(Qty) OVER Previous4 AS Total_Qty,
       AVG(Qty) OVER Previous4 AS Average_Qty
   FROM Orders
   GROUP BY PartName
   WINDOW Previous4 AS (EVENTS 4 PRECEDING)
```

This format is useful when you have multiple simple window definitions because you can define them all in one place: in the same WINDOW clause definition. This form also allows you to share the definition over multiple functions in the same query (as shown above) and allows you to use windows that extend another window's definition (see "Extending One Window Definition with Another" on page 275).

Note: Functions in the select list associated with a window must have alias names defined with the AS operator, such as AS Total_Qty in the examples above.

Multiple Windows Per Query

The WINDOW clause defines windows that can be shared throughout the query, and which may be extended by other windows. The following example defines two windows, each used by a different function in the query:

```
SELECT PartName,
       SUM(Qty) OVER Previous4 AS Total_Qty_4,
       AVG(Qty) OVER Previous10 AS Average_Qty_10
   FROM Orders
   GROUP BY PartName
   WINDOW Previous4 AS (EVENTS 4 PRECEDING),
          Previous10 AS (EVENTS 10 PRECEDING)
```

Extending One Window Definition with Another

When windows share common traits, you can define those traits in one window definition, then extend (inherit) that definition with other, unique aspects in different windows. For example, the following definition defines one window named Common with the PARTITION BY clause, then defines additional windows that extend the common traits with the range required for the unique windows:

```
SELECT PartName,
       SUM(Qty) OVER Events4 AS Total_Of_Qty_4,
       AVG(Qty) OVER Events10 AS Average_Of_Qty_10
   FROM Orders
   WINDOW Common   AS (PARTITION BY PartName),
          Events4  AS (Common EVENTS 4 PRECEDING),
          Events10 AS (Common EVENTS 10 PRECEDING)
```

The WINDOW definition above is the same as:

```
Events4  AS (PARTITION BY PartName EVENTS 4 PRECEDING),
Events10 AS (PARTITION BY PartName EVENTS 10 PRECEDING)
```

Restrictions

Window extension definitions may not include properties defined in the base window. For example, you cannot define an ORDER BY in both the base and extension windows. Further:

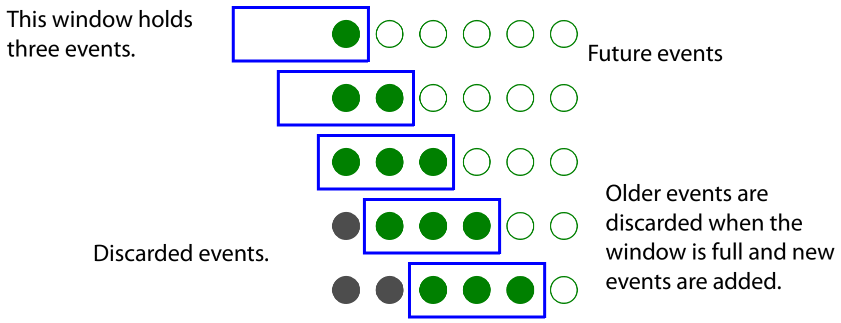
- A PARTITION BY clause can appear *in the base window definition only*; it cannot appear in extensions.
- These clauses can appear *in extension definitions only*; they cannot appear in the base window:

  ```
  BETWEEN/AND
  CURRENT EVENT
  EVENTS
  INITIALIZE
  RANGE
  REFERENCE
  SLIDE
  UNBOUNDED
  <window start integer> without an EVENTS or RANGE clause
  ```

- A window may only extend another window defined in the same query; a window in a derived view cannot extend a window in a base view.

Event-Series Windows

Event-series windows contain a maximum fixed-set of events. Initially the window is empty, but fills with new events until it reaches its defined capacity. After that, the oldest events are discarded one-for-one as the newest events are included.


This window holds three events.
Future events
Older events are discarded when the window is full and new events are added.
Discarded events.

EVENTS Clause

Event-series windows are identified by the EVENTS clause. There are two ways to express spans of events. The first method is to use the BETWEEN and AND clauses. which specify both the upper and lower boundary of the window. The second method is to use the PRECEDING expression, which specifies only the lower boundary of the window. The upper boundary is the current event in the second case. The PRECEDING expression clarifies that the event rows precede the current one.

```
([PARTITION BY <column>]
 EVENTS  { BETWEEN {<oldestEvent> | UNBOUNDED} PRECEDING
              AND { <newestEvent> PRECEDING | CURRENT EVENT }
           |{<oldestEvent> | UNBOUNDED} PRECEDING
          }
 [SLIDE <distance>]
 [REFERENCE {FRAME | OPERATOR} ]
)
```

PARTITION BY creates one window frame for each <column> of events, similar to a GROUP BY window. See "Window Partitions" on page 282 for details.

SLIDE identifies how to advance the window when new events arrive in the view. See "Window Advancement" on page 285 for a detailed description of this option.

REFERENCE tells the window when to determine if rows have expired from the window set. See "Window Update Reference" on page 288 for a detailed description of this option.

Examples

This window contains rows limited by a count of consecutive events in the view, such as the last 5 events,

```
SUM(Qty) OVER (EVENTS 4 PRECEDING) AS Total_Qty
```

or the 10 events starting 12 events ago.

```
SUM(Qty) OVER (EVENTS BETWEEN 11 PRECEDING AND 2 PRECEDING)
        AS Total_Qty
```

Notice that the size of the window frame is (<oldestEvent>–<newestEvent>+1). For example, the frame above contains 10 events (11–2+1).

To include all of the previous events, including the current one, with the UNBOUNDED option, like this:

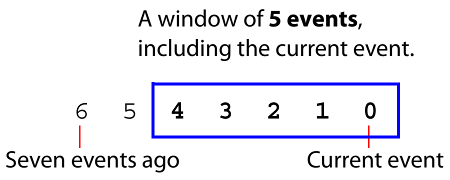
```
SUM(Qty) OVER (EVENTS UNBOUNDED PRECEDING) AS Total_Qty
```

Which is the same behavior as if no window was defined:

```
SUM(Qty) AS Total_Qty
```

Current Event

In the examples in the preceding section, notice that the starting event is numbered 1 less than the desired starting event. This is because the counting is zero-based: *event zero (0) is the current event*.



Another way to express the last 5 events is:

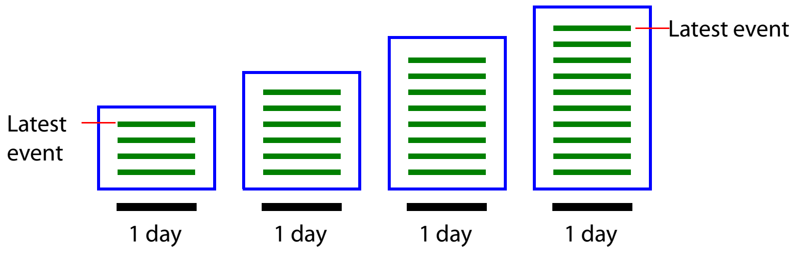
```
OVER (EVENTS BETWEEN 4 PRECEDING AND 0 PRECEDING)
```

Yet another way to express the range of events is to use the CURRENT EVENT literal as follows:

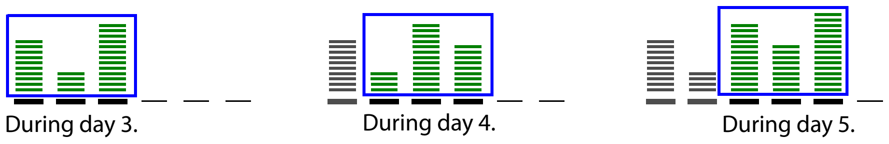
```
OVER (EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT )
```

Time-Series Windows

Time-series windows grow to include all of the events that occur within an interval of time. Such as the 1-day window shown in the illustration below that grows as new events arrive during the day.



Similarly, a 3-day window includes all of the events within the current 3-day window.



RANGE Clause

Time-series windows are identified by the RANGE clause. These windows contain rows that are limited to a time range in combinations of years, months, days, hours, minutes, or seconds. There are two ways to express spans of events. The first method is to use the BETWEEN and AND clauses. which specify both the upper and lower boundary of the window. The second method is to use the PRECEDING expression, which specifies only the lower boundary of the window. The upper boundary is the latest timestamp in the second case.

```
([PARTITION BY <column>]
 [ORDER BY {<date-time column> | <integer column>} [ ASC | DESC ] ]
 RANGE {BETWEEN
            {INTERVAL <oldestTime> | <oldestInt> | UNBOUNDED} PRECEDING
             AND {INTERVAL <newestTime> | <oldestInt>} PRECEDING
        |{INTERVAL <oldestTime> | <oldestInt> | UNBOUNDED} PRECEDING
       }
 [SLIDE [INTERVAL <distance>]]
 [REFERENCE {FRAME | OPERATOR} ]
 [INITIALIZE <initTimestamp>]
)
```

Further,

- The order-by, "oldest", and "newest" columns are usually date-time data types. However, you can also use an integer that represents a time-series. See "Integer Time-Series" on page 281 for details.
- PARTITION creates one window for each <*column*> of events, similar to a GROUP BY window. See "Window Partitions" on page 282 for details.
- ORDER BY identifies the column used to calculate the time of the event. See "ORDER BY Clause" on page 279 for details.
- SLIDE identifies how to advance the window when new events arrive in the view. See "Window Advancement" on page 285 for a detailed description of this option.
- INITIALIZE specifies a common date-time to which to initialize all associated windows in a view. When you use a time-series window it is best to initialize the start time to be midnight for day, month, and year windows; to the first day of the month for month and year windows; and to the first day of the desired range for year windows. See "Window Initialization" on page 288.
- REFERENCE tells the window when to determine if rows have expired from the window set. See "Window Update Reference" on page 288 for a detailed description of this option.
- ASC specifies that the rows are ordered according to their timestamps from oldest to newest. (For further explanation of ordering, see "Descending" on page 280.)
- DESC specifies that timestamps are ordered according to their timestamps from newest to oldest. (For further explanation of ordering, see "Descending" on page 280.)

Examples

This window totals the Qty column for the current month's worth of events:

```
SUM(Qty) OVER (RANGE INTERVAL '1' MONTH PRECEDING) AS Total_Qty
```

You can also identify very specific ranges, such as this one which starts18 hours and 15 minutes ago, and stops 45 seconds ago: See “Date-Time” on page 61 for detail about the date-time specifications.

```
OVER (RANGE BETWEEN INTERVAL '18:15' HOUR TO MINUTE PRECEDING
      AND INTERVAL '45' SECOND PRECEDING
)
```

Which Events Are Included?

The events to include in a time-series window are determined when a new event enters the window or view (see “Window Update Reference” on page 288 for more details). However, it is important to note that interval is inclusive of events that are exactly the size of the range interval from the current event. For example, consider a window with a one day interval:

```
SUM(order_qty) OVER ( RANGE INTERVAL '1' DAY  PRECEDING ) AS TotalQty
```

When two events have *exactly* one day between them, they are both included in the window. Notice the value of TotalQty after the two events:

```
order_qty TotalQty EventTime
--------- -------- ---------------------
        1        1 2003-12-01 09:00:00.0
        1        2 2003-12-02 09:00:00.0
```

When using the BETWEEN clause, both <oldestTime> and <newestTime> are included. For example, to have two windows, one of the current week and one for the week before that, use these definitions:

```
ThisWeek:  RANGE INTERVAL '7' DAY PRECEDING
LastWeek:  RANGE BETWEEN INTERVAL '14' DAY PRECEDING
                 AND INTERVAL '7' DAY PRECEDING
```

Notice that both ranges use '7' as a bounding value. The current week includes everything from now back seven days inclusive, while the previous week includes the seven days *before* seven days ago. Another way to define the windows above is to use '0' as the current time; for example:

```
ThisWeek:  RANGE INTERVAL '7' DAY PRECEDING
                 AND INTERVAL '0' DAY PRECEDING
LastWeek:  RANGE BETWEEN INTERVAL '14' DAY PRECEDING
                 AND INTERVAL '7' DAY PRECEDING
```

Note that the row at percisly 7 days ago would be included in both this week and last week.

ORDER BY Clause

Unless defined otherwise, all times are calculated based on each event’s internal event arrival timestamp. For example, this window contains events of the last hour in the order that they arrived in the system.

```
OVER (RANGE INTERVAL '1' HOUR PRECEDING)
```

You can designate any date-time column in the event as key. For example, you might want to use the time that an order was placed:

```
OVER (ORDER BY order.order_timestamp RANGE '4' DAY PRECEDING)
```

The ORDER BY argument is a *single column name reference*; you cannot use integers to represent the ordinal position of the column in the SELECT list. Further, the default sort order is ascending (ASC), though descending (DESC) is available also. See "Descending" on page 280 for details. The ORDER BY clause has the following syntax:

```
ORDER BY <columnNameReference> [ ASC | DESC ]
```

Out-of-Order Arrival

When the event stream is not in the expected order, the query engine attempts to insert the out-of-order event into its correct location in the window frame, and updates all aggregations accordingly. The query engine always uses the latest time of all the events received prior to the out-of-order event to determine if it should be included. If the event is not within the latest window frame, it is omitted.

For example, consider a window of 1-hour that receives the following events in the order listed. In the example, the second event is the latest, and as such, only events received after 08:10 (09:10 minus 1 hour) are included in the window. The fourth event, therefore, is rejected because its timestamp is 08:04.

```
OVER (ORDER BY Time RANGE INTERVAL '1' HOUR PRECEDING)

Arrival
Sequence  Time   Amount
--------  -----  ------
 1        08:45  242.69
 2        09:10  103.76  << Latest timestamp sets the window
 3        08:50   90.20  << Out-of-order, accepted
 4        08:04  188.88  << Out-of-order, rejected
```

After the four events above have been processed, the view that contains them looks like this:

```
Arrival
Sequence  Time   Amount
--------  -----  ------
 1        08:45  242.69
 3        08:50   90.20  << Reordered
 2        09:10  103.76
```

Descending

By default, order is assumed to be ascending: oldest events are processed first. However, you may specify DESC for descending sort order. When events arrive and they are not already in descending order, they are processed the same as noted above in "Out-of-Order Arrival."

```
OVER (ORDER BY Time DESC RANGE INTERVAL '1' HOUR PRECEDING)

Arrival
Sequence  Time   Amount
--------  -----  ------
 1        08:45  242.69
 2        09:10  103.76  << Out-of-order, expires after fourth event
 3        08:50   90.20  << Out-of-order, accepted
 4        08:04  188.88
```

After the four events above have been processed, the view that contains them looks like this:

```
Arrival
Sequence  Time   Amount
--------  -----  ------
 4        08:04  188.88
 1        08:45  242.69
 3        08:50   90.20
```

NULL Value Timestamps

When the referenced column contains a NULL value for the timestamp, the event is rejected *unless* the range is UNBOUNDED. When the window frame is unbounded, all events are included, including the NULL timestamps; though the NULL values are placed last in the set, in the order they arrived. For example:

```
Arrival
Sequence  Time   Amount
--------  -----  ------
 1        08:04  188.88
 2        08:45  242.69
 3               103.76  << First NULL time
 4        08:50   90.20
 5               157.11  << Second NULL time
```

When the above events are processed in ascending order by Time column, the resulting view looks like:

```
Arrival
Sequence  Time   Amount
--------  -----  ------
 1        08:04  188.88
 2        08:45  242.69
 4        08:50   90.20
 5               157.11  << Second NULL time
 3               103.76  << First NULL time
```

Integer Time-Series

A time-series range is usually expressed as a range of date-time or date interval values. However, you can also use an integer that represents a time-series. For example, consider this series of date-time values and matching integer values:

```
Date_time_value      Date_time_int
-------------------  -------------
2003-12-01 09:00:00     3795637500
2003-12-01 13:00:00     3795654167
2003-12-02 09:00:00     3795737500
2003-12-03 09:01:00     3795837569
2003-12-04 09:00:00     3795937500
```

The “time” that the integers represent is entirely arbitrary and not a factor in how Business Activity Monitoring processes the values. Rather, it is up to you to understand what the values mean. For example, in the series above, 100,000 represents one day. As such, a window of the previous two days for this data is defined as:

```
ORDER BY Date_time_int
RANGE 200000 PRECEDING
```

Or to see just the previous day:

```
ORDER BY Date_time_int
RANGE BETWEEN 200000 PRECEDING AND 100000 PRECEDING
```

When using an integer time-series, always use the ORDER BY clause and identify the integer column as the series order.

Window Partitions

All events in a view are included in a single set (window) unless defined otherwise by the GROUP BY or the PARTITION BY clause. These clauses sort events into windows based on a key value, such as a common name or ID. Use partitioned windows to aggregate the events specific to the window. For example, to collect the total volume for all transactions by security, the view definition might look like the following:

```
SELECT Trades.symbol, SUM(Trades.volume) OVER Symbols AS Total_volume
   FROM Trades
   WINDOW Symbols AS ( PARTITION BY Trades.symbol )
```

This is similar to a view defined with the GROUP BY clause in the following manner:

```
SELECT Trades.symbol, SUM(Trades.volume) AS Total_volume
   FROM Trades
   GROUP BY Trades.symbol
```

Note: Querying the two views above produces very different results. The grouped view returns one row for each group. While the partitioned view, by default, returns just one row containing the result of the last event that entered the view (though the information for each partition is maintained internally). To see more rows from a partitioned view, set the view's Maintain in view setting to a size greater than 1. For more information, see "Historical Results from Partitioned Views" on page 284.

PARTITION BY Clause

The PARTITION BY clause defines one or more columns that contain the values that identify a partition window.

```
PARTITION BY <column> [, <column> … ]
```

Where <column> is either the name of a column in the SELECT list, or is an ordinal integer that represents the position of a column listed in the SELECT list of columns (the first reference is 1). When you declare a list of columns, one partition is created for each unique value of the set. For example, this declaration creates partitions for individuals based on unique lastname+firstname combinations:

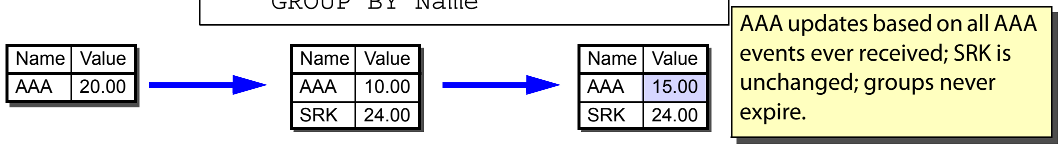
```
PARTITION BY last_name,first_name
```

Consider the illustrations in the following sections that show what happens when a new event arrives that is significantly newer than the most recent event already in the view.

View Update for a Simple GROUP BY

When an event arrives in a view with a simple GROUP BY clause, the new event is applied to the associated group. In this case, the new event is applied to the average for all AAA events ever received:

```
SELECT NAME, AVG(Value) AS AvVal
    GROUP BY Name
```



Name	Value
AAA	20.00

Name	Value
AAA	10.00
SRK	24.00

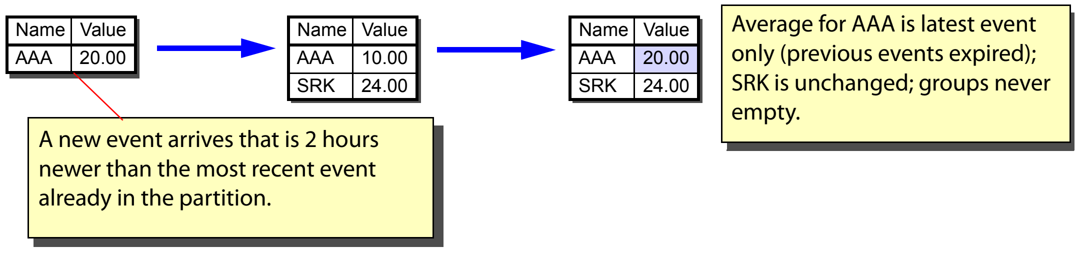
Name	Value
AAA	15.00
SRK	24.00

AAA updates based on all AAA events ever received; SRK is unchanged; groups never expire.

View Update for a Partition with Frame Reference

Now consider the same event entering a view partitioned by Name, and where only the partition window frame that identifies the event updates. In this example, the AAA event is 2 hours newer than the last AAA event. As such, all previous values for the partition expire and are discarded, and only the new event is used. Notice that the other partition is no affected.

```
SELECT Name, AVG(Value) OVER w AS AvVal
  WINDOW w AS (PARTITION BY Name
      RANGE INTERVAL '1' HOUR PRECEDING
      REFERENCE FRAME)
```



Name	Value
AAA	20.00

Name	Value
AAA	10.00
SRK	24.00

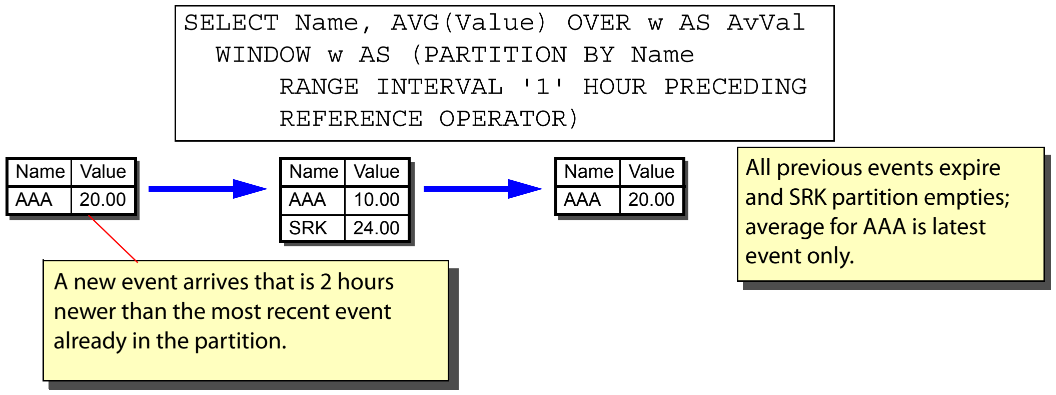
Name	Value
AAA	20.00
SRK	24.00

Average for AAA is latest event only (previous events expired); SRK is unchanged; groups never empty.

A new event arrives that is 2 hours newer than the most recent event already in the partition.

View Update for a Partition with Operator Reference

Finally, consider the same event entering a partitioned view that updates based on REFERENCE OPERATOR. The reference tells all partitions to update when an new event enters the window. In this illustration, all existing partitions expire and a new one is created for the new event because none of the events tracked by the existing partitions are within the range of the last hour.


SELECT Name, AVG(Value) OVER w AS AvVal
WINDOW w AS (PARTITION BY Name
RANGE INTERVAL '1' HOUR PRECEDING
REFERENCE OPERATOR)
Name Value
AAA 20.00
Name Value
AAA 10.00
SRK 24.00
Name Value
AAA 20.00
All previous events expire and SRK partition empties; average for AAA is latest event only.
A new event arrives that is 2 hours newer than the most recent event already in the partition.

The following query is similar to the one shown in the illustration above, except that in the illustration above, only the row corresponding to the last event entered the view is shown in the result:

```
SELECT Name, MOV_AVG(Value,HOUR,1)GROUP BY Name
```

Advantage of Partitions over Groups

The main advantage of partitions is that you can have multiple partitions based on different columns in the same view, while GROUP BY applies solely to the entire view.

Using Windows to Expire GROUP BY

One disadvantage of partitons is that you cannot look at the view to see the contents of the partitions; unlike a view with GROUP BY where you can view the results of all groups in the BAM Dashboard or in the Results tab of the BAM Workbench. A GROUP BY maintains results for each group *as long as there are data in the group*. In the previous illustration, if you use the view constructed with the GROUP BY expression instead of the PARTITION, you will be able to view the contents, and groups will expire when they have had no events in the last hour.

Historical Results from Partitioned Views

When you query a partitioned view, by default, the result is a view with one row containing the result of the last event that entered the view (though the information for each partition is maintained internally). For example, if you track the average price of securities, partitioned by symbol, querying SELECT * on the view would return a result similar to the following:

```
SELECT * FROM AveragePricesView

Symbol  AvgPrice  Date
------  --------  ----------
JMH     164.35    2003-07-14
```

In the results above, the last event that the *AveragePricesView* received was for the JMH symbol.

To see more rows, set the partitioned view's Maintain in view setting to a size greater than 1. If you have many aggregate events, set the value to a large number, such as 500. Thus, querying the view returns up to that many rows, for example:

```
Symbol  AvgPrice  Date
------  --------  ----------
AAA      24.35    2003-03-05
SRKH    102.07    2003-03-05
JMH      90.22    2003-03-05
SRKH    106.88    2003-03-06
AAA      25.66    2003-03-06
JMH      94.11    2003-03-06
...
```

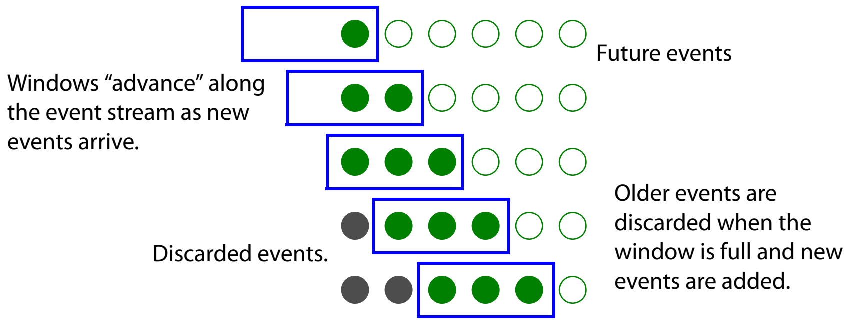
The results appear in the order that the view produced them: the order of the events that last entered each partition.

To get meaningful historical results, order the new view, such as on Symbol and Date. For example:

```
SELECT * FROM AveragePricesView ORDER BY Symbol,"Date"

Symbol  AvgPrice  Date
------  --------  ----------
AAA      24.35    2003-03-05
AAA      25.66    2003-03-06
AAA      25.25    2003-03-07
AAA      24.92    2003-03-08
...
```

Window Advancement

When a new event enters a window, the window determines which events to keep and which to discard when the window is full. A window frame is full if the next row causes an existing row in the frame to expire out of the window. When viewed in the context of future and past events in the event stream, the window can be seen to *advance* or *slide* along the event stream as it adds and discards events.


Future events
Windows "advance" along the event stream as new events arrive.
Older events are discarded when the window is full and new events are added.
Discarded events.

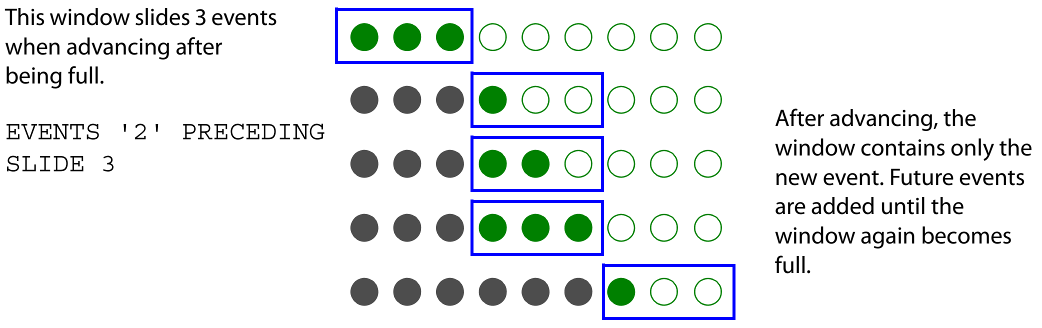
SLIDE Clause

The window advance clause (SLIDE) specifies the distance to advance when the window is full. By default, when SLIDE is omitted, event-series windows slide one event along the stream for each new event, while time-series windows advance to include the latest event and all events within the interval defined in the RANGE clause remain. The rest are discarded.

Including a SLIDE clause advances the window either the entire size of the window (when you omit the argument) or advances the event distance or time interval specified by the argument.

```
SLIDE [ <interval> | <distance> ]
```

For example, consider an event-series window whose size is 3 events. Declaring SLIDE with no arguments is the same effect as declaring SLIDE 3.

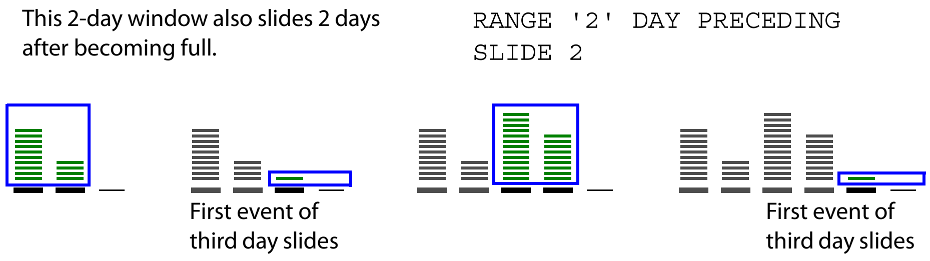

This window slides 3 events when advancing after being full.
EVENTS '2' PRECEDING
SLIDE 3
After advancing, the window contains only the new event. Future events are added until the window again becomes full.

When a window advances more than one event, it is *tumbling*, as described in the next section, "Tumbling Windows." Further, the TUMBLE_ functions are shorthand for complete window expressions that use this sliding behavior. See "Tumble Functions" on page 287 for more information.

Note: The SLIDE argument need not the same as the window size, but it mst be less than or equal to the window size.

Tumbling Windows

A *tumbling window* empties its contents when it advances to include the newest event. For example, a tumbling time-series window continues to grow until a new event enters that causes older events to be discarded.When the SLIDE interval is the same as the window size, the window dumps all existing events when a new one arrives and the window is full.


This 2-day window also slides 2 days after becoming full.
RANGE '2' DAY PRECEDING
SLIDE 2
First event of third day slides
First event of third day slides

This functionality is useful for tracking a full interval's worth of events during the interval. For example, if you start an interval on a Sunday and declare a slide interval of 7 days, the window will empty and advance every Sunday. Use the INITIALIZE clause to set the starting time appropriately. See "Window Initialization" on page 288 for details.

Trailing Tumbling Windows

A *trailing tumbling window* empties and begins refilling after it slides. However, it is important to understand that the items that enter the view must fall within the window as identified by the last event.

For example, consider this 7-day window that includes values from the previous week and empties every 7 days:

```
RANGE BETWEEN INTERVAL '14' DAY PRECEDING
      AND INTERVAL '7' DAY PRECEDING
SLIDE INTERVAL '7' DAY
```

This window only accepts values that are older than 7 days preceding the last event. Even though there might be a full week's worth of events, the window only contains those that are 7 days older than the last. For example, when these events are fed into the window, only the first event is included in the window because it is more than 7 days older than the last event:

```
Event_Time
-------------------
2003-12-01 09:00:00   << Only event included in the window
2003-12-01 10:00:00
2003-12-02 08:45:00
2003-12-08 09:10:00   << Last event
```

Each of the first three events is included only after receiving an event after 08:45 on the 12-09.

Tumble Functions

Most of the set functions have associated "tumble_" functions, which are shorthand for complete sliding window expressions. For example, consider this tumbling SUM() expression which sums all the events that arrive within a 1 hour interval:

```
TUMBLE_SUM(price, HOUR, 1, trade_time) AS T_Sum
```

The above function is shorthand for the following in-line window expression:

```
SUM(price) OVER (ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING SLIDE) AS T_Sum
```

Which in turn is equivalent to the following, after filling in all default values:

```
SUM(price) OVER ( ORDER BY trade_time
    RANGE INTERVAL '1' HOUR PRECEDING
    SLIDE INTERVAL '1' HOUR
    REFERENCE OPERATOR) AS T_Sum
```

Similarly, the function TUMBLE_SUM(price, EVENT, 5) is the shorthand for this complete window:

```
SUM(price) OVER (
    EVENTS BETWEEN 4 PRECEDING AND CURRENT EVENT
    SLIDE 5
    REFERENCE FRAME) AS T_Sum
```

See the descriptions of the individual tumble functions for details about their behavior.

Window Update Reference

When a view receives a new event, one or more of the view's windows may update to reflect the new information. Depending on each window's *reference* and definition, it is possible for all events in a window to expire and be removed from the window or to not be affected by the update.

Note: Events that are filtered out before they enter the view, such as when excluded by a WHERE Clause, do not affect the view's windows and do not cause the windows to update, regardless of the reference point.

REFERENCE Clause

A reference determines when to evaluate the set of events included in a window. The references are:

- OPERATOR — the window updates whenever a new event enters the view, whether or not the event is included in the window. For example, if a view tracks securities traded in the last hour, and partitions each security into its own window, each window evaluates its set whenever a new trade enters the view. If a security has not been traded in the last hour, its window becomes empty. All other windows include only those securities traded in the last hour; older trades are removed from their windows.

  ```
  SELECT Trades.symbol,
         AVG(Trades.price)
            OVER (RANGE INTERVAL '1' HOUR PRECEDING REFERENCE OPERATOR)
            AS av_price_last_hour
     GROUP BY symbol
     FROM Trades
  ```

  This mode is desirable — and the default — when using time-series aggregations and you want all windows to include events referenced from the same time: the time of the last event that arrived in the view. This mode is also useful for views where you want event-series windows to expire and thereby reduce memory consumption by the view.

- FRAME — the window updates only when a new event enters the window. For example, if the view tracks the last 10 trades for each security, only the window that receives the new trade updates. All other (security) windows retain their 10 event's worth of events.

  ```
  SELECT Trades.symbol,
         AVG(Trades.price) OVER (EVENTS 9 PRECEDING REFERENCE FRAME)
            AS av_price_last_10_trades
     GROUP BY symbol
     FROM Trades
  ```

  This mode is desirable when you want windows to retain a set of events, regardless of when they arrived, such as for event-based moving aggregates. This mode is the default for event based windows.

  Note: The FRAME reference is also useful for reclaiming server memory.

Window Initialization

When using a time-series window, the beginning time for the window frame is set by the first event that arrives in the window. When a view has several group or partition frames, each might have a different starting time. Consider these two events, which are the first to arrive in the view:

```
Symbol   Time
IBM      09:00:00.875
CQST     09:23:02.111
```

If the view that receives these events places them in different group-by frames, each will start at each event's Time and continue to reset based on that initialization time. For example:

```
SELECT Trades.Symbol, Trades.Time,
       AVG(Trades.Price) OVER An_Hour AS Avg_Price_One_Hour_Tumble
   GROUP BY Trades.Symbol, Trades.Time
   FROM Trades
   WINDOW An_Hour AS (ORDER BY Trades.Time
                     RANGE INTERVAL '1' HOUR PRECEDING SLIDE)
```

With this view definition, an event arriving at 09:10 will cause the initial IBM event to expire, but the CQST will remain in its window for at least another 13:02 minutes: the time remaining since it entered the view.

INITIALIZE Clause

To have all windows begin at the same time, use the INITIALIZE clause. This clause defines the initialization point for all frames based on the window definition. For example, to have all windows begin at the same time, initialize them to a date-time *older than the first event likely to arrive in the view*.

```
WINDOW An_Hour AS (ORDER BY Trades.Time
                  RANGE INTERVAL '1' HOUR PRECEDING SLIDE
                  INITIALIZE TIMESTAMP '2003-03-05 00:00:00.000')
```

With this definition, all windows initialize at the same time: midnight. As such, each frame expires at the top of the hour (when minutes is 00:00.000). Because the window includes the SLIDE clause, all previous trades are discarded when the frame expires, and only new events arriving during the current hour are accepted.

Note: This clause acts as a filter in that it excludes all events before the initialization time.

The initialization time is a date-time literal value — a TIMESTAMP Literal. Further, the initialization value is static: it cannot change after the view is created.

Another Example

This initialization definition defines the current fiscal year, which begins on 1 July of the calendar year:

```
(RANGE INTERVAL '1' YEAR PRECEDING SLIDE
 INITIALIZE TIMESTAMP '1963-07-01 00:00:00')
```

24 Reportlets

Reportlets describe the contents of a view and present that information in a report that is either attached to an alert message or presented by an external system. Frequently, reportlets provide information about an event that puts the event into context. For example, when an inventory is low for a product, and a restock shipment is overdue, an alert might notify purchasing managers of that state and a reportlet attached to the alert might list the alternative suppliers for that product. Reportlets are attached to all subscribers of the associated alert.

There are two types of reportlets:

- Internal reportlets are the visual representation of the information in a view when the alert generated the reportlet. The presentation is a table formatted in either text, HTML, or a Microsoft Excel worksheet, and contains all of the information that was in the view.
- External (3rd-party) reportlets are produced by external reporting systems. External reportlets present a report based on view data passed to them when the user clicks a link to the external system. That system is responsible for generating and presenting the report.

➤ **In this Chapter:**

Creating Reportlets

To create a reportlet you must have:

- Create permission for business activities (see "Creating Permission" on page 258 for details)
- Read and Write permission on the business activity that will contain the reportlet
- Read Only permission on the view that will feed the reportlet.

➤ **To create a reportlet:**

1. Open the BAM Workbench Scenario Modeler.
2. Open an existing scenario that will contain the reportlet.
3. Select the Reportlets tab.
4. Chose New Reportlet.
5. Select the type of reportlet to create.

   Reportlets are formatted as HTML tables or Microsoft Excel spreadsheets.

   External (3rd-party) reportlets are defined and produced by external reporting systems based on the data passed to them.

   Note: The External reportlets option is only available when external links have been defined. For more information, see "Working with External Links" in *Using Business Activity Monitoring Workbench*.
6. Fill in the attribute fields on the Create Reportlet dialog.

   Note: For details, see "Reportlet Attributes" on page 291 or "External Reportlet Attributes" on page 293.

Save the reportlet as enabled, and it will immediately be ready for use.

You can also create a reportlet when creating or editing an alert. Doing so automatically attaches the reportlet to that alert.

Reportlet Attributes

Reportlets are formatted as HTML tables or Microsoft Excel spreadsheets. Each reportlet has the following attributes:

Attribute	Description
Name	Identifies the reportlet. The name can contain letters and numerals only. This name must be unique among reportlets within the same scenario. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the reportlet is enabled, or disabled. **Note:** When the containing scenario is disabled, you cannot make the reportlet enabled. The scenario must be enabled before the reportlet may be enabled.

Attribute	Description	
View	Business view from which the report draws its data.	
Excel format	(optional) Specifies how to format Excel-type reportlets. Leave this setting as <an empty spreadsheet> to produce an HTML formatted reportlet.	
	Template	Identifies a worksheet template for formatting the reportlet. Options are: Select an existing template — One that has already been created and uploaded to the BAM Server. New Custom Template — Opens the Add Template dialog where you identify an existing Microsoft Excel template on your computer. Saving and closing this dialog uploads the template from your machine to the server. An empty spreadsheet — Uses the Microsoft Excel default worksheet format.
	Sheet name	Name of the worksheet to contain the reportlet. Default is “Sheet1”.
	Sheet address	Location on the worksheet to present the reportlet. Default is “A1”.

External Reportlet Attributes

External reportlets present reports based on view data passed to them when the user clicks a link to the external (3rd-party) reporting system. That system is responsible for generating and presenting the report. The external reportlet definition identifies the external link, and the view information to pass to the external system that that system then uses to identify the report to present. For example, an external report might present a PDF that contains the complete description of a product identified in an alert. External reportlets have the following attributes:

Attribute	**Description**
Reportlet Name	Identifies the reportlet. The name can contain letters and numerals only. This name must be unique among reportlets within the same scenario. See "Object Namespace" on page 248 for details.
Status	Whether or not the reportlet is enabled or disabled. **Note:** When the containing scenario is disabled, you cannot make the reportlet enabled. The scenario must be enabled before the reportlet may be enabled.
Description	Optional description that may contain any text characters.
Data from View	Business view from which the report draws its data. Contains the column data to send to the external report.
Report Name	Name of the report in the external system. This is the DocName element in the URL that communicates with the external report system: `http://localhost.com?`**`DocName=<Report Name>`**

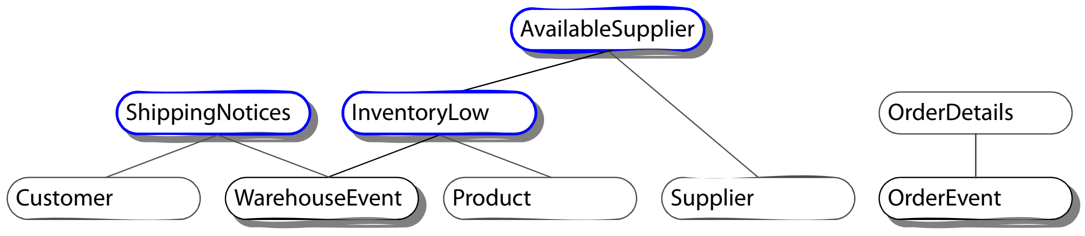
Report Parameters	Parameters to pass to the external system. Each parameter corresponds to a column in the view. The reportlet substitutes the value of each named column into the URL. For example, the URL is defined as follows:
	`...?DocName=<Report Name>`**`&Parameter1=PROD_ID&`**`...`
	It looks similar to this when sent to the external system:
	`...?DocName=<Report Name>`**`&product="product_id"&`**`...`

Attribute	Description
Report Arguments	This panel specifies argument values that can be assigned to arguments in the URL specified in the external link. For example, consider the case in which the external link defines a base URL for a Cognos report, and the base URL needs the following arguments: **ui.tool**=CognosView **ui.object**=contect/folder[@name='Demo Report']/report[@name='Inventory Levels by Retailers'] **ui.action**=run **run.outputFormat**=PDF The base URL is http//server-name/cognos8/cgi-bin/cognos.cgi?b_cation=xts.run&m=portal/launch.xts& When the Associated Report is called from the dashboard object, the values specified for the arguments in the Report Arguments pannel, the actual URL request is constructed as follows: http//server-name/cognos8/cgi-bin/cognos.cgi?b_cation=xts.run&m=portal/launch.xts&**ui.tool**-CognosView**& ui.object**=contect/folder[@name='Demo Report']/report[@name='Inventory Levels by Retailers']&**ui.action**=run&**run.outputFormat**=PDF
Display Link	Shows the complete, qualified URL that will appear in the alert message and is the link to the external report system.

The URLs used to communicate with the external report system begin with the string defined for the report in the External Links list on the BAM Workbench tab in the BAM Workbench. For more information about these locators, see "Working with External Links" in *Using Business Activity Monitoring Workbench*.

Reportlet Views

Reportlets retrieve their information from the business view that the alert is based on or from *any* view derived from the same event source (in the same event stream). For example, consider a rule that generated the customer alert based on the view *InventoryLow*. Another view, *AvailableSuppliers* adds context by indicating alternative suppliers. The reportlet attached to the alert may draw information from either of these views. Further, because *ShippingNotices* is derived from the same event source, you could also retrieve information from it. However, you cannot retrieve information from OrderDetails because it is on a different event stream.



For more details about the information that appears in the reportlet view, see "Reportlet Filtering" on page 36.

Note that the reportlet view may not be a synchronized join. See "Synchronized Joins" on page 353 for details.

25 Roles

Roles define permissions associated with User and provide a way to assign the same permissions to objects and groups of sure without having to set those permission for each individual user of the group.

➤ In this Chapter:

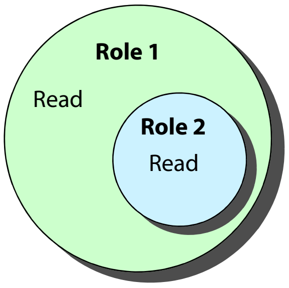
Overview

Roles define the minimum sets of Permissions associated with Users. Roles provide a way to quickly assign *the same permissions* to an object or class of objects and for groups of users without having to set those permissions for each individual user of the group. For example, an “operator” role might provide full permissions to agents but not to events or business activity objects, while a “application developer” role might have full permissions on all objects, except agents.

Users may belong to none, one, or more roles. To see which roles a user belongs to, edit the user account and view the User Details Tab. To see which roles you belong to, click Account Settings and view the tab.

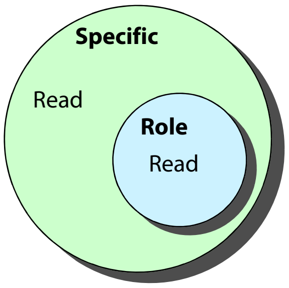
A user’s permission for a particular operation is the *maximum* of all the permissions associated with that user’s roles and with any individual permissions assigned to the user for the object. Consider a user with two roles: one has Read-only access to the views class, and the other has read and write access. The maximum permission on views for this user is Read and Write, and as such, this user can edit views.

When a user has multiple roles with overlapping permissions on an object, the greatest permission is used. This user has Read and Write access.



Similarly, if a user has one role, and that role is limited to Read only for all views but has been individually assigned Read and Write to a particular view, that user may edit that view.

When a user’s specific permission overlaps with a role, greatest permission is used. This user has Read and Write access.



Note: For details about specific access levels, see “Permissions” on page 253. Any unusual interactions between object permissions and roles are described in the discussions of that object.

Roles are objects maintain lists of users and associated permissions, and like all other Business Activity Monitoring objects, roles are protected by permissions. Only users with specific permissions on a role — or on all roles — can perform that action on the role. For example, to add users to a role, you need Read and Write permission on that role.

To see the roles defined in the installation, look at the Roles list in the Administration Console of the BAM Workbench.

Creating roles

To create roles, you need Create permission for roles. For each object class, you can assign up to the greatest permission that you have for that class. For example, if you have Create permission for a class, for that class, you can assign any of

- No Access,
- Read only,
- Read and Write
- Create

➤ **To create a role:**

1. Open the Administration Console.
2. Click Roles to see the list of all currently defined roles.
3. Click New Role.
4. Fill in the role attributes, assign access permissions, and identify the members of the role.
5. Save the role to begin using it.

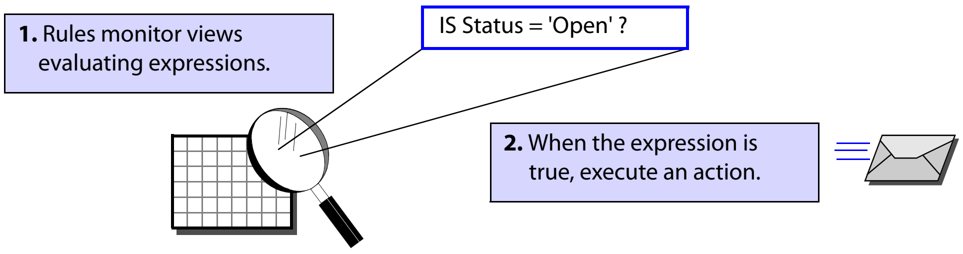
Role Attributes

Each role object has the following attributes:

Attribute	**Description**
Name	Identifies the role object. The name can contain letters and numerals only. This name must be unique among roles. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Access Permissions	Permissions for each class of objects assigned to this role. These are the *minimum* permissions. A user may have a greater permissions assigned individually to the class (see "Access Permissions Tab" on page 335) or to a specific object.
Members	Users associated with this role.

26 Rules

Rules monitor business activities by analyzing business views looking for metrics that meet specific conditions. Rule conditions are spreadsheet-like formulas that evaluate the changing business metrics looking for exceptional conditions. When a condition is found to exist, an alert of that incident is sent to key personnel.


1. Rules monitor views evaluating expressions.
IS Status = 'Open' ?
2. When the expression is true, execute an action.

```
WHEN [ Status = 'Open' ] FIRE [
```

You can create rules that send alerts every time the condition is found to exist (fire), create rules that send alerts once and ignore subsequent events until the initial condition is resolved (raise), or create rules that reset (lower) previously raised rules.

Note: Prior to Version 3.2, the rule could only monitor views derived from the scenario “default” view. This restriction no longer applies.

➤ **In this Chapter:**

Creating Rules

You can create rules from scratch or clone and modify existing rules. Both require the user to have Create permission for business activities (see "Creating Permission" on page 258 for details), Read and Write permission on the business activity that will contain the rule, and Read Only permission on the view that will feed the rule.

➤ To create a new rule:

1. Open the Scenario Modeler.
2. Open an existing business activity.
3. Open an existing scenario.
4. Click the New Rule button.
5. Fill in the rule attributes (described below) in Step 1 of the Rule Definition form. Additionally:
    - Choose the data source that the rule will monitor.
    - If the scenario has a "default view", that one appears selected by default. Choose another source to monitor by clicking Select Data Source.
    - For a view, choose the view.
    - For a cube, choose the dimension level in a cube. Optionally you may also apply a filter that further restricts the data that the rule monitors.

    If the source contains data, that data appears to provide a sample of what to expect. When the source is empty, the form displays just the column names and the message "No Data Available".

    - For the Action Taken, identify the alert to activate when the rule condition is met. Do one of the following:
        - Select an existing alert by clicking this alert in the Rule effect field.
        - Click Next to access the Alert Definition form and define or review the alert. See "Alert Attributes" on page 29 for details about the fields on this form.
6. Finish creating the rule.

If the rule was saved enabled, it will immediately begin monitoring the view for events.

➤ To clone an existing rule:

1. Edit the rule you want to clone.
2. Change the rule name, and change the other attributes that differ from the original rule.
3. Choose "Save as New Rule.

Rule Attributes

Every rule has the following attributes:

Attribute	**Description**
Data source	Identifies the business view or cube that the rule monitors.
Name	Identifies the rule object. The name can contain letters and numerals only. This name must be unique among rules within the same scenario. See "Object Namespace" on page 248 for details.
Status	Specifies if the rule is enabled (receiving new event information) or disabled. **Note:** When the containing scenario is disabled, you cannot make the rule enabled. The scenario must be enabled before the rule may be enabled.
Description	Optional description that may contain any text characters.
Rule condition	The spreadsheet-like, true or false formula that is evaluated against the associated business view. When the formula evaluates to True, the condition is said to exist and the system then sends the alert. See "Rule Condition" on page 302 for details.
Holds for	Sends the alert only when the condition holds true for an entire specified length of time. No alert is sent if the condition becomes false at any time during the wait. When you omit the duration, the system sends the alert as soon as the condition exists. You can specify a value using one of the following methods: • Specify a number to indicate the length of time wait. • Specify the name of a column in the view that contains the number. When you use this option, the rule takes the value from the event in the view. **Note:** In locales where daylight savings time is observed, durations of days, months, and years are adjusted accordingly. As such, while 1 day is typically 24 hours long, it may be 23 or 25 hours depending on the time of year.
Action taken	What to do when the condition exists. Rules can send alerts every time the condition is found to exist (*fire*), send alerts once and ignore subsequent events until the initial condition is resolved (*raise*), or reset (*lower*) previously raised rules. See "Rule Action" on page 302 for details.

Rule Condition

A rule condition is a formula that tests the row in the associated business view looking for a specific condition. When the condition exists the rule action activates an alert. Formulas can be simple tests for a value in a column in the view, such as Status='Resolved', or they can be complex Boolean expressions with functions, operators, and parenthesis groupings, such as (Status='Resolved' OR Status='Assigned') AND UPPER(cust_tier)='HIGH'.

A rule condition formula contains any number of column references, operators, and functions. However, the formula must conform to the following:

- The formula result must be Boolean, returning True or False.
- All column references must be in the associated business view.
- Only scalar functions (functions that apply to a single row in a view) may be used. To see which functions are available, click More Functions when entering the rule condition. See "Functions" on page 114 for detailed descriptions of the functions.

Rule Action

Rules can have one of three effects:

- Send alerts every time the condition is found to exist (*fire*). A fire action sends an alert every time a rule identifies an exceptional condition. For example, consider a customer support center that tracks customer problems as ticket events, an alert might be fired every time a new ticket is opened.
- Send alerts once and ignore subsequent events until the initial condition is resolved (*raise*). A raise action sends an alert message when the rule's condition applies but ignores subsequent events until after the initial condition is resolved. A raise action is useful when you do not want multiple alerts for situations where the rule condition is true for multiple, related events. For example, if an open customer problem ticket is edited, you do not want another alert for the edit event, even though the status of second event is still "open."

  The "for a specific occurrence" option allows you to send alerts once for each specific occurrence of the named column. For example, to send an alert every time a new problem ticket is opened, you might identify Ticket as the specific occurrence column. That way, one alert is sent for each ticket's "open" event, but the alert is ignored for all subsequent events to that ticket while its status remains "open."
- Reset (*lower*) previously raised rules to allow them to send alerts.

Specific Occurrences

When a Raise rule activates an alert, the alert does not activate again — subsequent Raise rules for the alert are ignored — until a Lower rule first resets it. For example, when a rule condition is "Status=Open", the first event below activates the alert, but the subsequent ones are ignored unless the alert's state is first lowered. For example:

```
Ticket Status
------ ------
0703   Open      << Raise
0706   Open      << Ignore
0704   Open      << Ignore
0705   Open      << Ignore
```

In this example, it is more likely that you would want an alert for each Open event. To do that, you can use a "For a specific" condition which activates an instance of the alert for each unique occurrence of the values in the specified columns.

```
Raise someAlert when Status = 'Open' for unique occurrences of Ticket.
```

By identifying Ticket as the specific column, an alert instance will be activated, and a message sent, every time for each Open event that does not already have an instance for the specific ticket number. Similarly, you can reset each alert instance individually with specific Lower rules.

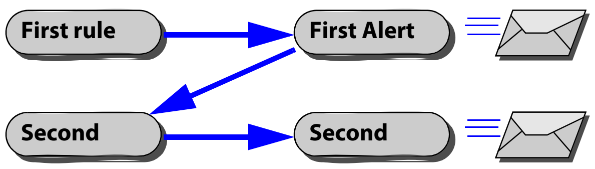
```
Lower someAlert when Status = 'Open' for unique occurrences of Ticket.
```

Note: If you ignore the specific fields in the Lower rule, the rule will reset all instances of the alert that have been raised.

Monitoring Alerts

Rules typically analyze Business Views looking for metrics that meet specific conditions. However, rules can also monitor generated alerts looking for conditions that require further attention with the IS_RAISED() function.

Monitor an alert and send a second one when the first remains raised.



For example, if an alert was sent 4 hours earlier and is still in a raised state, another rule might notice that fact and generate a new, escalated alert. Consider these rule descriptions, where *EscalateAlert* is raised only when *OpenAlert* remains raised for at least 4 hours:

```
Raise OpenAlert when Status = 'Open'
Reset OpenAlert when Status <> 'Open'
Raise EscalateAlert when IS_RAISED('OpenAlert') holds for 4 HOUR.
Reset EscalateAlert when NOT IS_RAISED('OpenAlert')
```

Specific Alerts

When an alert is generated for unique occurrences of fields, the system tracks each alert by those field values. For example, the following rule raises alerts and tracks the open ones by the unique values of the Ticket field:

```
Raise OpenAlert when Status = 'Open' for unique occurrences of Ticket
```

To properly track this alert, the rule with the IS_RAISED() needs the same specific condition:

```
Raise EscalateAlert when IS_RAISED('OpenAlert') holds for 4 HOUR
    for unique occurrences of Ticket
```

If you were to omit the specific field condition, the *EscalateAlert* would be raised for the first *OpenAlert* only. Similarly, you must reset the alerts with rule conditions specific to the same fields, or you will reset all of the raised alerts.

```
Reset OpenAlert when Status <> 'Open' for unique occurrences of Ticket
Reset EscalateAlert when NOT IS_RAISED('OpenAlert')
    for unique occurrences of Ticket
```

Monitoring the System Log

Business Activity Monitoring generates messages that facilitate software service and maintenance by producing reports suitable for analysis by end users, system administrators, support engineers, and software development teams. For details about the logging system, see "Understanding Logging" in *Using Business Activity Monitoring Workbench*.

You can build a rule that monitors the messages looking for high priority error conditions, then report them to key administrators. To do that, follow the steps in "Monitoring the Logs" in *Using Business Activity Monitoring Workbench*.

27 Salesforce

You can access and import tables from a Salesforce system.

This is a two-step process. The first step is the creation of a Salesforce agent that connects to Salesforce tables. The second step is the use of a separate extraction wizard to import Salesforce events and contexts.

The imported Salesforce objects can then be used as data sources for dashboard objects like business views and cubes.

➤ **In this Chapter:**

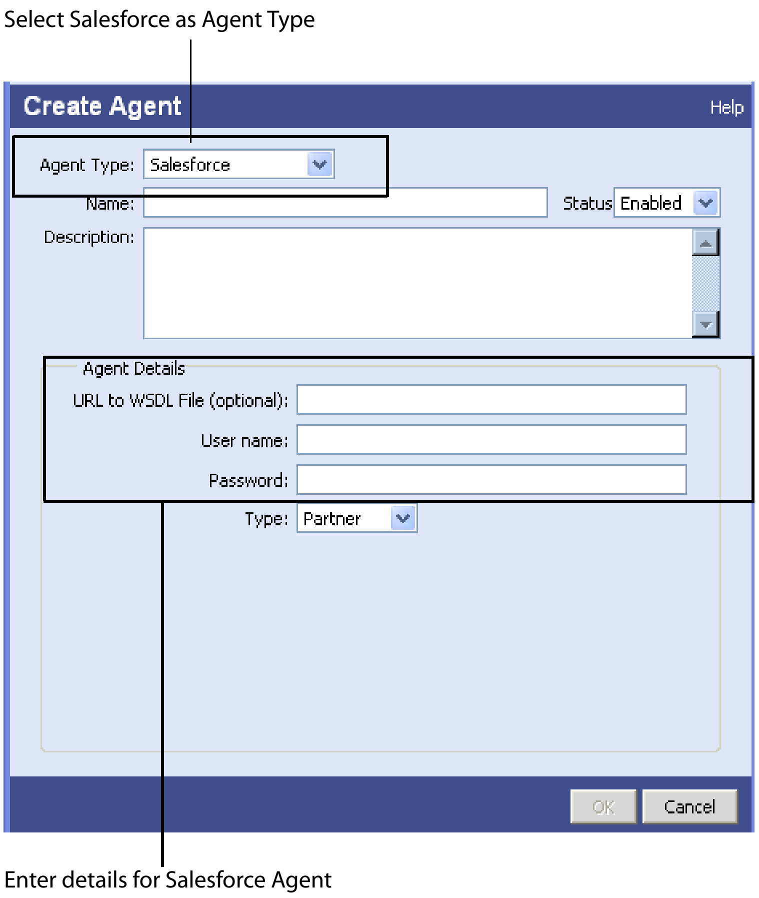
Creating a Salesforce Agent

A Salesforce agent imports Salesforce tables for use in events and contexts.

➤ **To create an Salesforce agent:**

1. Open the BAM Workbench Administration Console.
2. Click **New Agent.**
3. Choose **Salesforce** as the source type.

   When you select the source type, the subsequent settings redisplay to show settings appropriate to that type.
4. Enter a **Name** value and set **Status** to Enabled.
5. Enter a **User name** and **Password**.


Select Salesforce as Agent Type
Create Agent
Help
Name:
Description:
Agent Details
URL to WSDL File (optional):
User name:
Password:
OK
Cancel
Enter details for Salesforce Agent

Agent Type: Salesforce
Status: Enabled
Type: Partner

6. Click **OK**.

   You can now use a wizard to import Salesforce events and contexts into BAM Workbench as described in the next section.

Importing Salesforce Events and Contexts

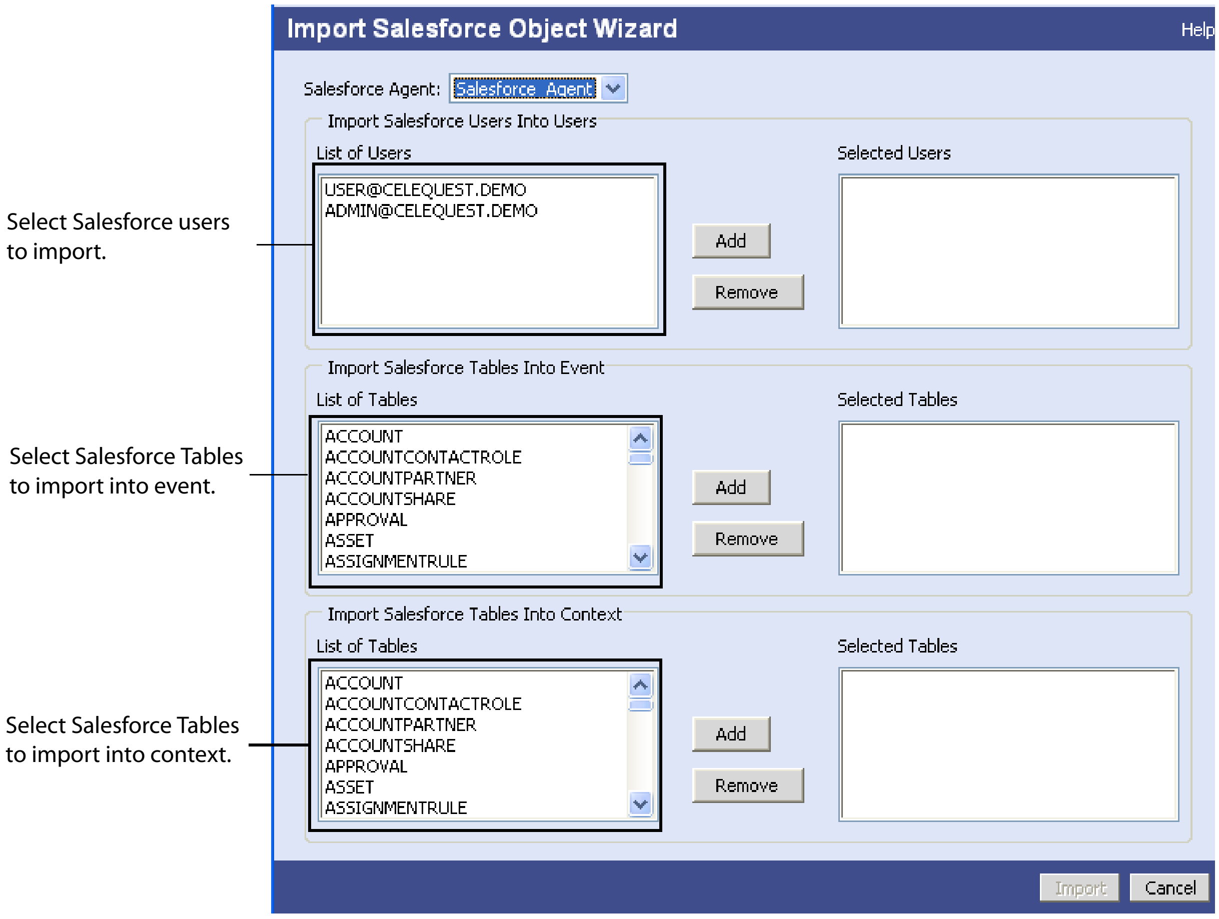
Execute the following procedure to use the Import Salesforce Object Wizard.

➤ **To import Salesforce events and contexts:**

1. Open the BAM Workbench Administration Console.
2. Click on **Wizards** in the Navigation Tree.

   This action lists the available wizards in the main panel to right.
3. Click **Import Salesforce Object Wizard**.

   The Import Salesforce Table dialog displays.



4. For Salesforce Agent, select the agent you created in "Creating a Salesforce Agent" on page 306.
5. Select the users to import from **List of Users**.
6. Select the tables to import from **List of Table for event** and click Add.

   When you add a table, it is removed from the **List of Tables** and placed in the **Selected Tables** list. To move it back, click Remove.

7. Select the tables to import from **List of Table** for context and click Add.

   When you add a table, it is removed from the **List of Tables** and placed in the **Selected Tables** list. To move it back, click Remove.

8. Click **OK**.

The events and contexts are created from the selected tables in Application Workbench with _events and _context to the table name under Events and Contexts, respectively. For example, if you imported a Salesforce table Account into an event, it has the event name Account_event.

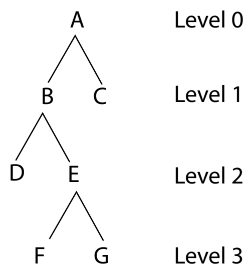
Note: If a table in the **Selected Tables** list is being used for an event or a context, you cannot remove it from **Selected Tables**. You must first delete the event or context.

Salesforce Flattening Function

A function is available for use with Salesforce Agents that flattens a Salesforce table. However, the table can only be flattened if the table represents a tree structure, where the top level represents the root element of the tree. After the table is flattened the each row in the new table represents a leaf node of the original tree. For example, consider the following table:

NodeName	NodeID	NodeParentId
A	1	NULL
B	2	1
C	3	1
D	4	2
E	5	2
F	6	5
G	7	5

This table can be represented by the tree shown below. The tree has four levels where Level 0 has the root element A.



After flattening, the rows in the new table represents the leaf nodes C, D,F, and G.

Level0	Level1	Level2	Level3
A	B	D	NULL
A	B	E	F
A	B	E	G
A	C	NULL	NULL

Syntax

The flattening function has the following syntax:

```
FLATTEN( tableName, idColumnName, parentIdColumnName, nameColumn,
prefixForLevelColumns )
```

Parameters

The parameters of the function are defined as follows:

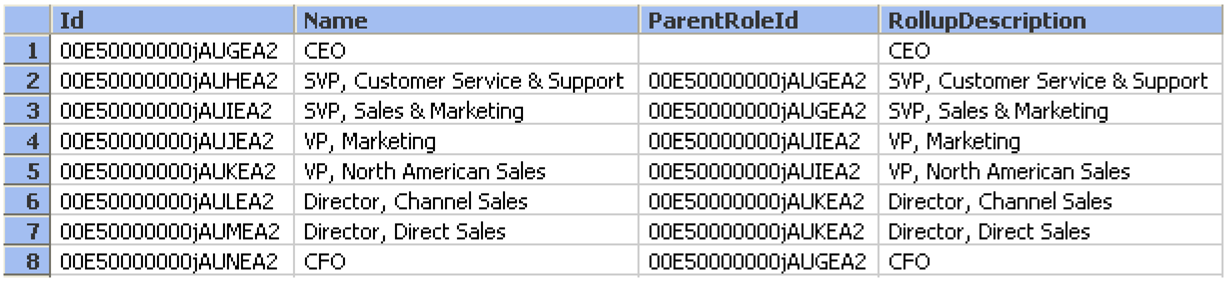
- tableName — The name of the table to flatten.
- idColumnName — The ID column of the table (that is, the column to which `parentIdColumnName` will join). For example, in the description above, the idColumnName is "NodeID."
- parentIdColumnName — The column containing the node IDs of the parent nodes for each row. For example, in the description above, node A has an ID of 1 and is the parent of node B; therefore, the parent ID in the column "NodeParentID" for Node B is 1. Likewise, the parent node of node D is node B; therefore, the NodeParentID for D is 2 because the node ID of node B is 2.
- nameColumn — The column representing the data to be shown in the flattened table.
- prefixForLevelColumns — The string prefix that will be prepended to the column. In the description above, the prefix is "Level."

Remarks

You must have data caching with prefetch enabled in your context for the flatten function to work properly. See "Caching Context Queries" on page 48 for details about enabling caching.

Example

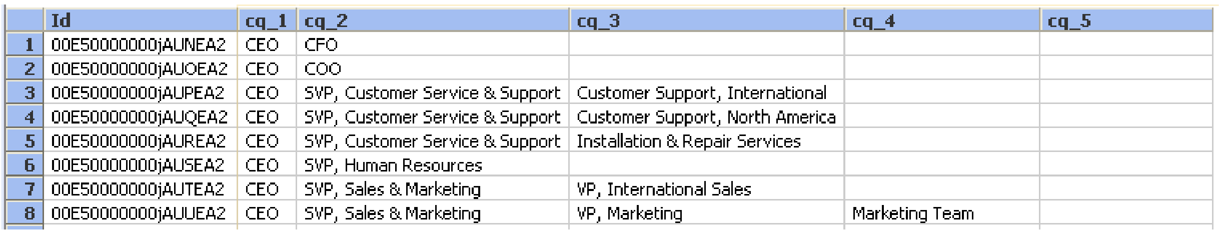
The following is a table before flattening.



	Id	Name	ParentRoleId	RollupDescription
1	00E50000000jAUGEA2	CEO		CEO
2	00E50000000jAUHEA2	SVP, Customer Service & Support	00E50000000jAUGEA2	SVP, Customer Service & Support
3	00E50000000jAUIEA2	SVP, Sales & Marketing	00E50000000jAUGEA2	SVP, Sales & Marketing
4	00E50000000jAUJEA2	VP, Marketing	00E50000000jAUIEA2	VP, Marketing
5	00E50000000jAUKEA2	VP, North American Sales	00E50000000jAUIEA2	VP, North American Sales
6	00E50000000jAULEA2	Director, Channel Sales	00E50000000jAUKEA2	Director, Channel Sales
7	00E50000000jAUMEA2	Director, Direct Sales	00E50000000jAUKEA2	Director, Direct Sales
8	00E50000000jAUNEA2	CFO	00E50000000jAUGEA2	CFO

The following is the table after it has been flattened with the function:

```
    flatten(USERROLE, ID, ParentRoleID, Name, cq_)
```



	Id	cq_1	cq_2	cq_3	cq_4	cq_5
1	00E50000000jAUNEA2	CEO	CFO			
2	00E50000000jAUOEA2	CEO	COO			
3	00E50000000jAUPEA2	CEO	SVP, Customer Service & Support	Customer Support, International		
4	00E50000000jAUQEA2	CEO	SVP, Customer Service & Support	Customer Support, North America		
5	00E50000000jAUREA2	CEO	SVP, Customer Service & Support	Installation & Repair Services		
6	00E50000000jAUSEA2	CEO	SVP, Human Resources			
7	00E50000000jAUTEA2	CEO	SVP, Sales & Marketing	VP, International Sales		
8	00E50000000jAUUEA2	CEO	SVP, Sales & Marketing	VP, Marketing	Marketing Team	

Salesforce Picklist Function

In addition to the flatten function, there is also a picklist function available for use with Salesforce agents. The picklist function returns a single column dataset of the possible values for a picklist typed column in Salesforce.

Syntax

The flattening function has the following syntax:

```
PICKLIST( tableName,pickListColumn )
```

Parameters

The parameters of the function are defined as follows:

- tableName — The name of the table from which to return the single column dataset.
- pickListColumn— The column in the table specified by tableName to return.

Remarks

You must have data caching with prefetch enabled in your context for the picklist function to work properly. See "Caching Context Queries" on page 48 for details about enabling caching.

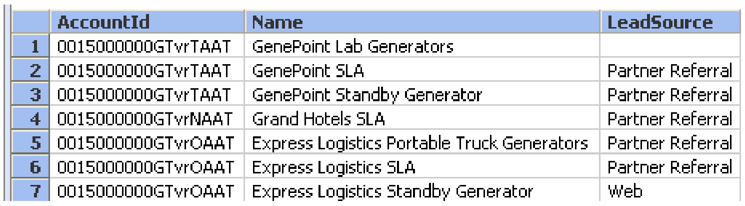
The column to return must be a picklist typed column.

Example

The following picklist function returns the dataset of possible values from the LeadSource column in the Opportunity table.

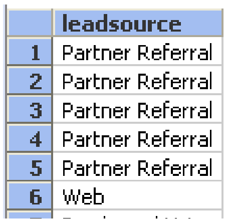
```
picklist(opportunity,leadsource)
```

The illustration below shows the Opportunity table.



	AccountId	Name	LeadSource
1	0015000000GTvrTAAT	GenePoint Lab Generators	
2	0015000000GTvrTAAT	GenePoint SLA	Partner Referral
3	0015000000GTvrTAAT	GenePoint Standby Generator	Partner Referral
4	0015000000GTvrNAAT	Grand Hotels SLA	Partner Referral
5	0015000000GTvrOAAT	Express Logistics Portable Truck Generators	Partner Referral
6	0015000000GTvrOAAT	Express Logistics SLA	Partner Referral
7	0015000000GTvrOAAT	Express Logistics Standby Generator	Web

The picklist function retrieves the LeadSource column.



	leadsource
1	Partner Referral
2	Partner Referral
3	Partner Referral
4	Partner Referral
5	Partner Referral
6	Web

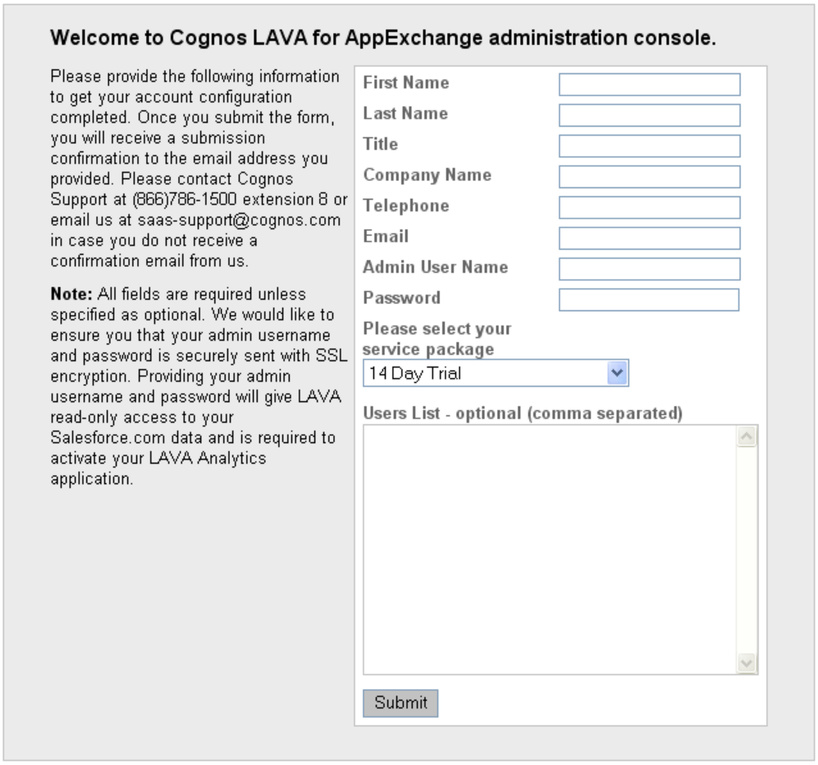
Salesforce Administration Console

To obtain a Salesforce.com account you will need to access the Cognos LAVA for AppExchange administration console. To access the console, enter the following in your browser:

http://[localhost]:[port]/bam/jsp/salesforceadmin.jsp

On the page that displays, enter the following:

- Your first and last name
- Title
- Company name
- Contact telephone
- E-mail address
- Administrator's name.
- Under service package select either 14 Day Trail or Standard Service Subscription.
- List the names of additional users for this account in the User's List field.



Welcome to Cognos LAVA for AppExchange administration console.

Please provide the following information to get your account configuration completed. Once you submit the form, you will receive a submission confirmation to the email address you provided. Please contact Cognos Support at (866)786-1500 extension 8 or email us at saas-support@cognos.com in case you do not receive a confirmation email from us.

Note: All fields are required unless specified as optional. We would like to ensure you that your admin username and password is securely sent with SSL encryption. Providing your admin username and password will give LAVA read-only access to your Salesforce.com data and is required to activate your LAVA Analytics application.

First Name
Last Name
Title
Company Name
Telephone
Email
Admin User Name
Password
Please select your service package: 14 Day Trial
Users List - optional (comma separated)

After you click Submit, you will be sent an e-mail confirming that your request has been successfully submitted. It will also inform you that a representative will contact you to confirm the submission and provide you with the information about next steps to complete the configuration process.

28 SAP Connectivity

You can access and import fact tables (in the form of ODS objects) and cubes (in the form of OLAP cubes) from an SAP system.

This is a three-step process. The first step is the creation of a standard JDBC agent that connects to the SAP database. The second is the creation of an ERP agent that uses an SAP metadata JDBC agent to access the SAP system as a source type. The third step is the use of a separate extraction wizard to import ODS objects or OLAP cubes.

The imported ODS objects and cubes can then be used as data sources for dashboard objects like business views and cubes.

➤ **In this Chapter:**

Creating an SAP Agent

An SAP agent is an ERP agent that uses a JDBC agent to access the database for the SAP system.

➤ **To create an SAP agent:**

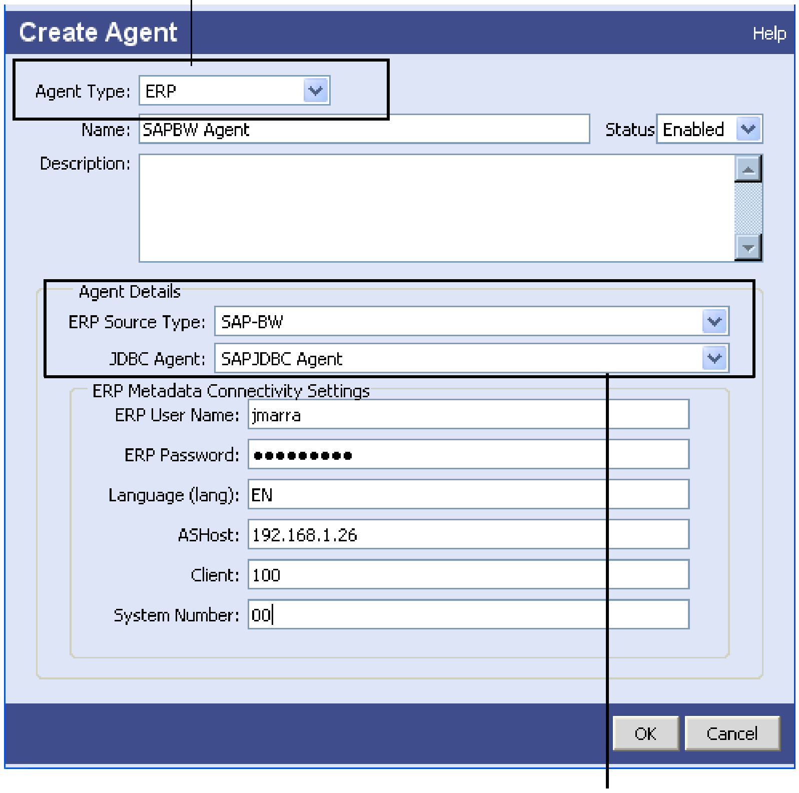
1. Open the BAM Workbench Administration Console.
2. Create a JDBC agent that connects to the SAP database.

   For instructions on creating a JDBC agent, see "JDBC Agents" on page 224.
3. Click **New Agent.**
4. Choose ERP as the source type.

   When you select the source type, the subsequent settings redisplay to show settings appropriate to that type.
5. Enter a **Name** value and set **Status** to Enabled.
6. For **ERP Source Type**, select SAP (or variation thereof).

   Note: This selection may cause the connectivity settings below to redisplay again.

Select ERP as Agent Type



Agent Type: ERP
Name: SAPBW Agent
Status: Enabled
Description:

Agent Details
ERP Source Type: SAP-BW
JDBC Agent: SAPJDBC Agent

ERP Metadata Connectivity Settings
ERP User Name: jmarra
ERP Password: ●●●●●●●●●●
Language (lang): EN
ASHost: 192.168.1.26
Client: 100
System Number: 00

Select SAP as ERP Source Type and JDBC Agent created for the SAP agent

7. For **JDBC Agent**, select the JDBC agent you created earlier to connect to the SAP database.
8. Complete the **ERP Metadata Connectivity Settings** as appropriate.

9. Click **OK**.

   You can now use a wizard to import ODS objects and OLAP cubes into BAM Workbench, as described in the following sections.

Importing ODS Objects

Execute the following procedure to use the Import ODS Object Wizard.

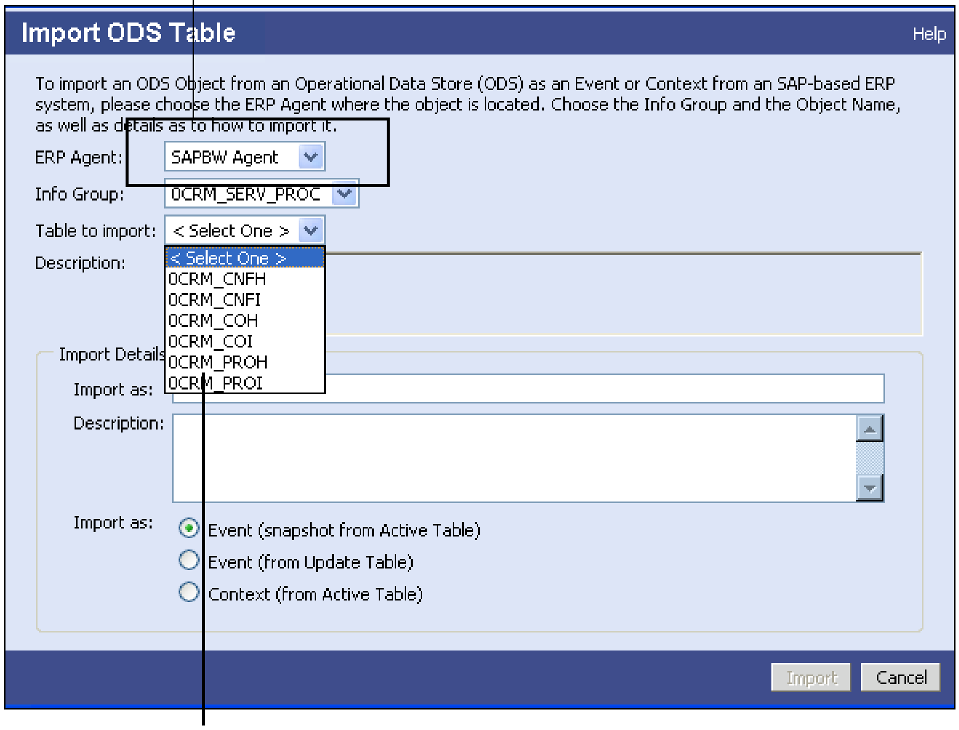
➤ **To import ODS objects from an SAP system:**

1. Open the BAM Workbench Administration Console.
2. Click on **Wizards** in the Navigation Tree.

   This action lists the available wizards in the main panel to right.
3. Click **Import ODS Object Wizard**.

   The Import ODS Table dialog displays.
4. For ERP Agent, select the agent you created in "Creating an SAP Agent" on page 314.
5. Select the Info Group that contains the desired ODS object.
6. This selection populates the **Table to Import** dropdown list.

Select the ERP agent previously configured



To import an ODS Object from an Operational Data Store (ODS) as an Event or Context from an SAP-based ERP system, please choose the ERP Agent where the object is located. Choose the Info Group and the Object Name, as well as details as to how to import it.

ERP Agent: SAPBW Agent
Info Group: 0CRM_SERV_PROC
Table to import: < Select One >
Description:

< Select One >
0CRM_CNFH
0CRM_CNFI
0CRM_COH
0CRM_COI
0CRM_PROH
0CRM_PROI

Import as:
Description:
Import as: [✓ Event (snapshot from Active Table)] Event (from Update Table) Context (from Active Table)

Select the desired fact table from the populated list

7. From the **Table to Import** dropdown list, select the desired fact table.

8. Specify the following import parameters:
   - For **Import as...**, enter the name by which the resulting event or context will be identified in BAM Workbench.
   - For Description, enter an optional description.
   - Specify how the imported ODS object will be utilized: Event (two options) or Context.

   Note: For more information about Events and Contexts, see “Events” on page 76 and “Context” on page 43.

9. Click **OK**.

   The selected ODS object is imported and can now be accessed via the **Events** or **Contexts**, depending on the import options selected.

Importing OLAP cubes

Execute the following procedure to use the Import ERP Cube Wizard.

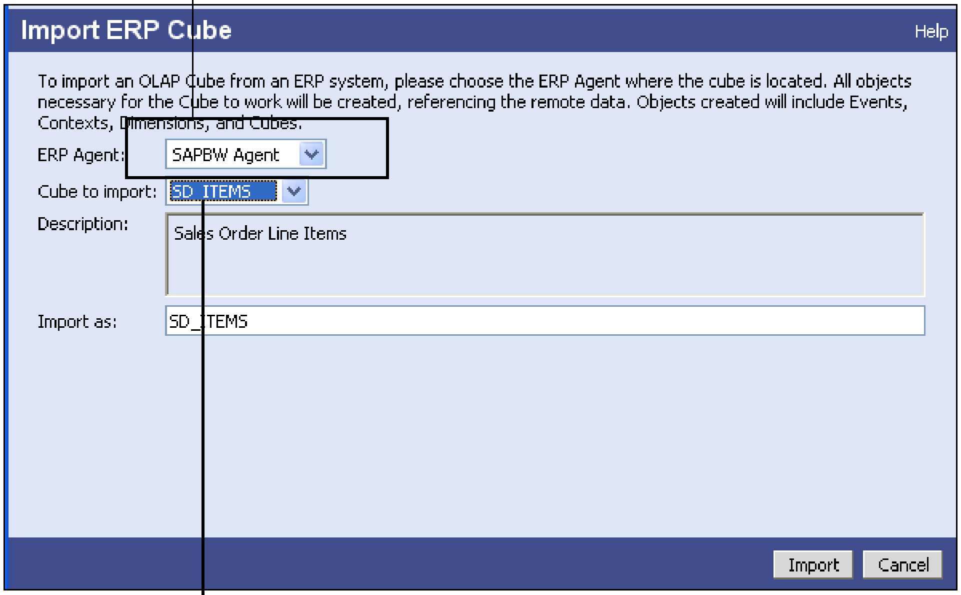
➤ **To import OLAP cubes from an SAP system:**

1. Open the BAM Workbench Administration Console.
2. Click on **Wizards** in the Navigation Tree.

   This action lists the available wizards in the main panel to right.
3. Click **Import ERP Cube Wizard**.

   The Import ERP Cube dialog displays.
4. For **ERP Agent**, select the agent you created in "Creating an SAP Agent" on page 314.
5. Select the cube you wish to import.
6. In the **Import as** field, accept the automatic default (the same names as in SAP) or enter a new name by which the imported cube will be identified in BAM Workbench.

Select the ERP agent previously configured


Import ERP Cube

To import an OLAP Cube from an ERP system, please choose the ERP Agent where the cube is located. All objects necessary for the Cube to work will be created, referencing the remote data. Objects created will include Events, Contexts, Dimensions, and Cubes.

ERP Agent: SAPBW Agent

Cube to import: SD_ITEMS

Description: Sales Order Line Items

Import as: SD_ITEMS

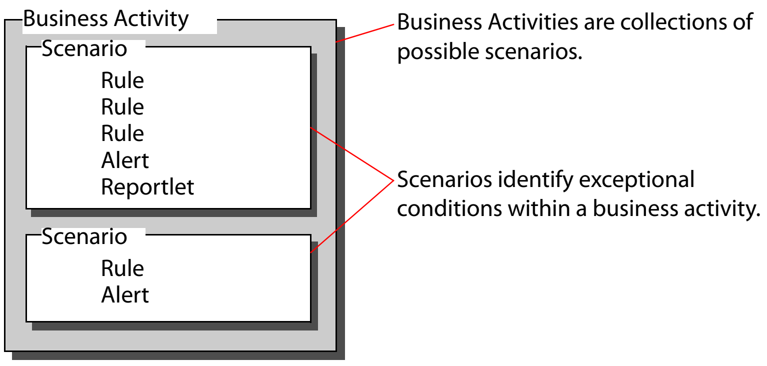
Select the desired cube from the populated list

7. Click **OK**.

   The selected fact table is imported and can now be accessed via the **Cubes** panel.

29 Scenarios

A scenario is a collection of rules, alerts, and reportlets that identify exceptional business conditions. The rules in the scenario are the tests that determine when the exceptional condition exists or when it no longer exists.


Business Activity
Scenario
Rule
Rule
Rule
Alert
Reportlet
Scenario
Rule
Alert
Business Activities are collections of possible scenarios.
Scenarios identify exceptional conditions within a business activity.

Tips:

- Deleting a scenario deletes its contained rules, alerts, and reportlets.
- Disabling a scenario disables its contained rules, alerts, and reportlets.

➤ **In this Chapter:**

Creating Scenarios

To create a scenario, you must have the following:

- Create permission for business activities (see "Creating Permission" on page 258 for details)
- Read and Write permission on the business activity that will contain the scenario
- (Optional) Read Only permission on the default view for the scenario.

➤ **To create a new scenario:**

1. Open the Scenario Modeler.
2. Select a Business Activity to contain the scenario
3. Click New Scenario
4. Fill in the fields of the New Scenario dialog.

Scenario Attributes

Every scenario has the following attributes:

Attribute	Description
Folder Status	Specifies if the rule is enabled (receiving new event information) or disabled. When an scenario is disabled, all of its rules, alerts, and reportlets are also disabled. **Note:** When the containing business activity is disabled, you cannot make the scenario enabled. The business activity must be enabled before the scenario may be enabled.
Scenario Name	Identifies the scenario object. The name can contain letters and numerals only. This name must be unique among scenarios. See "Object Namespace" on page 248 for details.
Description	Optional description of the scenario that may contain any text characters.
View	Identifies the default business view that the rules of this scenario monitor, and reportets report on. "No default view" requires that you pick a view for rules and reportlets when you create them.
View Description	Displays the description of the selected View.

Deleting Scenarios

Deleting a scenario deletes its contained rules, alerts, and reportlets. Once deleted, they cannot be restored.

➤ **To delete a scenario:**

1. Open the Scenario Modeler.
2. Select the business activity that contains the scenario.
3. Select the scenario to remove.
4. Click Delete Scenario.

30 SELECT

C-SQL SELECT statements define the views that manage information in Business Activity Monitoring. BAM Workbench constructs SELECT statements based on the views you define in its graphical user interface, and then passes them to the BAM Server(s) for instantiation. You can see the complete SELECT statement that defines a view in the BAM Workbench by clicking the Displaying SQL Expressions option when creating or editing a view.

Some of the advanced features of the SELECT command cannot be expressed by the options in the BAM Workbench. For example, complex join conditions, query windows, and table expressions must be entered manually in fields in the user interface.

This topic describes the syntax and features of the C-SQL SELECT command in detail.

Note: The C-SQL SELECT command is a subset and extension of ANSI SQL-99, a query language standard. The C-SQL implementation supports outer joins using the ANSI outer join (left outer join and right outer join) syntax, and aggregation functions in the select clause. Further, each individual statement is treated as a transaction and is committed as soon as it executes.

➤ **In this Chapter:**

Syntax

The operators that define the SELECT specifications are:

```
SELECT selectList
       FROM joinClauses
     [ WHERE searchCondition ]
     [ GROUP BY groupClause ]
     [ WINDOW windowClause ]
     [ ORDER BY orderClause ]
```

The operators are applied in the following order:

1. FROM Clause — specifies the base tables or views that provide data to this view.
2. WHERE Clause (optional) — filters the input to match specified criteria.
3. GROUP BY Clause (optional) — groups the resulting table on one or more columns.
4. Select List — defines the columns to appear in the resulting table.
5. WINDOW clause (optional) — defines windows used by aggregate functions in the Select list. This clause is described in the section "Query Windows" on page 272.
6. ORDER BY (optional) — orders (sorts) the resulting table.

Select List

Defines the columns to appear in the resulting virtual table.

The select list immediately follows the SELECT keyword and has two forms:

- Just an asterisk (*) to choose all columns that are part of the source table.

  ```
  SELECT * ...
  ```

  The resulting view contains the columns of each of the input tables or views, in the order that they occur in the source, and in the order listed in the FROM Clause. For Outer Joins, resulting columns that do not exist in both references are assigned NULL values.

- A list of unique column names or derived columns.

  ```
  SELECT columnNameList
  ```

  Where each columnName is separated by a comma (,) and is defined as

  ```
  columnName [[AS] aliasName [ OVER (windowClause) ] ]
  ```

  The AS option assigns a new name to the associated column. The literal "AS" is optional.

  The OVER option defines an in-line window. See "Query Windows" on page 272 for details.

  Where each columnName is one of the following:

  - simple column reference — the column name.

    ```
    current_rental_price
    ```

- qualified column reference — one prefaced by the table name.

  ```
  movie_titles.current_rental_price
  ```

  To select all columns from one table while selecting some columns from other tables, use a qualified column reference and specify an asterisk (*) for the column name. For example, the following selects all *movie_title* columns, and two columns from the *media* table:

  ```
  movie_titles.*, media.media_type, media.name
  ```

- derived column — an expression (possibly a case expression, see “CASE Expression” for details):

  ```
  MAX((movie_titles.current_rental_price/2)) AS Half_Price
  ```

  For derived columns the aliasName is required.

CASE Expression

A CASE expression returns the result of an expression that corresponds to a matching true condition. Optionally, each condition may return NULL instead. If no condition is found to be true, the expression returns the result of the ELSE condition or NULL when ELSE is omitted.

There are two forms of CASE expressions:

- Simple condition — Evaluates the *caseExpression* and compares it against the result of each *equalsExpression* until one matches, then returns the corresponding *resultExpression* result. Each of the *equalsExpression* must of a type comparable to the *caseExpression*.

  ```
  CASE caseExpression
      [ { WHEN equalsExpression THEN { resultExpression | NULL }}...]
      [ ELSE { resultExpression | NULL } ]
  END
  ```

- Search condition — Evaluates each *searchCondition* until one is found to be true, then returns the corresponding *resultExpression* result.

  ```
    CASE WHEN searchExpression THEN { resultExpression | NULL }
        [ { WHEN searchExpression THEN { resultExpression | NULL }}...]
        [ ELSE { resultExpression | NULL } ]
    END
  ```

These CASE expressions have the same result:

```
SELECT Tier AS
    CASE WHEN Tier = 'High' THEN 'Priority customer'
    ...

SELECT Tier AS
    CASE Tier WHEN 'High' THEN 'Priority customer'
    ...
```

The following is an example that generates running totals for each ticket status at each tier level:

```
CREATE VIEW VTotal_Tickets AS
    SELECT Tier,
          SUM(CASE Status WHEN Open     THEN 1 ELSE 0 END) AS Opens,
          SUM(CASE Status WHEN Reopen   THEN 1 ELSE 0 END) AS Reopens,
          SUM(CASE Status WHEN Resolved THEN 1 ELSE 0 END) AS Closes,
          ( Opens+Reopens-Closes) AS Pending
        FROM VCustomerTickets
        GROUP BY Tier
```

Tier	Opens	Reopens	Closes	Pending
High	2	0	0	2
Medium	1	2	3	0
Low	1	1	1	1

FROM Clause

Specifies the tables and views from which to build the new view.

```
FROM reference [[ AS ] aliasName ]
  [, reference [[ AS ] aliasName ] … ]
```

Where a reference is:

- Simple reference:

  ```
  tableOrView
  ```

- Join operation (see Join Operations” below, for details):

  ```
  ( reference [ {LEFT | RIGHT} [ OUTER ] | INNER ]
      JOIN reference ON searchCondition )
  ```

- Table expression (see “Table Expressions” on page 327 for details):

  ```
  ( SELECT selectList FROM joinClauses [ WHERE searchCondition ]
       [ GROUP BY groupClause ] [ WINDOW windowClause ] )
  ```

Specifying a single, simple reference creates a view that is a snapshot view of the source table or view. Including more than one reference specifies a *join* operation.

View Constraints

Views have these constraints of the sources to the FROM clause:

- A view may be derived from an event table or another view.
- A view may join an event table or view, and one or more context tables.
- A view may *not* join two or more event streams, or views based on different event streams. (A *Consolidated Event* is a special-case join of event streams.)
- A view may *not* be derived from context tables only.

Join Operations

The C-SQL SELECT supports these join operations:

- Cross Joins
- Inner joins
- Outer Joins
- Nested Joins

Including more than one reference causes a *join* operation. The resulting view is *cross join* unless you either use the JOIN operator to specify another type of join condition or include a WHERE Clause that specifies a join condition

Cross Joins

If you omit the JOIN operator, you define a a view that is a cross join of the input views (also know as a *cartesian product*), such as this join of the *Product* and *Manufacturer* views:

```
FROM Product AS P, Manufacturer AS M
```

Note: DO NOT create a cross join unless you are sure that is what you want. A cross join creates a view whose count of rows is the equal to the count of rows in the first view times the count in the second view ($rows^{join} = rows^{view1} * rows^{view2}$). This severely impacts the system and does not usually produce the view you desire.

Instead of creating a cross join, you should specify another type of join with the JOIN clause:

```
reference [ joinType ] JOIN reference ON searchCondition
```

Inner joins

The JOIN clause performs an inner join unless you specify a joinType. An inner join is one where the rows in the result table are the rows from the first table that meet the specified criteria, combined with the corresponding rows from the second table that meet the specified criteria.

```
FROM (Product AS P INNER JOIN Manufacturer AS M
      ON P.productName = M.ProductName)
```

Note: Inner joins are sometimes called *equi-joins*.

Outer Joins

An outer join is where the rows in the result table are the rows that would have resulted from an inner join plus the rows from the first table (LEFT OUTER JOIN) or the second table (RIGHT OUTER JOIN) that had no matches in the other table. For example:

```
FROM (Product AS P LEFT OUTER JOIN Manufacturer AS M
      ON P.productName = M.ProductName
```

Note: The first table in a LEFT OUTER JOIN and the second table in a RIGHT OUTER JOIN must be an event table or a view; it cannot be a context table.

Resulting columns that do not exist in both references are assigned NULL values.

Nested Joins

Joins can be nested and there is no practical limit on the maximum level of nesting. For example:

```
FROM (Product AS P LEFT OUTER JOIN Manufacturer AS M
      ON P.productName = M.ProductName)
      AS Temp, inventoryContext AS INVvt
```

Table Expressions

A *table expression*, also called an *in-line view*, is a sub-query that creates a view that can be referenced by the containing query. It is essentially a SELECT statement, bounded by parenthesis, and appearing in the FROM clause. For example, the following is a table expression contained in a query:

```
SELECT *
    FROM Warehouse AS wh,
        (SELECT *
            FROM WarehouseQtyChange AS wqc,
                 Product AS pr
            WHERE wqc.wprod_id = pr.pprod_id) AS sv,
    WHERE sv.warehouse_id = wh.wh_region_id
```

The example above has the same result as *WHRegionView* in this example:

```
SummaryView:
SELECT *
    FROM WarehouseQtyChange AS wqc,
         Product AS pr
    WHERE wqc.wprod_id = pr.pprod_id

WHRegionView:
SELECT *
    FROM Warehouse AS wh,
         SummaryView AS sv
    WHERE sv.warehouse_id = wh.wh_region_id
```

Syntax

A table expression is a limited SELECT statement, enclosed in parenthesis, and with restrictions.

```
( SELECT selectList FROM joinClauses
     [ WHERE searchCondition ]
     [ GROUP BY groupClause ]
     [ WINDOW windowClause ]   ) AS aliasName
```

Restrictions

In-line views have the same semantic restrictions as standard views. For example, a derived column in an in-line view cannot have the same name or alias as a column in the containing selectList list. Additionally, they have these restrictions and limitations:

- Must be enclosed in parenthesis.
- Must be in the same event stream (have the same base event table) as the other views and tables in the containing query.
- Must be defined in the FROM clause only.

"HAVING" Example

Some SQL implementations include a HAVING clause that allows you to filter the aggregate results of a view. C-SQL does not include HAVING. However, you can construct a HAVING by using a WHERE Clause to filter the results of an in-line view. Essentially:

```
SELECT * FROM (inlineView) WHERE filterCondition
```

For example, to create a view that filters the result of an aggregation, you first need to perform the aggregation in an in-line view, then filter the results with the containing view. The following in-line view sums the total sales for each product line, and then the containing view displays — in descending order — only those results greater than $1,000,000.

```
SELECT FamilyTotals.family AS "Product Line",
       FamilyTotals.sales_for_family AS "Total Sales"
    FROM (SELECT family,
                    SUM(total_price) AS sales_for_family
                 FROM OrdQtyDemand
                 GROUP BY Family
             ) AS FamilyTotals
    WHERE "Total Sales" > 1000000
    ORDER BY "Total Sales" DESC;
```

In the example above, as new events enter the *OrdQtyDemand* view, the totals are updated and the order of product families can change. In fact, new families can enter the view as their sales totals exceed a million.

WHERE Clause

A WHERE clause examines each row in the input and accepts only those that match the specified condition. The syntax is

```
WHERE searchCondition
```

A searchCondition is a combination of Boolean predicates that together make a test. Only those input row that pass the test are inserted into the new view. Rows that do not meet the condition are discarded, not tracked, and not included in the calculations of a set function, moving set function, or rank function.

Note: All dependant views update and their functions recalculate, regardless of whether or not the input met the condition. See "Updating Views Through Event Propagation" on page 356 for more information.

Predicates

A *predicate* is an Boolean expression that asserts a fact about values. Each expression may be stated alone or compared to one of the Boolean test values. For example, these expressions are equivalent

```
WHERE (Age >= 21)
WHERE (Age >= 21) IS TRUE
```

The predicates that the WHERE clause supports are listed in "Operators and Constants" on page 249.

Predicates may include functions, but functions that reference columns may only reference event columns.

Aliases

C-SQL extends the ANSI standard to permit alias references in the WHERE clause. For example:

```
SELECT user_age AS Age
   FROM user_list
   WHERE (Age >= 21)
```

GROUP BY Clause

The GROUP BY clause groups the resulting virtual table one or more columns. The syntax is

```
GROUP BY [tableName.]columnName [ , [tableName.]columnName … ]
```

Where tableName is a source table or view. All columnNames in the Select List that are not referred directly by a rank function or scalar function must appear in the GROUP BY list.

Note: Another and more powerful way to group data is with the windows PARTITION clause. See "Window Partitions" on page 282 for details.

The Group by option produces summary information for groups of rows whose values in the selected fields are the same. Consider this set of data:

```
Name           Quantity
------------   --------
Nano Webber          10
Fizzy Lifter        700
Nano Webber          50
Nano Webber          20
Nano Webber          15
Smoke Shifter       310
```

If you create a view that groups by Name and determines the sum of the quantity for each group, it would look like the following:

```
SELECT product.name AS Name, SUM(product.quantity) AS Qsum
    FROM product
    GROUP BY product.name

Name           Qsum
------------   ----
Nano Webber      95
Fizzy Lifter    700
Smoke Shifter   310
```

You can also group on multiple fields. For example,:

```
SELECT product.name AS Name, product.location AS Locale,
        SUM(product.quantity) AS Qsum
    FROM product
    GROUP BY product.name, product.location

Name           Location Qsum
------------   -------- ----
Nano Webber    West       10
Fizzy Lifter   East      700
Nano Webber    East       85
Smoke Shifter  West      310
```

When the Select List includes a Moving Set function, each group contains a result for the moving set.

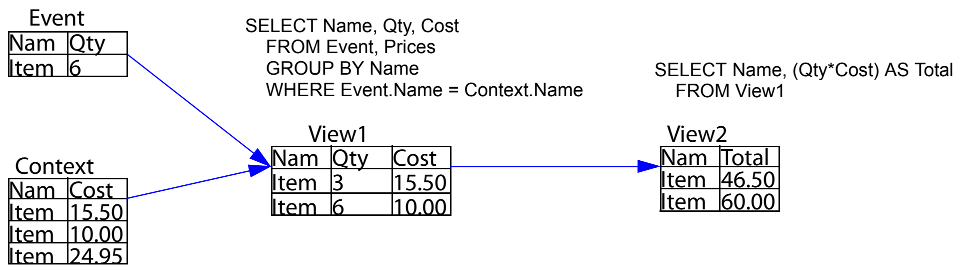
Aliases

C-SQL extends the ANSI standard to permit alias references in the GROUP BY clause. For example:

```
SELECT product.name AS Name, product.location AS Locale,
        SUM(product.quantity) AS Qsum
    FROM product
    GROUP BY Name, Locale
```

Derived Views

When a view is defined with a GROUP BY clause, any view derived from that view has an implicit GROUP BY clause. This is known as *view merging* or *view expansion*. For example, even though the SELECT statement for View2 does not include a GROUP BY clause, its results include the same groups as View1.



```
SELECT Name, Qty, Cost
   FROM Event, Prices
   GROUP BY Name
   WHERE Event.Name = Context.Name
```

```
SELECT Name, (Qty*Cost) AS Total
   FROM View1
```

Stateful View Semantics

When a view contains a GROUP BY clause, that view is a stateful view; it maintains information from previous events, not just the most recent event. (See "Stateless and Stateful Views" on page 356 for details.) As such, View2 in the example above is stateful even though its SELECT definition does not contain a set function or an explicit GROUP BY clause; rather, it is stateful because it is derived from a stateful view.

ORDER BY Clause

The Order BY clause orders (sorts) the resulting view based on column names or on expression results. The syntax is

```
ORDER BY columnName [{ASC|DESC}] [ , [columnName [{ASC|DESC}] ] …
```

Without this clause, there is no guarantee that the same query will produce rows in the same order on subsequent queries.

Note: Any sort key mentioned in the ORDER BY must refer to a column name in the Select List.

By default, the view is ordered in ascending order (ASC). To order in descending order, specify the DESC option.

The following is an example that orders the view first by supplier name in ascending order and then by price in descending order within each supplier:

```
SELECT Product.prod_id AS ProductID,
        orderStatusEvent.OS_PRICE AS Price,
        Supplier.supp_name AS SupplierName
    FROM orderStatusEvent, Product, Supplier
    WHERE orderStatusEvent.OS_PROD_ID = Product.prod_id AND
        Product.prod_supp_id = Supplier.supp_id
    ORDER BY Supplier.supp_name ASC, orderStatusEvent.OS_PRICE DESC
```

31 Users

Each user that interacts with Business Activity Monitoring is known to the system by their user account information. When administrators create or edit accounts, or when users views their Account Settings, they access specific account information from these tabs:

- User Details Tab, which details the attributes that describe a user.
- Delivery Profiles Tab, which specifies how and where a user may receive alerts.
- Access Permissions Tab, which provides access to the permissions assigned to the user.

➤ **In this Chapter:**

System User

Every installation has a default *system* user who is identified during installation and who has all permissions. That user creates other users and assigns permissions. Among the permissions that system user may grant is the ability to create users. See the Release Notes or contact Adobe Systems Incorporated to learn your installation's default system user username and password.

➤ **To edit your own account information:**

- Click Account Settings at any time.

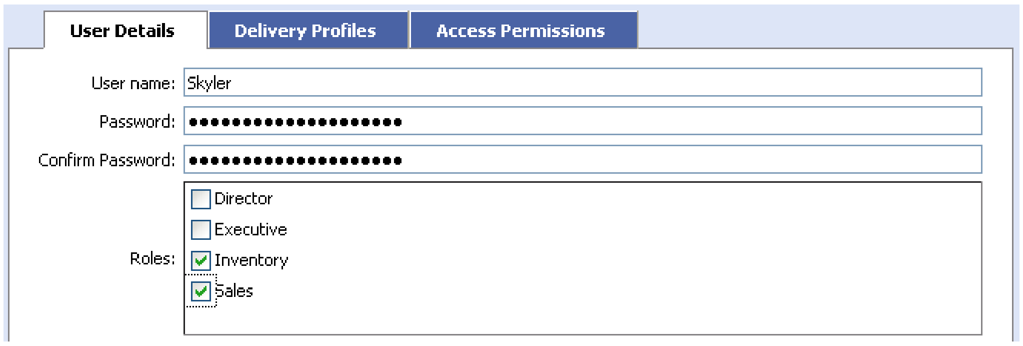
➤ **To create, edit, or delete a user account:**

1. Open the Administration Console.
2. Click Users to see a list of all users currently defined in the system. To
   - Create a new user account, click New User and specify the user's User Details Tab and Delivery Profiles Tab. Optionally, you can specify Access Permissions Tab as well.
   - Edit an existing user's account information by double-clicking the name in the list.
   - Delete an existing user by selecting one or more users in the list and clicking Delete Users. Note that you cannot delete the *system* user or yourself.

User Details Tab

User details identify a user with the following attributes:

Attribute	**Description**
Username	Login name of the user. The name can contain letters and numerals only. This name must be unique among business activities and users; you cannot have a user with the same name as a business activity. See "Object Namespace" on page 248 for details.
Password	User password. Any combination of letters, numerals, or characters.
Roles	Roles that this user may belong too, and which ones the user does belong to. See "Roles" on page 296 for details.

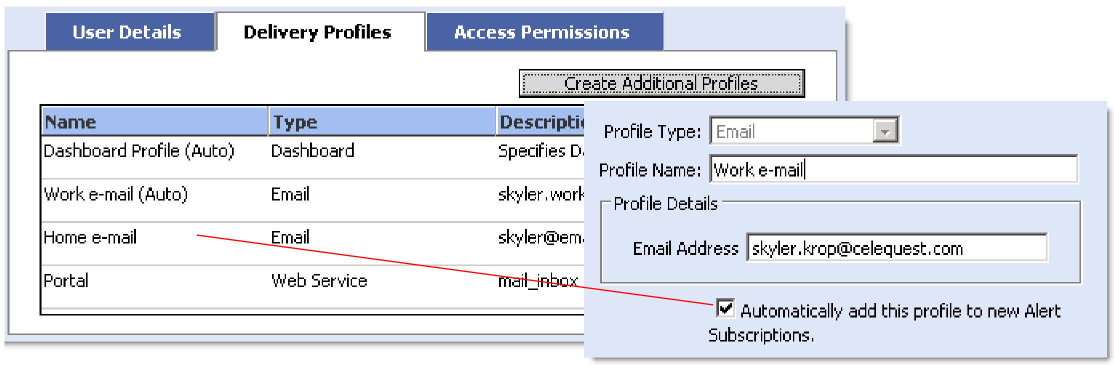


User name: Skyler
Password: ••••••••••••••••••••
Confirm Password: ••••••••••••••••••••
Roles:
- ☐ Director
- ☐ Executive
- ☑ Inventory
- ☑ Sales

Delivery Profiles Tab

Delivery profiles specify where and how to deliver alerts and data feeds to the user. Users may have multiple profiles, and in the Alert Manager in the BAM Dashboard, they can identify which profiles receive which alerts. Further, every user has at least one profile; Dashboard Profile sends notifications to the Alert Manager in the BAM Dashboard.

At least one of the profiles must be designated as the one to use when subscribing to an alert. All profile flagged as Auto are automatically added to new alert subscriptions.



Name	Type	Descripti
Dashboard Profile (Auto)	Dashboard	Specifies D
Work e-mail (Auto)	Email	skyler.work
Home e-mail	Email	skyler@em
Portal	Web Service	mail_inbox

Profile Type: Email
Profile Name: Work e-mail
Profile Details
Email Address skyler.krop@celequest.com
[✓] Automatically add this profile to new Alert Subscriptions.

The profile types are the possible delivery mechanisms available in the installation, and they include:

- Dashboard Profile — The Alert Manager in BAM Dashboard. You cannot delete this profile.
- E-mail — An e-mail account to receive the generated alert message.
- Web Service — A Web Services method that receives the alert notification and attached reportlet as XML data.

E-mail

E-mails is An address where the user receives e-mail messages. It has the following attributes:

Attribute	Description
Profile Name	Identifies this profile. The name can contain letters and numerals only.
E-mail address	E-mail address to use.

Web Service

Web Service receives the alert notification and any attached reportlets as XML data. Web Service has the following attributes:

Attribute	Description
Web service URL	HTTP location of the application providing the DOC (SOAP) service. Note that RPC style messages are not supported.
Method	Method of the Web service to use.
Username	(optional) Account name to use when connecting to the service.

Attribute	Description
Password	(optional) Password for the account.
UDF	(optional) A user-defined function (UDF) used to launch the Web Service. If a UDF is not specified, the default WSDL is used to invoke the Web Service. (See "Working with UDFs and JAR Files" on page 94 in the BAM Workbench documentation and "User-Defined Functions" on page 346 for information about UDFs.)

To use a Web Service, your administrator will need to reference the AlertMessage.xsd and the AlertService.wsdl file which describes the subscription service, the data it provides, and how to exchange data with the service.

Note: This service publishes a SOAP doc-style message, not an RPC style message.

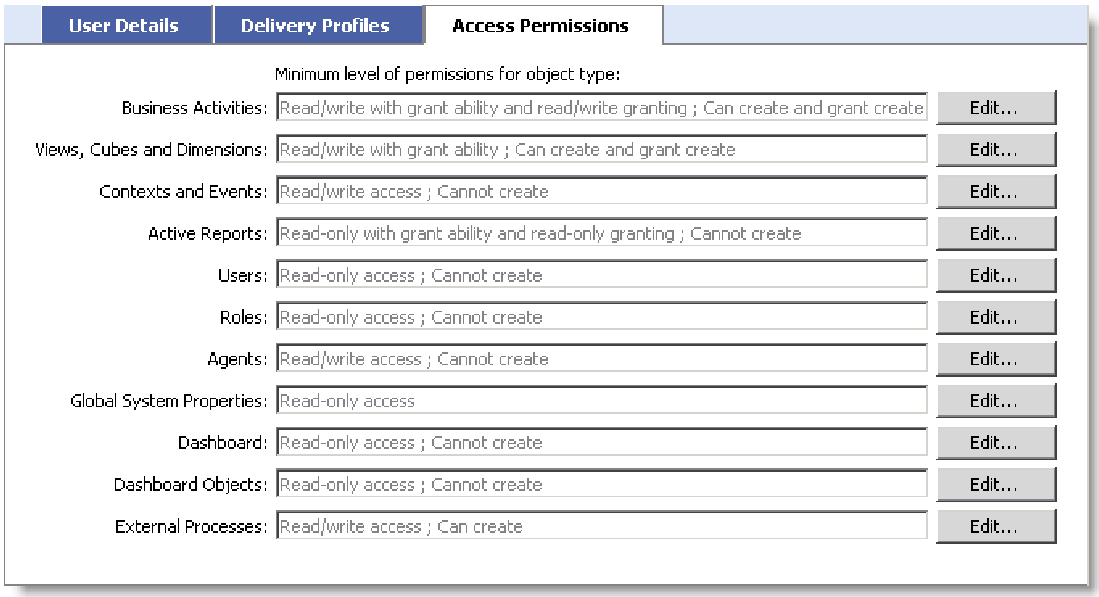
Your administrator can locate the file on Business Activity Monitoring CD-ROM in this directory:

```
BAM/api/wsalert
```

For more information about Business Activity Monitoring use of Web Services, see "Web Services" on page 362.

Access Permissions Tab

Access permissions are the global permissions that a user may have. From this tab, administrators (or anyone who has access to BAM Workbench) can assign the user's permissions to classes of objects. For a detailed discussion about permissions, see "Permissions" on page 253.

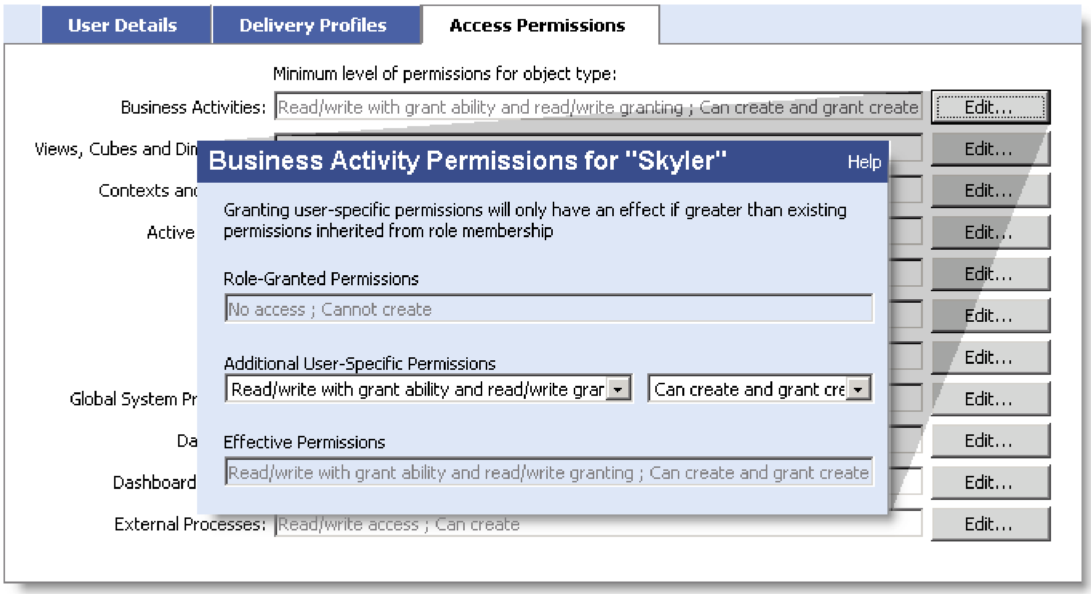


Minimum level of permissions for object type:

Business Activities:	Read/write with grant ability and read/write granting ; Can create and grant create	Edit...
Views, Cubes and Dimensions:	Read/write with grant ability ; Can create and grant create	Edit...
Contexts and Events:	Read/write access ; Cannot create	Edit...
Active Reports:	Read-only with grant ability and read-only granting ; Cannot create	Edit...
Users:	Read-only access ; Cannot create	Edit...
Roles:	Read-only access ; Cannot create	Edit...
Agents:	Read/write access ; Cannot create	Edit...
Global System Properties:	Read-only access	Edit...
Dashboard:	Read-only access ; Cannot create	Edit...
Dashboard Objects:	Read-only access ; Cannot create	Edit...
External Processes:	Read/write access ; Can create	Edit...

Note: Every new user has No Access permissions for everything. This allows the user to receive and view alerts and reportlets as the result of mandatory subscriptions, but they may not subscribe to any.

When an administrator creates or edits permissions, they can specify which objects a user may create and assign permissions to all existing objects.


User Details | Delivery Profiles | Access Permissions

Minimum level of permissions for object type:

Business Activities: Read/write with grant ability and read/write granting ; Can create and grant create — Edit...

Views, Cubes and Dir

Contexts and

Active

Global System Pr

Da

Dashboard

External Processes: Read/write access ; Can create — Edit...

Business Activity Permissions for "Skyler" Help

Granting user-specific permissions will only have an effect if greater than existing permissions inherited from role membership

Role-Granted Permissions
No access ; Cannot create

Additional User-Specific Permissions
Read/write with grant ability and read/write grar | Can create and grant cre

Effective Permissions
Read/write with grant ability and read/write granting ; Can create and grant create

For a detailed discussion about assigning permissions, see “Permissions” on page 253.

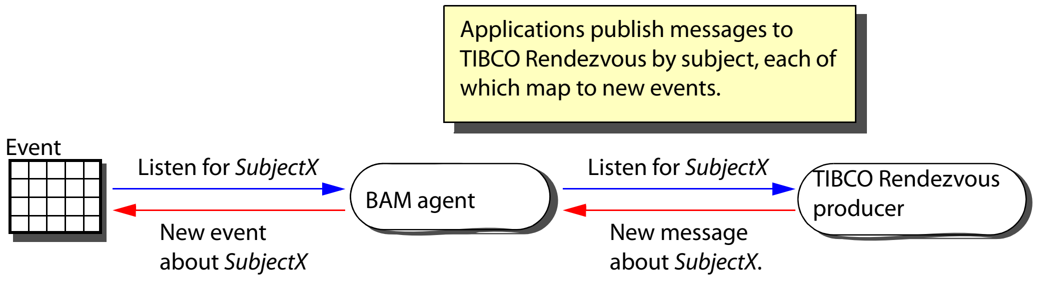
32 TIBCO Rendezvous

TIBCO Rendezvous is a messaging system for business applications. Business applications *publish* messages to the stream managed by TIBCO Rendezvous transport servers. Each message has a name that identifies the *subject* of the message. Other applications monitor the stream looking for messages that, when found, are provided to other applications, such as Business Activity Monitoring TIBCO Rendezvous Agents.

➤ **In this Chapter:**

How It Works

Business Activity Monitoring event tables receive TIBCO Rendezvous messages as events. Each event table corresponds to a single message subject. The tables identify the message subjects to a listening daemon application through the agent. When the daemon locates a new message of the requested subject, it passes the message to the table through the agent. The table definition then maps the message into the table as a new event.


Applications publish messages to TIBCO Rendezvous by subject, each of which map to new events.
Event
Listen for *SubjectX*
New event about *SubjectX*
BAM agent
Listen for *SubjectX*
New message about *SubjectX*.
TIBCO Rendezvous producer

TIBCO Rendezvous Tables

A TIBCO Rendezvous event table receives messages from a business application through a TIBCO Rendezvous message stream. Each message is identified by subject, and each new message for a subject is a new event. When the table receives a new message, it first maps the message data into the event table data types.

Limitations

All messages for an event subject must be in the same form: every message must have the same fields, though a field may be empty. Furthermore,

- Business Activity Monitoring does not support nested messages.
- Some TIBCO Rendezvous data types are not supported and cannot be mapped into a Business Activity Monitoring event. See "Mapping TIBCO Rendezvous Data Types" on page 342 for details.

Prerequisites

Before creating a TIBCO Rendezvous event, you need:

- Permission — Create permission for tables (see "Creating Permission" on page 258) and Read Only access permission on the agent that will feed the table.
- An agent — An existing TIBCO Rendezvous agent that connects to the TIBCO Rendezvous message stream. Create an agent with the BAM Workbench Administration tab. See "TIBCO Rendezvous Agents" on page 344 for details.
- The subject name — Each TIBCO Rendezvous message has a subject name that identifies the event source. You identify the subject name and the agent will monitor the message stream looking for the messages. When it finds one, it passes the message information to the event table. Subject names consist of one or more elements separated by dot characters (periods), such as: SUPPORT.TICKETS
- The format of the message — Each TIBCO Rendezvous message contains fields of information. You tell the event object what fields to extract from the message, and how they map into Business Activity Monitoring data types.

- A sample file — (*optional*) If the message contains a complex string, it is helpful to have a sample file that contains data in the format of the actual event string. You can use this sample when you create the event to ensure that the fields map correctly into the event table by seeing how the data lines up in the columns.

For the details of the subject name and message format, consult the IT specialist who maintains your TIBCO Rendezvous system. A TIBCO Rendezvous event table has the following attributes:

Attribute	**Description**
Name	Identifies the event object. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the table is enabled (monitoring for events) or disabled (not monitoring for events).
Log event data for recovery	When on, logs event data that arrived after the last checkpoint started. This "recovery" log is used to restore the state of the system in the event of an abnormal shutdown of the servers. For complete details, see "Working with Checkpoint and Recovery" in *Using Business Activity Monitoring Workbench*.
Process events in the order of arrival	Choose this option when events must be processed in the order received. Otherwise, if events may be processed out of order, turn on this attribute. **Note:** To join events in a view, the events must be processed in order: leave this option off to join the events.
TIBCO Rendezvous agent	An existing agent that connects to the TIBCO Rendezvous message stream. Create an agent with the BAM Workbench Administration tab. See "TIBCO Rendezvous Agents" on page 344 for details.
Subscription	Identifies the *subject* on which the message is being sent, and defined by the message publisher. Typically, this string looks similar to: com.celequest.mytibcotopic.

Attribute	**Description**
Column Information	The Column Information fields define how to map the fields from the TIBCO Rendezvous message into columns in the event table. There is one column for every field in the event table. See "TIBCO Column Information" on page 341 for details.
Clear State Interval	This tab contains several options for clearing persisted event data that is propagated from the event in the views, objects, and dashboards that depend on it. The three options are: • Do Not Clear State —This is the default. Data persists. • Clear State on a Schedule — Select to clear the state on a schedule. Selecting this option activates the scheduling feature positioned to the right. • Always Clear State (Every Event) — The state is refreshed each time the event is updated.

Creating a TIBCO Rendezvous Event Table

Use the following procedure to create a TIBCO Rendezvous event table.

➤ **To create a TIBCO Rendezvous event table:**

1. Open the BAM Workbench Administration Console.
2. Create a new event.
3. Select TIBCO Rendezvous as the agent type.
4. Identify the Subject of the message events to collect. See Prerequisites above, for details.
5. Select an existing TIBCO Rendezvous agent.
6. Identify the fields in the message, and how they map to Business Activity Monitoring data types. See TIBCO Column Information for details.

Save the TIBCO Rendezvous table as enabled and it will immediately be ready to receive event messages.

TIBCO Column Information

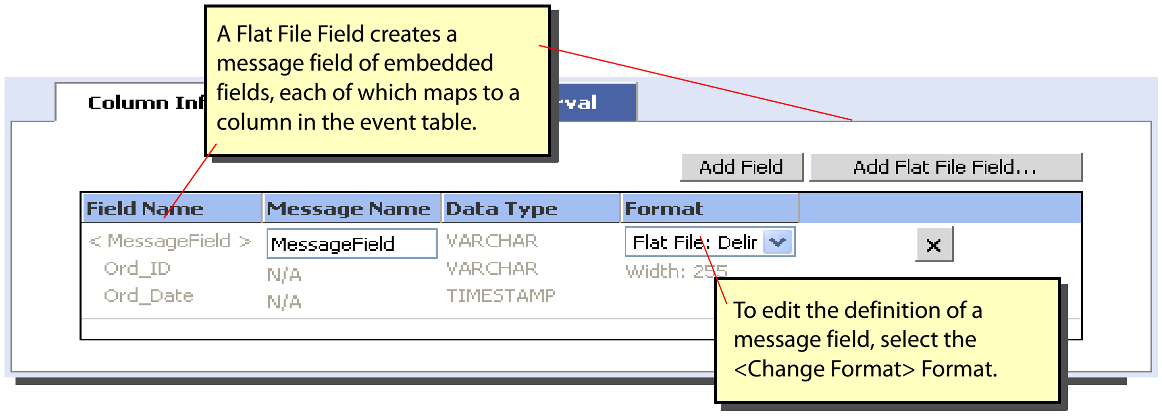
The Column Information fields define how to map the fields from the TIBCO Rendezvous message into columns in the event table. There is one column for every field in the event table.

Each field in the message can be a simple field that maps directly into a event column, or it can be a complex field (a *flat file field*) that contains several fields that each map into columns in the table. Complex fields are treated as Flat Files in either delimited (CSV), fixed-width, or XML formats. See "Flat Files" on page 81 for detailed descriptions of these file types.

Each column in the event table has the following attributes:

Attribute	**Description**
Field Name	Name of the column in the event table.
Message Name	Name of the field in the message. When mapping a MessageField, the name for each embedded field is N/A and uneditable.
Data Type	Data type of the event column. See "Mapping TIBCO Rendezvous Data Types" on page 342 for details.
Format	(optional) Format of the event column for VACHAR (string) and DECIMAL values.

Add columns by clicking Add Field or Add Flat File Field.


A Flat File Field creates a message field of embedded fields, each of which maps to a column in the event table.
Column Inf
val
Add Field
Add Flat File Field...
To edit the definition of a message field, select the <Change Format> Format.

Field Name	Message Name	Data Type	Format
< MessageField >	MessageField	VARCHAR	Flat File: Delir
Ord_ID	N/A	VARCHAR	Width: 255
Ord_Date	N/A	TIMESTAMP	

➤ **To add a message field:**

1. Click Add Flat File Field.

2. Choose the flat file type of the message field.

   (Optional for fixed-width and delimited files.) Identify a sample file to assist in mapping the columns. This file is a sample of the real data file. Data from this file appears in the next step to assist you as you map the event data into the table.

3. For fixed-width files, define the positions of the data columns with the Set Field Widths dialog.

4. Identify the flat-file attributes. See "Flat File Event Tables" on page 82 for details.

5. Define the format-specific Column Information. For details about the source type, see:
   - "Fixed-Width Files" on page 94.
   - "Delimited Files" on page 93
   - "XML Files" on page 95

6. Click Save Event to save the message field definition.

➤ **To edit the definition of a message field:**

In the field's Format column, change the value from "Flat File: *file type*" to "<Change Formatting>".

Note: When editing a message field, the sample file option for delimited and fixed-width file types is not available.

Mapping TIBCO Rendezvous Data Types

Each message is a set of fields that each contain one data item of a specific data type. You identify each field by its defined name and specify the field's associated Business Activity Monitoring data type.

The TIBCO Rendezvous data types map to Business Activity Monitoring Data Types as follows.

TIBCO Rendezvous Type	Business Activity Monitoring Type
Custom Data Types	Not Supported
BOOL	Boolean
DATETIME	Timestamp
F32	Double
F32ARRAY	Not Supported
F64	Double
F64ARRAY	Not Supported
I16	Integer
I16ARRAY	Not Supported
I32	Integer
I32ARRAY	Not Supported
I64	Decimal
I64ARRAY	Not Supported
I8	Integer
I8ARRAY	Not Supported
IPADDR32	Not Supported
IPPORT16	Not Supported
MSG	Not Supported
OPAQUE	Not Supported
STRING (see note below)	Varchar
U16	Integer
U16ARRAY	Not Supported
U32	Decimal
U32ARRAY	Not Supported
U64	Decimal
U64ARRAY	Not Supported
U8	Integer

TIBCO Rendezvous Type	Business Activity Monitoring Type
U8ARRAY	Not Supported
XML	Varchar

TIBCO Rendezvous Agents

A TIBCO Rendezvous agent communicates with a TIBCO Rendezvous daemon running in the application server environment. The daemon listens for messages on a TIBCO Rendezvous message stream. When the daemon finds a message requested by one of the TIBCO Rendezvous Tables, it retrieves the message data and passes it to the table through the agent.

Note: TIBCO Rendezvous agents are asynchronous, they receive event messages as the events occur as summarized in the following table. You *cannot* retrieve context from a TIBCO Rendezvous agent.

Event push	Event pull	Context pull
Yes	No	No

Prerequisites

Before creating an agent, you need to:

- Create permission for agents (see "Creating Permission" on page 258 for details).
- Connect to the TIBCO Rendezvous listener daemon.

Note: To connect, you need the Service, Network, and Daemon names. For specific values, consult the IT specialist who maintains your TIBCO Rendezvous system .

Attributes

A TIBCO Rendezvous agent has the following attributes:

Attribute	Description
Name	Identifies the agent. This name must be unique among agents. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Service	TIBCO Rendezvous service port. Leave this blank to use the default port 7500. Change this value only if your TIBCO Rendezvous administrator gives you another port.
Network	Identifies the network interface to use when the host is connected to more than one network, or when the host supports multicasting (in which case the address will look similar to ;222.1.2.3). Change this value only when the host machine is not on the default network, then use the IP address provided by your TIBCO Rendezvous administrator.

Attribute	**Description**
Daemon	Port of the routing daemon on the TIBCO Rendezvous host found on the network and identified by the Network attribute. Leave this blank to use the default port 7500. Change this value only if your TIBCO Rendezvous administrator gives you another port.
Status	Whether or not the agent is enabled (monitoring for events) or disabled (not monitoring for events).

Creating a TIBCO Rendezvous Agent

Use the following procedure to create a TIBCO Rendezvous agent.

➤ **To create a TIBCO Rendezvous agent:**

1. Open the BAM Workbench Administration Console.
2. Click New Agent…
3. Choose TIBCO Rendezvous as the source type
4. Fill in the fields that define the agent’s attributes.ef

Save the agent as enabled and it will immediately begin monitoring for events.

33 User-Defined Functions

User-defined functions (UDFs) provide a mechanism for extending C-SQL by defining your own functions for use in queries, views, and rules. With this feature you can define a scalar function or set function by implementing the appropriate Business Activity Monitoring Java interfaces.

UDFs are Java programs that take arguments and return a value, just like the internal Business Activity Monitoring functions. For example, you might have a UDF that takes a set of values and concatenates them alphabetically while ignoring NULL values. You would use that UDF in an expression like this:

```
ConcatSet(Product.Name)
```

After compiling the Java program, you deploy (load) it into Business Activity Monitoring where it is then available to all users who can create or edit queries, views, and rules.

For information about creating user-defined functions, see Creating and Using a UDF," below. For further details about the interfaces, see "com.celequest.api.function" in the Javadoc documentation. You can access the Javadoc documentation with a Web browser directly from Business Activity Monitoring CD-ROM by pointing your browser to: <cd>/helpdocs/javadoc/index.htm.

➤ **In this Chapter:**

UDF Restrictions

User-defined functions have these restrictions:

- Scalar and set functions only; no rank functions.

  UDFs can define Scalar functions by implementing the IUDScalarFunction interface, or Set functions by implementing the IUDAggregateFunction and IUDAggregateState interfaces. See "com.celequest.api.function" in the Javadoc documentation for details.
- UDFs are not exposed to the BAM Workbench.

  The formula editor does not recognize UDFs. As such, you are responsible to keeping track of which UDFs are registered with the system.
- All users have access to all UDFs.

  You cannot apply access permissions to a UDF. Similarly, multiple users may define different UDFs with the same name. In that case, the system uses the first on it finds.
- Set functions must implement object serialization and maintain backward compatibility.

  Failure to implement meaningful serialization/deserialization routines may result in unpredictable behavior in many areas including checkpoint and recovery, and parallel execution.
- UDFs are never pushed as predicates to a remote source.

  Essentially, UDFs are never sent to a DBMS for evaluation. See "Context Column Limitations in Queries" on page 46 for more details.

Creating and Using a UDF

These instructions use the ConcatSet sample UDF included on the product CD-ROM in the /samples/udf/ directory. To follow along using the sample, first copy the contents of that directory to a location on your local machine. See the README.txt file in that directory for more information about the sample.

➤ **To create and use a UDF:**

1. Implement the interfaces.
2. Create your UDF by implementing the appropriate com.celequest.api.function interfaces.

   Note: See the Javadoc documentation for details about the interfaces.
3. Compile your implementation.

   When you compile your UDF implementation, include the cognosAPI.jar file. (The file in on the product CD-ROM in the /BAM/CognosAPI/ directory.) For example:

   ```
   javac -classpath <CD-ROM>/BAM/cognosAPI/cognosAPI.jar
   src/samples/udf/concatset/*.java src/samples/udf/concatlist/*.java
   src/samples/udf/util/*.java -d jar
   ```
4. Create a manifest for the UDF JAR file.

   A manifest is an XML file that describes the UDF JAR file to Business Activity Monitoring. For a description of the file and a sample listing, see "Manifest Files" on page 348.

5. Create the UDF JAR:

   Create a JAR file containing your class(es) (such as ConcatSet.class) and the manifest (manifest.xml). The manifest must appear under com/celequest/manifest in the jar. For example:

   ```
   jar -cvf udf.jar samples/* com/celequest/manifest/manifest.xml
   ```

6. Add the UDF to the list of available UDFs.
   - Log in to the BAM Workbench and open the BAM Workbench tab.
   - Select the User Defined Functions folder and choose New User Defined Function.
   - Specify or load the JAR file that contains the function, and click **Continue**.
   - Select the functions to add, and optionally assign new names to them.
   - Choose Finish to add the UDFs to the list.

7. Use the UDF in formulas. Once the UDF is deployed, you can use it in formulas, similar to this:

   ```
   ConcatSet(Product.Name)
   ```

This completes the steps for creating and using a UDF.

➤ **To alter an existing UDF:**

1. Change the implementation and create an updated JAR file.
2. Upload the JAR over the existing one.
3. Re-add the UDFs to the list of User Defined Functions.

Manifest Files

A manifest is an XML file that describes the contents of the UDF JAR to Business Activity Monitoring. For every function in your jar file, define <UDF> and <name> elements. Further, for every data-type that the function can return, define a <UDFDescriptor> element. Finally, if subsequent calls to the same function with the same argument values can return different values, set the <isVariant> element to true. For example, the CURRENT_TIMESTAMP() internal function takes no arguments but returns a different result each time it is called. That function is "variant".

The following is a sample manifest for the ConcatSet() function. This listing is adapted from the file in the samples directory at /samples/udf/jar/com/celequest/manifest.xml.

Sample manifest.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<jarManifest xsi:schemaLocation="http://www.celequest.com/3
  jarManifest.xsd" xmlns="http://www.celequest.com/3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <jarManifestXSDVersion>1</jarManifestXSDVersion>
    <author>Adobe Systems Incorporated</author>
    <UserDefinedFunctions>
        <UDF>
            <name>ConcatSet</name>
            <description>Concatenate a set of values alphabetically into a
large string. Ignores nulls.</description>
            <implementor>com.udfcelequest.concatset.ConcatSet</implementor
>
            <UDFDescriptors>
                <UDFDescriptor>
                    <result>VARCHAR</result>
                    <argument>
                        <type>ANY</type>
                    </argument>
                </UDFDescriptor>
            </UDFDescriptors>
            <isVariant>false</isVariant>
        </UDF>
    </UserDefinedFunctions>
</jarManifest>
```

34 Views

Business views are data models that provide a real-time picture of a business activity. Records of changes and transactions in your business enter Business Activity Monitoring as events. Each new event drives an immediate update of the views — the business models — derived from that event, thereby providing a real-time picture of the business metrics. Further, after a view has been updated, the system evaluates the rules associated with the view looking for exceptional business conditions that require attention.

A business view is a virtual table that resides in memory and whose contents come from one or more sources as defined by a C-SQL SELECT query statement. You define views with the graphical user interface in the BAM Workbench. That system then constructs a well-formed SELECT statement before passing it to the Business Activity Monitoring servers for execution and maintenance. For details about the SELECT statement and its syntax and usage, see "SELECT" on page 321.

You can limit the rows that a user sees by associating an access filter to the view, and applying the filter to users or roles that see the view. See "Access Filters" on page 15 for complete details.

Note: The data in the context views are static or slowly changing. As such, the query engine does not update the view due to changes in the context tables. Changes in the context table are, however, reflected in the view when the context table is joined with the next event row.

➤ **In this Chapter:**

Creating Views

You need Create permission for views (see "Creating Permission" on page 258) and Read Only access permission on the table (and optional views) that will feed the new view.

➤ **To create views with the BAM Workbench:**

1. Click the New View button.

2. Select the existing event, view, or views on which to build your view.

3. Fill in the fields in the Create View form.

➤ **To create views for a specific events with the BAM Workbench:**

1. Select the existing event, view, or views on which to build your view.
2. Click the New View For This Event button.
3. Fill in the fields in the Create View form.

Copying a View

You can also copy the definition of an existing view to a new view.

➤ **To clone an existing view:**

1. Edit the view you want to copy.
2. Change the view name, and change the other attributes that differ from the original view.
3. Choose “Save as New View”.

View Attributes

Every view has the following attributes:

Attribute	Description
Name	Identifies the view object. The name can contain letters and numerals only. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Status	Specifies if the view is enabled (receiving new event information) or disabled. When an view is disabled, all objects that depend on the view are also disabled, including rules, alerts, and reportlets.
Description	Optional description that may contain any text characters.
Workset	Event table and or views on which the view is derived.
Field list	Columns to include in the view, including columns whose values are derived from formulas. This is the Select List in the underlying SELECT statement. The Group By option identifies columns on which to group the results. This is the GROUP BY Clause in the underlying SELECT statement.
Maintain in view	Allows tracking of past event information for stateless views. See "Maintaining Events in Stateless Views" on page 358 for details.
From Clause	How to join the information from multiple tables and views in the Workset. This is the FROM Clause in the underlying SELECT statement.
Where Clause	Identifies which source information to include in the new view. Events which do not meet the specification are not included in the view. This is the WHERE Clause in the underlying SELECT statement. Note that even though an event's information might be discarded, derived views will still update, though they too will not contain the event information. See "Updating Views Through Event Propagation" on page 356 for details.
Window Clause	Defines windows for aggregating sets of rows in the view. See "Query Windows" on page 272 for details.
Order by Clause	Sorts the resulting view based on column names or on expression results. This is the in the underlying SELECT statement.
View Persistence	Save view data to a database for later analysis. See "Persisting Views to a Database" on page 359 for details.
Drill Back to Detail	Enables the ability for users to see the details about the data presented by a dashboard object. See "Enabling Drill Back to Detail" on page 361 for details.

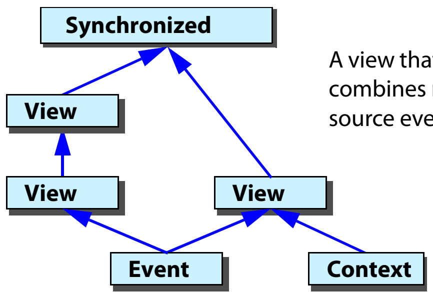
View Constraints

Business Activity Monitoring views have these constraints:

- A view may be derived from an event table or another view.
- A view may *not* join different event steams. (Though a *Consolidated Event* is a union of two identical event streams; see "Consolidated Events" on page 355 for details.)
- A view may join two views of the same event stream. (A *synchronized join* is a view derived from multiple views based on the same event stream; see "Synchronized Joins" on page 353 for details)
- A view may join an event table and one or more context tables.
- A view may join a view and one or more context tables.
- A view may *not* be derived from context tables only.

Synchronized Joins

A *synchronized join* is a view derived from two views based on the same originating event stream. In a synchronized join, the resulting view contains rows that are the combination of the *same events* in the source views.



A view that is a synchronized join combines rows from the same source event.

Synchronized joins are always based on the event's internal ID; there is an implicit join condition on the internal event column. However, you still should define a WHERE clause or join condition in the FROM clause to avoid a possible cross join result (see "Cross Joins" on page 326 for details).

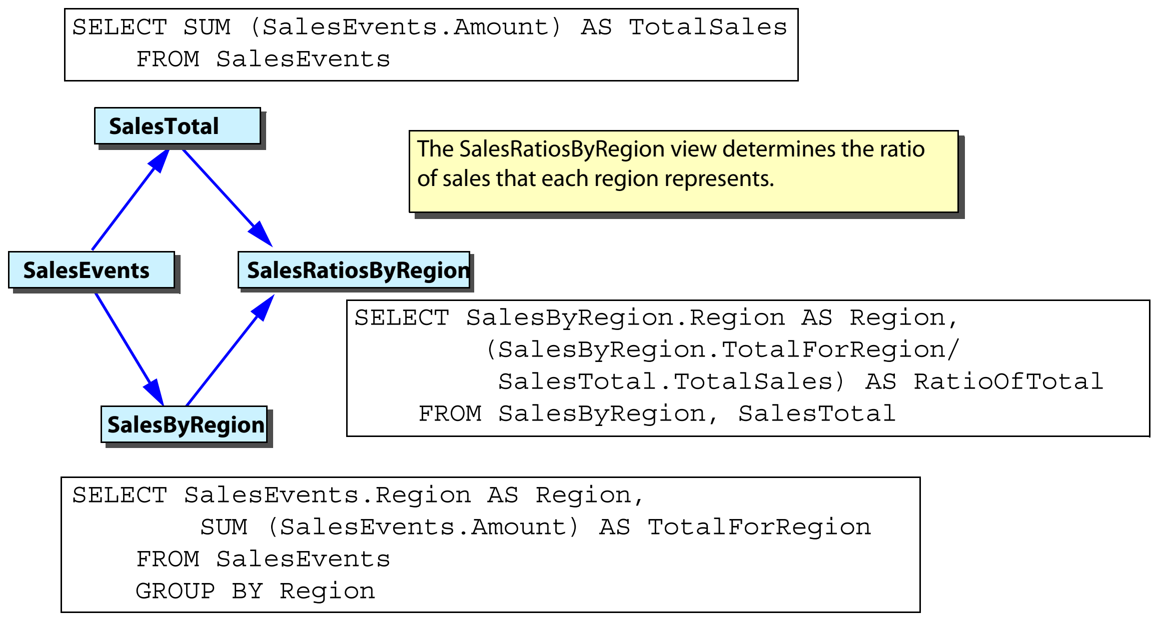
Note: A synchronized join stream always results in a stateless view.

Restriction

A synchronized join requires that events be processed in the order that they arrive. As such, the source event object must have Process events in the order of arrival turned on.

Example

One example of a synchronized join is one shown in the following illustration which determines the ratio of total sales by region. One view (*SalesTotal*) determine the total sales for all events, while the other (*SalesByRegion*) determines the totals for each region. Finally, the synchronized join (*SalesRatiosByRegion*) determines the percentage of each region by joining the two “total” views and dividing the region totals into the grand total.



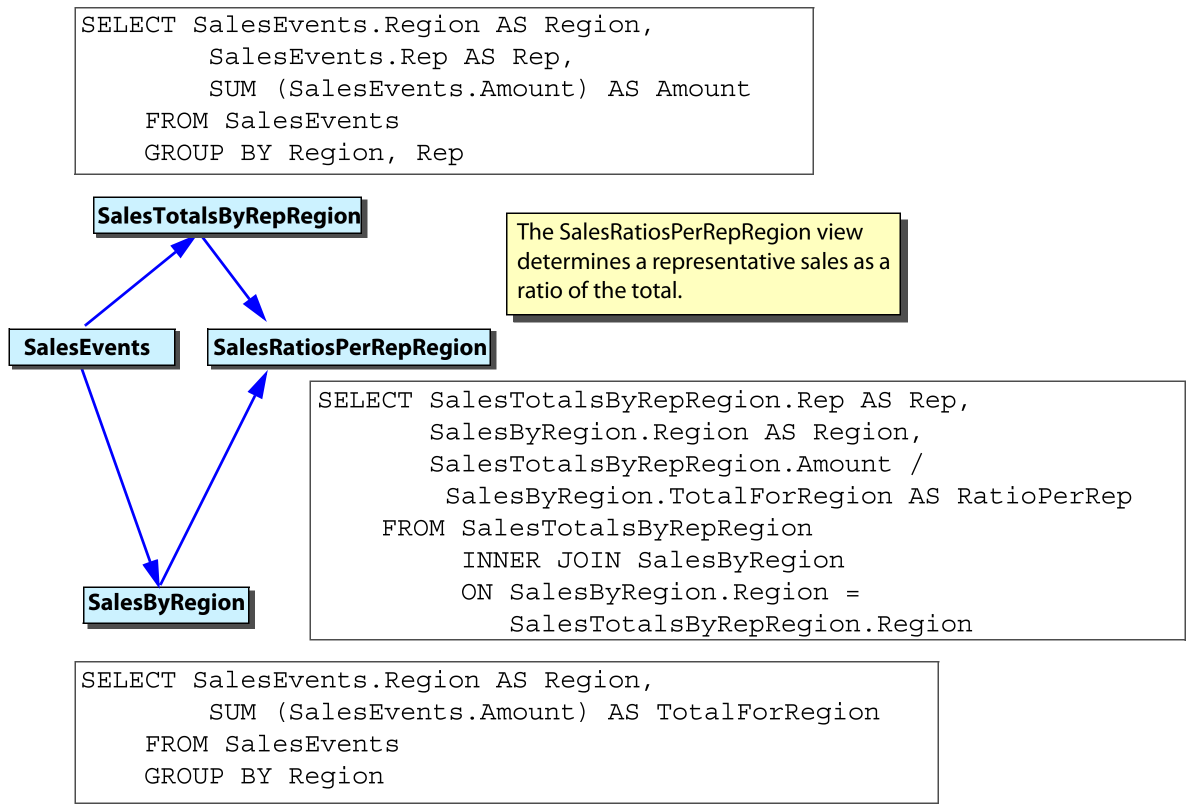
```
SELECT SUM (SalesEvents.Amount) AS TotalSales
    FROM SalesEvents
```

The SalesRatiosByRegion view determines the ratio of sales that each region represents.

```
SELECT SalesByRegion.Region AS Region,
        (SalesByRegion.TotalForRegion/
         SalesTotal.TotalSales) AS RatioOfTotal
    FROM SalesByRegion, SalesTotal
```

```
SELECT SalesEvents.Region AS Region,
        SUM (SalesEvents.Amount) AS TotalForRegion
    FROM SalesEvents
    GROUP BY Region
```

Note that the above synchronized join is a cross join, which in this case is acceptable. However, consider this next example that determines each sales representatives ratio relative to the total sales:



```
SELECT SalesEvents.Region AS Region,
        SalesEvents.Rep AS Rep,
        SUM (SalesEvents.Amount) AS Amount
    FROM SalesEvents
    GROUP BY Region, Rep
```

The SalesRatiosPerRepRegion view determines a representative sales as a ratio of the total.

```
SELECT SalesTotalsByRepRegion.Rep AS Rep,
       SalesByRegion.Region AS Region,
       SalesTotalsByRepRegion.Amount /
        SalesByRegion.TotalForRegion AS RatioPerRep
    FROM SalesTotalsByRepRegion
         INNER JOIN SalesByRegion
         ON SalesByRegion.Region =
            SalesTotalsByRepRegion.Region
```

```
SELECT SalesEvents.Region AS Region,
        SUM (SalesEvents.Amount) AS TotalForRegion
    FROM SalesEvents
    GROUP BY Region
```

In the above illustration, you do not want a cross join because it creates one row for each sales representative in every region, including the regions that the representatives do not belong too. Instead, the SalesRatiosPerRepRegion view declares an inner join to limit the results by region.

Consolidated Events

Consolidated events are special views that accept events from two different event streams. For a detailed discussion, see "Working with Consolidated Events" in *Using Business Activity Monitoring Workbench*.

Aggregate Views

A key power of business views is the ability to aggregate event and context information to extract, analyze, and combine the information into meaningful business metrics. Aggregate views have at least one field definition that includes an aggregation or GROUP BY Clause.

For example, consider this simple view that tracks the total count of events that arrived in the last hour. Every time a new event arrives the MOV_COUNT() function recalculates the count of all events in the view, thereby providing a real-time metric about the event stream:

```
SELECT MOV_COUNT( *, HOUR, 1) AS "Events in the last hour"
    FROM Events
```

You can make the previous example a little more complex by limiting the events that the view sees. For example, to count only those events whose *Status* value is Open:

```
SELECT MOV_COUNT( *, HOUR, 1) AS "Opens in the last hour"
    FROM Events
    WHERE Status='Open'
```

By combining the aggregate information with other context, you can generate more meaningful metrics. For example, this view reports the count of events whose *Status* value is Open, and groups them by Feature:

```
SELECT Context.Topic AS Topic,
       MOV_COUNT( *, HOUR, 3) AS "Opens in 3 hours"
    FROM Events INNER JOIN Context ON Event.Feature = Context.Feature
    WHERE (Status='Open')
    GROUP BY Feature

Feature        Opens in 3 hours
-------------- ----------------
Configuration  12
Install         3
Servers         6
```

For detailed information about aggregate and moving set functions, see "Function Types" on page 109.

Updating Views Through Event Propagation

When a view receives a new event, it attempts to update itself with the new information, and if the update occurred, the view then notifies all dependant views to also update. However, these exceptions can keep the view from updating:

- If a view is empty when it receives an event, and it remains empty after processing the event, it never notifies the dependant views.
- If a stateful view becomes empty as a result of a deletion, such as when an existing event is discarded from a moving set function set, all dependant views are notified to update as well.
- An update may cause a stateless view to become empty, and any dependant views will also be empty. If subsequent event also results in an empty view, the view will appear to have not updated, even though it did.
- If the new event is discarded because it does not meet some criteria, the stateful view is not updated. However, it still publishes a snapshot of itself to all dependant views which can cause dependant moving set functions to update their views.

If an error occurs when processing an event while updating a view, all rows related to the entire event are discarded, and the view remains valid and enabled.

Stateless and Stateful Views

All views in Business Activity Monitoring are either *stateless* or *stateful*:

- Stateful views contain the results of aggregations derived from past events in a single row. A view is stateful if it
  - contains an set function, or moving set function in the SELECT clause, or
  - contains a GROUP BY Clause (in which case each group contains only one row), or
  - is derived from a stateful view.
- Stateless views are any views that are not stateful.

Generally, a stateless view shows the information about a single event, such as a single purchase order. A stateful view, on the other hand, shows the aggregate information about multiple events, such as the average price of multiple purchase order events.

View Initialization

Each view maintains two snapshots of the data it contains:

- Current view — This is the data currently in the view. For a stateful view, the snapshot shows all rows in the view. For a stateless view, it shows all the rows corresponding to the last event, which after aggregation might be an empty set.
- Recent view — A snapshot of the last *non-empty* current view. When the view has a moving set window, the recent view contains the last non-empty rows in the window. For example, if the window is 2 days, the recent view contains the last 2-day set that was not empty, event if the current view is empty. A recent view is what appears in the BAM Workbench when editing an object that displays view results, and what is used by derived views during view initialization.

When you create or enable a view, it is initialized to a state based on the data in the base view as follows. When a view is derived from a

- stateful view, the new view is initialized with the data in the base view's *current view* snapshot. For example, consider a stateful base view which tracks sales by region:

  ```
  SELECT region, SUM(sales) AS region_sales GROUP BY region
  ```

  When you derive a new view from *sales_by_region*, the new view is immediately populated with the data in *sales_by_region*'s current view.

  ```
  SELECT SUM(region_sales) AS total_sales FROM sales_by_region
  ```

- stateless view, the new view is initialized with the data in the base view's *recent view* snapshot. For example, consider this stateless view. This view's current snapshot is empty when no sales are greater than 1,000,000.

  ```
  SELECT region, sales WHERE sales > 1000000
  ```

  However, a view derived from this view will be initialized with the data in the view's recent view snapshot
- event table, the new view is empty; event tables do not maintain snapshots.

Maintaining Events in Stateless Views

By default, a stateless view contains only rows representing the last event and which satisfied the view condition; rows from previous events are discarded. If the event did not meet the condition, the view will be empty. With the Maintaining Events in a Stateless View option you can specify a set of recent *non-empty* event information to maintain in the view.

Use this option to include the recent events in

- The BAM Workbench when displaying a the view's contents on the Results tab. This tab displays the current rows in the view.
- External applications that receive the view as a real-time data feed. This allows the external application to perform trend or historical analysis.

➤ To maintain events in a stateless view:

1. Open the View Editor on a view to persist.
    - On an existing view choose Edit View, or
    - When creating a new view…
2. Check Maintain in view.
3. Enter either the count or time-span of events to retain.
    - An event count is the maximum number of *non-empty* events to maintain. The view discards the oldest event rows that do not fit in the specified size.
    - A time interval defines a set of the most recent events. The count of events in the view varies depending on the number of events in the interval when the view was updated. For example, if an event arrived that did not meet the view criteria, it is excluded from the view, but the view recalculates the interval that time.

    Note: The set of events is determined *when the last event was inserted*, not at the current time. For example, an interval of one hour shows all the events that arrived in the view for the hour previous to the last update. If no events were inserted in the last day, the view might still show an hour's worth of events from the previous day. However, as soon as new event arrives at the view, all those events are discarded.

See also "Moving Set Functions" on page 150 for a means of performing aggregations on sets of recent events.

Persisting Views to a Database

Business Activity Monitoring can persist business view data to an external DBMS for future reporting by third party tools. The information in the table is sufficient for the reporting tools to recreate a complete snapshot of the view. When persisting, the view information is written to a table in the DBMS at a rate following a policy that you define.

Note: Business Activity Monitoring can create the table automatically, or you can predefine the table in the DBMS. See the table in "View Persistence Attributes" below for details.

➤ **To activate view persistence:**

1. An application specialist must have first define a JDBC Agent to the RDBMS that will store the view data. See "JDBC Agents" on page 224 for details about creating this agent.
2. Open the View Editor on a view to persist.
   - On an existing view, choose Edit This View...
   - When creating a new view, choose New View...
3. Select the Other tab.
4. Under View Persistence, Click Define…
5. Fill in the fields in the Define Database Connection dialog. (See the table under "View Persistence Attributes.")

The view begins persisting data as soon as it is enabled.

View Persistence Attributes

The database connection attributes include:

Attribute	Description
JDBC Agent	Agent to the RDBMS defined in step 1 above.
Table	Target table in the RDBMS to receive the persisted data. If you omit this name, the table will have the same name as the business view. The columns in the target table must have the same names as the columns in the view, appear in the same order as in the view, and they must be at least the same width as the columns in the view. When a column width in the target is smaller, the RDBMS either silently truncates the data to fit or generates an error. Similarly, the target table must support row lengths at least as long as the rows in business view.
Create this table if it isn't there	Indicates that the application specialist or database administrator has not already created the named table. If this option is selected, Business Activity Monitoring attempts to create the target table using a CREATE TABLE command in the JDBC user's default table space.

Attribute	Description
Persistence Policy	Persistence depends on whether the view is stateless or stateful. Stateful views are written as snapshots that are persisted based on the specified interval, and only the snapshots are persisted. Stateless views are written as snapshots as well but are also logged so that the condition between snapshots is captured as well.
Number of Events	Specifies the number of events to write to the database. For example, if the number of events is set to 10, then 10 events must occur before they are written to the database. Setting the events to 1 will cause a write with every event.
Time interval	Specifies the interval at which to write to the database.
Stop logging after...	Specifies how many consecutive errors to write to the error log before disabling persistence. This option prevents the server from repeating the same error.

View Columns to Persist

The database receives all of the columns and rows currently in the view. Additionally, each row contains these additional internal columns:

Column	Description
VC_EVENT_ID	Event identifier identifies the event that produced the most recent row included in the view.
VC_LATEST_EVENT_ID	Latest event identifier identifies the last event that caused the view to update, though data from that event might not be included in the view.
VC_TIMESTAMP	Event timestamp identifies when the last event was included in the view.

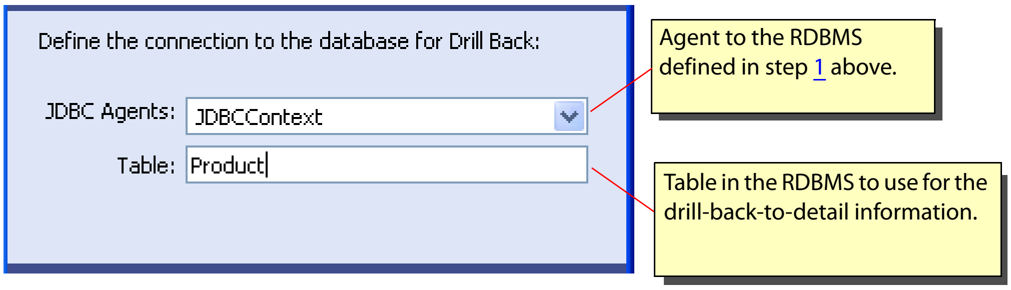
Enabling Drill Back to Detail

You can create views that make it possible to drill back to detail on a chart. This allows users to see the details about the data presented by a dashboard object. "Drill Back to Detail" has the following requirements:

- The data source of the dashboard object must be a cube.
- The view on which the cube is build must be stateless (see "Maintaining Events in Stateless Views" on page 358).
- The dimensions of the cube must come from a single external database and the database should be the same as the measures for the cube.
- If there is more than one table in a view, you can use a persisted table for the drill back to detail table. (see "Persisting Views to a Database" on page 359).

➤ **To activate drill back to detail:**

1. An application specialist must have first defined a JDBC Agent to the RDBMS that will store the view data. See "JDBC Agents" on page 224 for details about creating this agent.
2. Open the View Editor on a view for which you want to enable Drill Back to Detail.
   - On an existing view, choose Edit This View...
   - When creating a new view, choose New View...
3. Click the Advanced tab and under Drill Back to Detail area, click Define…
4. Fill in the fields in the Define Drill Back to Detail dialog.


Define the connection to the database for Drill Back:

JDBC Agents: JDBCContext

Table: Product

Agent to the RDBMS defined in step 1 above.
Table in the RDBMS to use for the drill-back-to-detail information.

5. Click OK.
6. Save the view.

35 Web Services

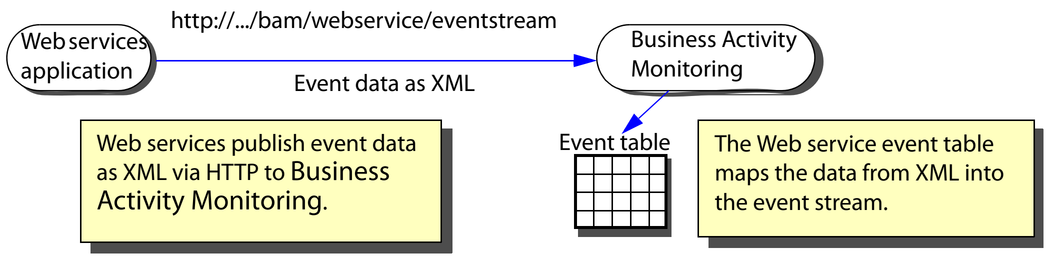
A Web service is an interface to an application running on a Web application server. The service can be a simple database lookup script, or a complex enterprise application integration (EAI) product, like those provided by Siebel or SAP. Business Activity Monitoring connects to Web services to do the following:

- Receive events (as describe in "Web Service Events" on page 363),
- Retrieve context (as described in "Web Service Context" on page 366),
- Publish alert messages:
    - to a subscriber's delivery profile (see "Delivery Profiles Tab" on page 334 for details)
    - on an individual basis as initiated by a user viewing the message in the BAM Dashboard ("Web Service External Processes" on page 371 for details).

➤ **In this Chapter:**

Web Service Events

Web services publish event data as XML through HTTP directly to the Business Activity Monitoring servers. All Web service publishers use the same URL. As such, encoded in XML with the event data is the name of the event to receive the data as well as the Business Activity Monitoring account that has access to the event table. When the servers receive the event data, they parse the XML, decode the data, and insert it into the identified event stream.


Web services application
http://.../bam/webservice/eventstream
Event data as XML
Business Activity Monitoring
Web services publish event data as XML via HTTP to Business Activity Monitoring.
Event table
The Web service event table maps the data from XML into the event stream.

Web Service Event Attributes

Each Web service event table has the following attributes:

Attribute	**Description**
Name	Identifies the table and is the name accessed by the Business Views that depend on this table. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	(*optional*) Description of the table.
Status	Whether or not the object is enabled (able to receive and pass data) or disabled (not receiving or passing data).
Log event data for recovery	When on, logs event data that arrived after the last checkpoint started. This "recovery" log is used to restore the state of the system in the event of an abnormal shutdown of the servers. For complete details, see "Working with Checkpoint and Recovery" in *Using Business Activity Monitoring Workbench*.
Process events in the order of arrival	Choose this option when events must be processed in the order received. Otherwise, if events may be processed out of order, turn on this attribute. **Note:** To join events in a view, the events must be processed in order. Leave this option off to join the events.
Disable event after this number of consecutive errors	Disables the event when a consecutive count of errors occur. For example, if set to 5, disables the event after 5 consecutive errors. However, if 4 errors occur, and then no errors occur followed by 2 errors, the event remains enabled. The default is *off*: Do not disable.

Attribute	Description
Column Information	The Column Information fields define the columns in the event table and are the same name as the fields in the XML message, as described in the WSDL for the event.
Clear State Interval	This tab contains several options for clearing persisted event data that is propagated from the event in the views, objects, and dashboards that depend on it. The three options are: • Do Not Clear State — This is the default. Data persists. • Clear State on a Schedule — Select to clear the state on a schedule. Selecting this option activates the scheduling feature positioned to the right. • Always Clear State (Every Event) — The state is refreshed each time the event is updated.

Creating a Web Service Event

Before creating a Web service event table, you need to create permission for tables (see "Creating Permission" on page 258).

➤ **To publish event from a Web service:**

1. Create a Web service event in the BAM Workbench.
2. (*Optional*) Create a Business Activity Monitoring user account for the Web service to use when publishing the event. The account must have at least Read-Write on the event table to publish to the event stream.
3. Retrieve the event WSDL definition using HTTP.

   ```
   http://<host:port>/bam/wsdl/eventstream.wsdl
   ```

   The eventstream.wsdl file describes all defined Business Activity Monitoring Web service events streams. See the documentation in the WSDL for descriptions of the XML elements and attributes.
4. Create the Web service publisher and define the XML event data to conform to the WSDL.

   This example of XML carries data to the *OrderWSEvent* event stream:

   - The account used to access the event stream (*WSInputAccount*)

- There are four columns of data defined in the <OrderWSEventData> element

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <OrderWSEvent_input xmlns="http://www.celequest.com">
         <cqesi>
            <eventname>OrderWSEvent</eventname>
            <username>WSInputAccount</username>
            <password>wspwd</password>
         </cqesi>
         <OrderWSEventData>
            <ProdName>Plywood</ProdName>
            <OrderQuantity>150</OrderQuantity>
            <OrderTotal>987.34</OrderTotal>
            <IsBackordered>true</IsBackordered>
         </OrderWSEventData>
      </OrderWSEvent_input>
   </soapenv:Body>
</soapenv:Envelope>
```

5. Publish events to the URL identified in the eventstream.wsdl file, similar to the following:

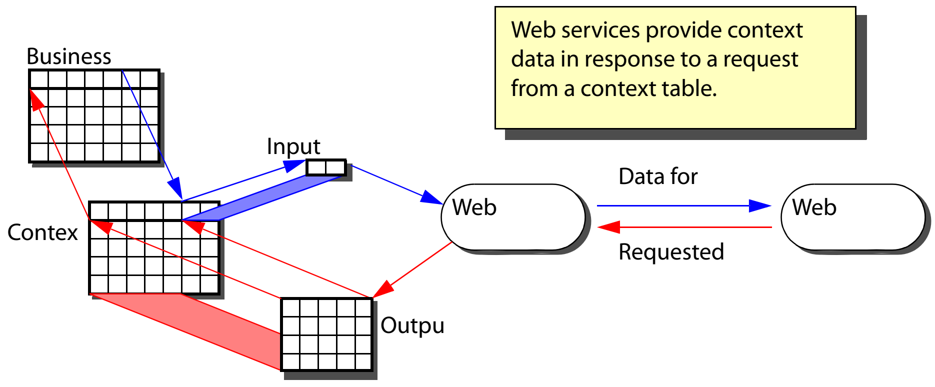
   `http://<host:port>/bam/webservice/eventstream.wsdl`

   If you later change the machine that hosts the Business Activity Monitoring servers, be sure to re-query the WSDL file to determine the correct URL.

This completes the steps for creating and using a Web service event.

Web Service Context

Business views request rows from a context table that match one or more input values, such as a list of suppliers that supply an item, where the item ID is the input. That input is then passed to the Web service application though the agent as XML. The application then returns one or more rows of data as XML, which are then mapped into the context table. The table then passes the requested data to the requesting business view.


Web services provide context data in response to a request from a context table.
Business
Input
Contex
Web
Data for
Web
Requested
Outpu

Business Activity Monitoring Web service agents are synchronous: they retrieve context data as the result of a specific request. When requesting data, the agent uses Simple Object Access Protocol (SOAP binding) to communicate with the application through an HTTP connection. The application then returns one or more rows of data in XML, following the Web service definition language (WSDL) format, Doc-type format. (Note that WSDL RPC-style is *not* supported.)

Note: For details about context agents, see "Web Service Agents" on page 370

Before creating a Web service context table, you must:

- Create permission for tables (see "Creating Permission" on page 258)
- Have a Web service agent that will feed the table (see "Web Service Agents" on page 370 for details)
- Read Only access permission on the agent.

A Web services table has the following attributes:

Attribute	**Description**
Name	Context table name. This name must be unique among views, events, context, and consolidated events. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the table is enabled (monitoring for events), or disabled (not monitoring for events).
Web Service Agent	An existing Web service agent that connects to a Web service application. Create an agent with the BAM Workbench Administration tab. See "Web Service Agents" on page 370 for details. This value cannot be changed.
Method	Method of the Web service to use. When the service provides multiple methods, you need to choose which one to use. This value cannot be changed in this release.

Attribute	Description
XPath Root	Identifies the XPath root of the repeating elements in the output, typically /Envelope/Body. This path is prepended to all paths in the Output Field Name list.
Disable context after errors	Count of consecutive errors to receive before the system disables this context. Once disabled, a context must be re-enabled manually.
Outputs	Columns that receive the information from the Web service. See "Output Columns" on page 368 for details.
Inputs	Columns that contain the data which identify what to look up in the query. See "Input Columns" on page 368 for details.
Caching	See "Caching Context Queries" on page 48 for details about this feature.

Creating a Web Service Context Table

Use the following procedure to create a Web service context table.

➤ **To create a Web service context table:**

1. In the BAM Workbench, Workbench tab, create a New context and choose Web service as the table type.

2. Select an existing **Web service** agent.

3. Select the service method to use. Each Web service provides one or more methods for accessing the data it provides according its WSDL file definition (the URL location of which you specified when creating the Web service agent). Choose the method that performs the query your context needs.

4. Define columns that receive information from the Web service — the Output columns. By default, the editor defines one column for each element returned by the method. See "Output Columns" on page 368 for details about defining these columns.

5. Define the columns that contain the data that identify what to look up in the query — the Input columns. By default, the editor defines one column for each element returned by the method. See "Input Columns" on page 368 for details.

6. Specify how many results to cache, if any. See "Caching Context Queries" on page 48 for details about this feature.

Save the Web services table as enabled and it will immediately be ready to receive context.

Output Columns

The Output columns receive the information from the Web service and define the table to receive the data. The editor automatically defines one column for each element returned by the method. Further, each column has the following attributes:

Attribute	Description
Column Name	Name of the table column that contains the result returned by the Web service. By default, the name is the same as the element in the Output Field Name. You may assign any valid name.
Output Field Name	(*Cannot be changed*.) Identifies the element in the XML returned by the service. Note that you can view the entire path to the element by opening the Hide/Show dialog.
XSD Data Type	(*Cannot be changed*.) Identifies the data type of the element in the XML. Note that only the basic data types are supported: numbers, strings, dates, and boolean. Complex types like ANY and ARRAY and mime types are *not* supported.
Business Activity Monitoring Data Type	Data type of the column in the table. Choose a data type appropriate to the data returned.
Formatting	Formats the String, Date-Time, or DECIMAL value returned. This option is not available for other data types. See "Data Types" on page 55 for details.

To exclude columns from the result, open the Hide/Show dialog and deselect the fields to exclude.

Input Columns

The Input columns pass information to the Web service to identify the information to return (the Outputs). The editor automatically defines one column for each element identified by the method. Each column has the following attributes:

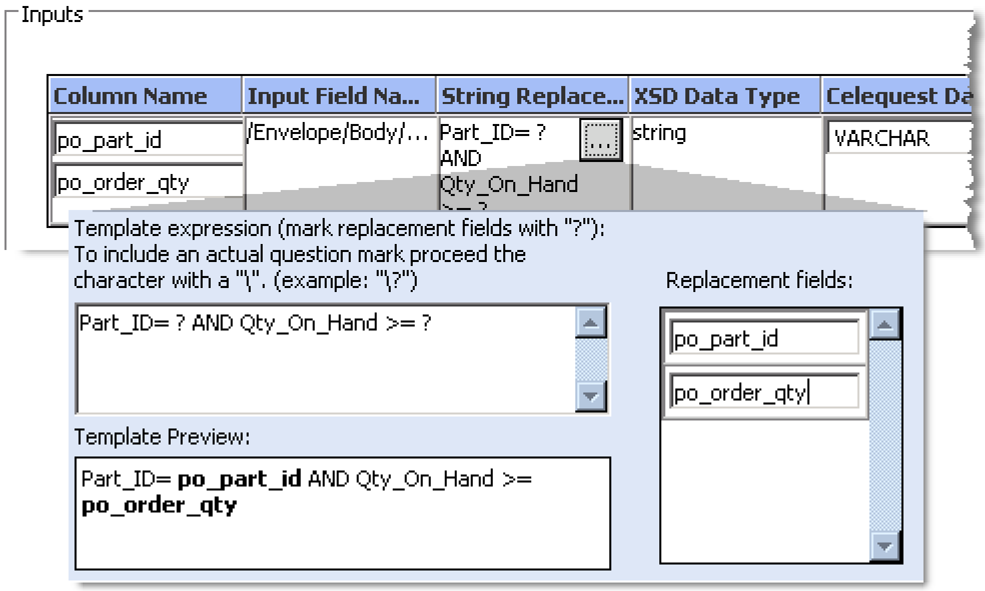
Attribute	Description
Column Name	Name of the column that contains the information passed to the Web service query. For example, it might contain an ID that identifies a product to look up. This column is populated by the business view that requires the context information. By default, the name is the same as the element in the Input Field Name. You may assign any valid name.
Input Field Name	(*Cannot be changed*.) Identifies the element in the XML passed to the service. Note that you can view the entire path to the element by opening the Hide/Show dialog.
String Replacement Text	A string to pass to the service which contains values inserted by the business view requesting the information. See "String Replacement Templates" on page 369for details.

Attribute	Description
XSD Data Type	(*Cannot be changed.*) Identifies the data type of the element in the XML. Note that only the basic data types are supported: numbers, strings, dates, and boolean. Complex types like ANY and ARRAY, and mime types are *not* supported.
Business Activity Monitoring Data Type	Data type of the Column Name attribute. Choose a data type appropriate to the data to pass.
Formatting	Formats the String, Date-Time, or DECIMAL value returned. This option is not available for other data types. See "Data Types" on page 55 for details.

To exclude unnecessary columns from the query, open the **Hide/Show** dialog and deselect the fields to exclude.

String Replacement Templates

Use a String Replacement Template when the Web service requires a string with embedded lookup data. For example, some services require an expression that is the query to issue to the data source. The illustration below contains returns context, where Part_ID and Qty_On_Hand values are provided by the business view requiring the information. In the illustration, each Column Name (Alias) is a column in the context table. When the Web service is queried, the values in those columns are inserted in the string passed to the service.


Inputs
Column Name
Input Field Na...
String Replace...
XSD Data Type
Celequest Da
po_part_id
/Envelope/Body/...
Part_ID= ? AND Qty_On_Hand >= ?
string
VARCHAR
po_order_qty
Template expression (mark replacement fields with "?"):
To include an actual question mark proceed the character with a "\". (example: "\?")
Part_ID= ? AND Qty_On_Hand >= ?
Template Preview:
Part_ID= **po_part_id** AND Qty_On_Hand >= **po_order_qty**
Replacement fields:
po_part_id
po_order_qty

➤ **To use string replacement templates:**

1. Click on the (**...**) to open the template editor.
2. Type the template expression and enter a question mark (?) for each piece of information to retrieve from a column. Each question mark corresponds to one replacement field. The fields are the column names in the context table and appear in the order that the question marks appear in the expression.

Save the template to update the Inputs column display.

Web Service Agents

A Web service agent communicates with an application running on a Web application server for the purpose of retrieving context data. The agent connects to the application through an HTTP connection, using Simple Object Access Protocol (SOAP).

Note: Web service agents are synchronous, they retrieve context data as the result of a specific request as summarized in the following table.

Event Push	Event Pull	Context Pull
Yes	No	Yes

Before creating a Web services agent, you need:

- To create permission for agents (see "Creating Permission" on page 258 for details).
- To know the HTTP location of the WSDL file that defines the service to use. Note that the service must publish its data in SOAP binding; RPC binding is not supported.

Web Service Agent Attributes

A Web service agent has the following attributes:

Attribute	Description
Name	Identifies the agent. This name must be unique among agents. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters.
Status	Whether or not the agent is enabled (monitoring for events) or disabled (not monitoring for events).
URL	HTTP location of the Web service definition language file (WSDL) that describes the service, the data it provides, and how to exchange data with the service. Note that returned data must be a SOAP doc-style message; RPC binding is not supported.
User name	(optional) User name to use when connecting to the service. This parameter is passed to the server when the server requires a user name.
Password	(optional) User password to use when connecting to the service. This parameter is passed to the server when the server requires a password.

Creating a Web Service Agent

Use the following procedure to create a Web service agent.

➤ **To create a Web service agent:**

1. In the BAM Workbench, Administration Console, click New Agent…
2. Choose Web Service as the source type.
3. Fill in the fields that define the agent's attributes.

Save the agent as enabled and it will immediately be ready to retrieve data.

Web Service External Processes

External Web service processes are methods that receive XML documents that describe the alert message or dashboard object that a BAM Dashboard user is viewing, and which was sent to the service by the user. The XML document describes all of the data in the item that the user was viewing.

➤ **To publish an item to an external Web service:**

1. Define the external process to receive the message in the BAM Workbench. See Creating an External Process.
2. Send the item from the BAM Dashboard. When viewing an alert message or dashboard object, select **Take Action > Initiate Process** and select the process.
3. The external service receives the item as an XML document and processes it. See "Implementing the External Service" on page 372 for details about the document.

External Process Attributes

Each external process has the following attributes:

Attribute	Description
Name	External process name to appear in the Initiate Process dialog in the BAM Dashboard. This name must be unique among external processes. See "Object Namespace" on page 248 for details.
Description	Optional description that may contain any text characters. This description appears in the Initiate Process dialog in the BAM Dashboard.
Status	Whether or not the process is enabled (sending XML documents) or disabled (not sending documents).
Web Service URL	HTTP location of the application providing the RPC (SOAP binding) service. Note that DOC style messages are not supported.
Method	Method of the Web service to use.

Attribute	Description
Username	(Optional) Account to use when connecting to the service.
Password	(Optional) Password for the account.

Creating an External Process

Define an external process in the Administration Console of the BAM Workbench.

Before creating a Web services external process, you need:

- An already defined, external Web service method to receive the published method. You will need to know the URL for connecting to the service, the name of the method that will receive the message, and any user or account name and password required by the service. See Implementing the External Service for additional details.
- Create permission for external processes (see "Creating Permission" on page 258 for details).

➤ To create an external process

1. In the BAM Workbench, open Administration Console > External Processes list, and click New Process.
2. Define the attributes for the process.
3. Save the process and it is immediately available to all users with access to it.

To use the process, in the BAM Dashboard, choose Take Action > Initiate Process and select the process.

> **Note:** To see and use a process from the BAM Dashboard, users will need Read permission for that process. You can assign permissions for the new object by clicking Permissions in the External Processes list, or an administrator can grant Read access to the class of external processes for the users. See "Accessing Permissions" on page 255 for details.

Implementing the External Service

To implement the external Web service,

- Define it to receive a SOAP binding message with the fields in the external action XSD, and
- Create a WSDL (definition file) following Business Activity Monitoring target and import requirements.

Message fields

The XML message fields are defined in the invokeExternalAction.xsd definition file. You can find this (and all XSD files) in the /api/metadata directory on the product CD-ROM file. For more information about XML and XSD files in Business Activity Monitoring, see "XML/XSD" on page 375.

> **Note:** You will also need common.xsd located in the same directory.

Every message contains at least these fields:

- description — Description of the external process defined in the BAM Workbench
- actionName — Name of the external process defined in the BAM Workbench.

- severity — Severity either of the original alert message, or as chosen by the user that initiated the message.

Other fields are included as necessary based on the object that the user was viewing in the BAM Dashboard when they initiated the action, such as the subject of an alert message, or the row set of the data in the view on which a chart was presenting. See the XSD file for details.

Web Service WSDL

When implementing the Web service, define it to receive a SOAP message and with the following attributes:

Attribute	**Value**	**Comment**
style	rpc	Do not use “document”.
target namespace (tns)	http://www.celequest.com	—
encoding (soap:body)	encoded	Do not use “literal”.
import namespace	http://www.celequest.com/2	Defines Business Activity Monitoring data types. Alternatively, you can define the types in the WSDL, but that is beyond the scope of this document.
import location	*Installation-specific*	Location of invokeExternalAction.xsd in your installation.

The following is an example WSDL that handles the invokeExternalAction message on a machine and port named *host:80*. Note that it imports the invokeExternalAction.xsd definition file.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:impl="http://www.celequest.com/2"
    xmlns:tns="http://www.celequest.com"
    targetNamespace="http://www.celequest.com"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

<import namespace="http://www.celequest.com/2"
     location="/celequest/api/metadata/invokeExternalAction.xsd"/>

<message name="invokeExternalActionRequest">
    <part name="request" element="impl:invokeExternalAction" />
</message>

<portType name="invokeExternalActionPortType">
    <operation name="invokeExternalActionOperation">
        <documentation>Receives a COGNOS LAVA external action.
        </documentation>
        <input message="tns:invokeExternalActionRequest"/>
    </operation>
</portType>

<binding name="invokeExternalActionBinding"
         type="tns:invokeExternalActionPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                  style="rpc" />
    <operation name="invokeExternalActionOperation">
        <soap:operation soapAction=""/>
        <input>
            <soap:body
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://www.celequest.com" use="encoded"/>
        </input>
    </operation>
</binding>

<service name="invokeExternalActionService">
    <port name="invokeExternalActionService"
          binding="tns:invokeExternalActionBinding">
        <soap:address
 location="http://host:80/axis/services/invokeExternalActionService"/>
    </port>
</service>
</definitions>
```

36 XML/XSD

All Business Activity Monitoring Objects and many system operations can be expressed in XML format and uploaded into the system. The XML must be properly formatted as defined by XML schema files (XSD). For a complete list of the objects defined and operations you can perform, see "Business Activity Monitoring XSD files" on page 383.

> **Note:** Details about XML and XSD are beyond the scope of this documentation. For information about XML and XSD, see www.xml.org.

➤ **In this Chapter:**

About XML and XSD Files in Business Activity Monitoring

All XML files in Business Activity Monitoring have schema files that define the structure of the XML. The XSD files and sample XML files, are provided on the product CD-ROM.

/api/metadata	`XSD files`
/samples/metadata	`Sample XML files`

In addition to the samples, another way to see properly defined XML files is to first create objects in the BAM Workbench or BAM Dashboard, then use the Administration Console to export the entire set. All exported objects are written as XML files to the export directory on the server. For more information about this procedure, see "Importing/Exporting Metadata" in *Using Business Activity Monitoring Workbench*.

Dependencies

Most of the objects in the system depend on other objects. When you define a new object, all of its dependences must be defined first. Do that by defining and uploading base objects in the order of dependency, or by defining them in batch as described in "Defining Multiple Objects with XML" on page 380.

White Space

When an XML element value contains multiple, contiguous white space characters that must be retained, direct the system to keep the spaces with xml:space="preserve". Otherwise, the XML specification says to remove extra spaces. For example, without the preservation directive, the following description would be trimmed of the trailing spaces and would have only one space between the words:

```
<description xml:space="preserve">A   note   </description>
```

Escape Characters

The characters "<" and "&" are illegal in XML. Some other characters are legal but can cause confusion when looking at them. For these characters, use these XML escape entities instead.

Sequence		Result
&lt;	<	Less than
&gt;	>	Greater than
&amp;	&	Ampersand
&apos;	'	Apostrophe
&quot;	"	Quotation mark

For example, when expressing a query that contains a less-than symbol, use &lt; instead, for example:

```
<query>SELECT c1, c2 FROM event1 WHERE c3&lt;=100 AND
c2=&apos;CQST&apos;</query>
```

Character Data

Instead of using escape characters, another way to express special characters is to use a CDATA tag. This tag tells the parser to ignore all special characters and treat them as literals. For example:

```
<query><![CDATA[SELECT c1, c2 FROM event1
WHERE c3<=100 AND c2='CQST']]></query>
```

Uploading XML Files

There are two ways to upload XML files into Business Activity Monitoring:

- From a Command Line with the cqupload.jar utility.
- From a Web Browser with the fileupload.jsp script.

Both methods require that the Business Activity Monitoring server to be running. Also, each method uses a Business Activity Monitoring user account to log in to the server and perform the action. In each case, the account must have create rights to the class of objects to create, or administration rights to the operations to perform. See "Users" on page 332 for information about user accounts.

From a Command Line

From a command line, use the cqupload.jar utility to upload files. You can find it on the product CD-ROM in the /BAM/CQUpload/ directory.

This utility has three options:

Option	Description
-s applicationURL	A URL that locates Business Activity Monitoring
-u userName	(Optional) User account to use. Omit this option to use the default system administrator account.
-p password	Password for the user account. Required if you include -u.

For example, to upload an XML using the default system administrator account:

```
java -jar <CD-ROM>/BAM/CQUpload/cqupload.jar
-s http://<applicationServer>/lava createUserSkyler.xml
```

To include a username or password:

```
java -jar cqupload.jar -u skyler -p roo -s ...
```

A successful operation occurs silently; however, if the operation fails, the utility returns an error message to the command window. Review the error message to identify the problem. For example, this message indicates an error in the XML:

```
Error uploading file: createUserSkyler.xml
---------------------------------------------------------------------
java.io.IOException: Error parsing an XML document.  Ensure that the
XML conforms exactly to the XML schema definition.  The XML that
cannot be parsed is:
```

Further down the message you can find the actual cause: an invalid element:

```
Caused by:
javax.xml.bind.UnmarshalException: Unexpected element
{http://www.celequest.com/2}:nome
```

From a Web Browser

The fileupload.jsp script presents a form where you identify the XML file to upload. It includes a file picker where you to identify the XML file to load and displays a message with the results of the upload.

➤ **To upload from a Web browser:**

1. Run the fileupload.jsp script. Use an address similar to the following URL. Use the localhost only if you are running the browser on the same host as Business Activity Monitoring servers; otherwise, use the same location that you use to run the BAM Workbench.

   ```
   http://localhost/bam/jsp/fileupload.jsp
   ```

2. Identify the user name and password of the Business Activity Monitoring account to use.
3. Identify the XML file to upload.
4. Choose **Upload** to perform the action.

The results page displays the name of the XML file uploaded and its result. If the Status is Failed, review the exception to see what went wrong and correct the problem. For example, this message reveals that the operation failed because the user object *Rolf* did not exist.

```
Exception Encountered
com.celequest.exception.VCException: Cannot alter the [User]
named [Rolf] because that object does not exist.
```

Defining an Object with XML

This section describes how to define and upload a single XML definition. To upload multiple XML files — especially objects with dependencies — follow the instructions in the next section, "Defining Multiple Objects with XML" on page 380.

➤ **To define an object with XML:**

1. Create XML definition. Use the associated XSD file to determine the valid elements of the XML file. See Example: Create User below for an example of a complete XML file.

2. Ensure that Business Activity Monitoring is running.

3. Upload the XML file.

   Use either of the methods described in "Uploading XML Files" on page 377. To upload the file from the command line with the cqupload.jar utility:

   ```
   java -jar <CD-ROM>/BAM/CQUpload/cqupload.jar
   -s http://<applicationServer>/lava createUserSkyler.xml
   ```

This completes the steps for creating a new object.

Example: Create User

```
<?xml version="1.0" encoding="UTF-8"?>
<createUser
   xsi:schemaLocation="http://www.celequest.com/2
                    ../../api/metadata/createUser.xsd"
   xmlns="http://www.celequest.com/2 "
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
   <name>Skyler</name>
   <description xml:space="preserve">A power user</description>
   <Password>roo</Password>
</createUser>
```

Defining Multiple Objects with XML

When defining multiple objects — especially objects with dependencies — use commandBatch.xsd: the "batch mode" XML definition object. When you use the batch mode, include all of the XML in a single file, then upload that file. All of the operations must be valid or none of them are accepted. To define and upload a single XML object, follow the instructions in "Defining an Object with XML" on page 379.

➤ **To define multiple objects with XML:**

1. Create an XML batch file. Use commandBatch.xsd as the definition. Within the file, nest each definition within a <command> element and place them all in a single <commands> element in the order that the objects must be defined.

   Note: See "Example: Batch command" on page 381 for a listing that defines multiple, dependant objects.

2. Ensure that Business Activity Monitoring is running.

3. Upload the batch XML file. Use the cqupload.jar utility to upload the XML file the application server running Business Activity Monitoring. For example, to upload the example batch file:

   ```
   java -jar <CD-ROM>/BAM/CQUpload/cqupload.jar
   -s http://<applicationServer>/lava commandBatchSkyler.xml
   ```

This completes the steps for defining multiple objects.

Example: Batch command

This batch command defines a user account, two delivery profiles for the user, a user preference, and assigns one permission to the user.

```
<?xml version="1.0" encoding="UTF-8"?>
<commandBatch
   xsi:schemaLocation="http://www.celequest.com/2
                     ../../api/metadata/commandBatch.xsd"
    xmlns="http://www.celequest.com/2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
  <commands>
    <command>
      <createUser>
        <name>Skyler</name>
        <description xml:space="preserve">A power user</description>
        <Password>roo</Password>
      </createUser>
    </command>
    <command>
      <createUserProfile>
        <name xml:space="preserve">Work e-mail</name>
        <UserName>Skyler</UserName>
        <isDefault>true</isDefault>
        <EmailProfile><typeName/>
          <emailAddress>skyler@celequest.com</emailAddress>
        </EmailProfile>
      </createUserProfile>
    </command>
    <command>
      <createUserProfile>
        <name xml:space="preserve">Second profile</name>
        <UserName>Skyler</UserName>
        <isDefault>false</isDefault>
        <EmailProfile><typeName/>
          <emailAddress>skyler@viewceler.com</emailAddress>
        </EmailProfile>
      </createUserProfile>
    </command>
    <command>
      <setUserPreferences>
        <userName>Skyler</userName>
        <userPreference>
          <operation>set</operation>
          <name>polling_interval</name>
          <value>5</value>
        </userPreference>
      </setUserPreferences>
    </command>
    <command>
      <setPrivilege>
        <operation>GRANT</operation>
        <privilegeTuple>
          <objectType>VIEW</objectType>
```

```
            <privilege>UPDATE</privilege>
            <accessorType>USER</accessorType>
            <accessorName>Skyler</accessorName>
          </privilegeTuple>
          <withGrant>false</withGrant>
        </setPrivilege>
      </command>
    </commands>
  </commandBatch>
```

Altering an Existing Object with XML

To alter an existing object's definition, use the same XML and schema as when creating the object, but include an <alterInformation> element to identify the alter operation. (The <alterInformation> element is defined in common.xsd.) For example, this definition renames a view from *OldName* to *NewName*; note that it uses the createView.xsd schema:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<createView xsi:schemaLocation="http://www.celequest.com/2
    /api/metadata/createView.xsd"
    xmlns="http://www.celequest.com/2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <alterInformation>
        <previousName>OldName</previousName>
    </alterInformation>
    <name>newName</name>
    <description>My test view</description>
    <query>SELECT c1, c2 FROM event1 WHERE c3&lt;=100</query>
</createView>
```

No matter what change you are implementing, you must use <previousName>. If you are not changing the object's name, use the same name for both <previousName> and <name> elements, like this:

```
    ...
    <alterInformation>
        <previousName>OldName</previousName>
    </alterInformation>
    <name>OldName</name>
    <description>New description</description>
    <query>SELECT c1, c2 FROM event1 WHERE c3&lt;555</query>
    ...
```

By default, an alter operation fails if the existing object does not exist. However, you can force the object to be created regardless of the existence of the existing object by including a <createIfNotFound> element, like this:

```
    ...
    <alterInformation>
        <previousName>OldName</previousName>
        <createIfNotFound>true</createIfNotFound>
    </alterInformation>
    ...
```

Dependencies

When you alter an existing object all other objects that depend on the altered object are evaluated and made “invalid” if their definition is broken as a result of the change. You will have to alter the invalid objects and correct their definitions before they can be re-enabled.

Issuing Commands with XML

Many system operations can be performed with XML commands. Here are some of the common operations:

Operation	**Schema**
Enable an object	enableObject.xsd
Disable an object	disableObject.xsd
Drop (delete) an object	dropObject.xsd
Set a system property	setProperty.xsd
Import or export the system metadata	performImportExport.xsd
Perform a “checkpoint”	systemCommand.xsd

The following is an example operation.

Example: Enabling an Object and Its Dependencies

To enable an object and all of its dependencies, use the enableObject.xsd schema. You must identify the name of the object and its object type. (The valid <type> values are defined in common.xsd by the <VCEnableObjectType> element.)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<enableObject xsi:schemaLocation="http://www.celequest.com/2
    /api/metadata/enableObject.xsd"
    xmlns="http://www.celequest.com/2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <name>MyView</name>
    <type>VIEW</type>
    <cascade>true</cascade>
</enableObject>
```

This example enables all dependent objects because <cascade> is set to true. Omit this element, or set it to false to enable just the named object.

Business Activity Monitoring XSD files

These are Business Activity Monitoring XML schema files, arranged by category:

- “Users, Profiles, and Roles” on page 385” (below)
- “Dashboard” on page 385

- "Rules, Alerts, and Reportlets" on page 385
- "Events, context, cubes, and views" on page 387
- "Scenarios and business activities" on page 387
- "Agents" on page 388
- "System administration" on page 388
- "Object management" on page 388
- "Miscellaneous files" on page 389

The files are located are located on the product CD-ROM in this directory: /api/metadata.

Users, Profiles, and Roles

Schema	**Description**
addMembersToRole.xsd	Adds one or more existing users to an existing role.
addMemberToRole.xsd	Adds an existing user to an existing role.
createRole.xsd	Creates a user role object.
createSecurityFilter.xsd	Creates an access filter
createUser.xsd	Creates a user object.
createUserProfile.xsd	Creates a user delivery profile object.
setPrivilege.xsd	Sets a user's or role's permission on an object or class of objects.
setUserPreferences.xsd	Set a user's preferences.

Dashboard

Schema	**Description**
createBookmarks.xsd	Creates a dashboard bookmark
createDashboard.xsd	Creates a dashboard
createPlan.xsd	Creates dashboard references and plans
createPortlet.xsd	Creates a dashboard object
createTask.xsd	Creates a dashboard task
createTaskMessage.xsd	Creates a dashboard task message

Rules, Alerts, and Reportlets

Schema	**Description**
alertCommon.xsd	Common alert XSD definitions.
alertMessage.xsd	Alert message to be delivered to a Web Service.
alterAlertState.xsd	Alters the state of an existing alert.
alterRule.xsd	Alters an existing rule object.
createAlert.xsd	Creates an alert object.
createExcelTemplate.xsd	Creates a Excel Template object that describes the template that a reportlet might use.

Rules, Alerts, and Reportlets

Schema	Description
createReportlet.xsd	Creates a reportlet object.
createRule.xsd	Creates a rule object.
createRuleBundle.xsd	Specifies the values associated with parameters in a rule template, and generates the rules, alerts and reportlets based on the template definition.
createRuleTemplate.xsd	Creates a template of parameterized definitions of a set of rules, the alert used by the rules, and the reportlets associated with the alert.
ruleCommon.xsd	Contains common rule XSD definitions.

Events, context, cubes, and views

Schema	Description
createConsolidatedEventView.xsd	Creates a consolidated event view based on an existing event stream and one or more additional event streams or views.
createContext.xsd	Creates a context object based on an existing source.
createCube.xsd	Creates a cube object.
createDimension.xsd	Creates a cube dimension
createEventStream.xsd	Creates an event (stream) object based on an existing source.
createView.xsd	Creates a view object based on an existing source table (event stream) and other, optional (joined) tables (event or context).
createViewPersistence.xsd	Creates a view persistence definition.
ffsourceType.xsd	A flat-file event object.
jdbcSource.xsd	A JDBC context object.
messageSource.xsd	A message object passed from an agent to an event or context, used by JMS, TIBCO RV, log4j, and HTTP.
queryCube.xsd	Issues a query on a cube against measures in dimensions.
queryInformation.xsd	Contains the elements of a query (SELECT statement).
sourceDefinition.xsd	Source types (agent types) supported by the system.
sourceDefinition.xsd	Contains common source definitions for event and context object XSD definitions.
tableDefinition.xsd	Supports event source definition.
webServiceSource.xsd	A Web Service event object.

Scenarios and business activities

Schema	Description
createBusinessActivity.xsd	Creates a business activity object.
createScenario.xsd	Creates a scenario object in an existing business view and linked to an existing view.

Agents

Schema	Description
FFConnection.xsd	A flat-file agent.
jdbcConnection.xsd	A JDBC agent.
JMSTopicConnection.xsd	A JMS agent.
log4jConnection.xsd	A log4j messaging agent (used internally for logging).
procSource.xsd	Stored procedure definitions.
RVConnection.xsd	A TIBCO RV agent.
SOAPConnection.xsd	A Web Service connection.
xmlBuffer.xsd	An XML buffer, holds part of a message.

System administration

Schema	Description
propertyTypeDefinition.xsd	Describes a system property.
setLogLevel.xsd	Sets the logging level is a system logger (logging module).
setProperty.xsd	Command to set a system property.
systemCommand.xsd	Performs a checkpoint.

Object management

Schema	Description
createKeyRelationship.xsd	Creates a relationship between two objects.
disableObject.xsd	Disables an object.
dropObject.xsd	Deletes an object
enableObject.xsd	Enables an object.
setObjectRelation.xsd	Relates two objects.

Miscellaneous files

Schema	**Description**
commandBatch.xsd	Defines multiple objects to be loaded (defined) in batch. All definitions must be valid or no objects are defined.
common.xsd	Contains common XSD definitions used by most XSD schemas.
createJar.xsd	Creates a JAR object.
createUDF.xsd	Creates a UDF object.
invokeExternalAction.xsd	Describes an external action message sent to a Web service. For more information about this file, see "Web Service External Processes" on page 371.
jarManifest.xsd	Defines the manifest in a JAR.
performImportExport.xsd	Command to the system to perform an import or export
schedule.xsd	Contains common schedule and schedule interval definitions used by XSD definitions.

Glossary

business data modeling

A technique for describing the events, context, views, and rules that depict how your business functions.

cascade

A operation that propagates the exact same operation to all dependant objects.

consolidated event

An event table that captures events from different, but similar event sources and combines them into a single event stream. See Working with Consolidated Events for details.

current view

A snapshot of the data currently in a view. For a stateful view, the snapshot shows all rows in the view. For a stateless view, it shows all the rows corresponding to the last event, and might be an empty set. See "View Initialization" on page 357 for details.

delivery profile

Specifies where and how to deliver alerts and data feeds to the user. See "Delivery Profiles Tab" on page 334 for details.

enabled

An object that is accepting new data and is processing them. All objects are created *enabled*. See "Object Status" on page 244 for details.

disabled

An object that is not accepting new data. Disabling an object does not affect the definition or existence of that object; rather, it just keeps new data from flowing into the object and to all objects that rely on the target object. See "Object Status" on page 244 for details.

inner join

A join where the rows in the resulting view are the rows from the first table or view that meet the specified criteria, combined with the corresponding rows from the second view that meet the specified criteria. Inner joins are sometimes called *equi-joins*.

invalid

An object that has a reference to another object which cannot be satisfied. A reference can be invalid because an object does not exist or because some attribute of an object does not match the requirements of the dependent (such as a data type mismatch), not because the dependent is disabled. Note that all objects that depend on an invalid object are also invalid. See "Object Status" on page 244 for details.

invalid and disabled

An object that is both disabled and invalid; it cannot receive data and it has no state. See "Object Status" on page 244 for details.

metrics

Measurements taken over time that monitor, assess, and communicate vital information about the results of a program or activity.

moving set function

A function that performs calculations on a set of the *latest rows* in a view. The set of rows to include is determined only when a new event arrives. For more information, see "Moving Set" on page 110.

outer join

A join where the rows in the result table are the rows that would have resulted from an inner join and the rows from the first table (LEFT OUTER JOIN) or the second table (RIGHT OUTER JOIN) that had no matches in the other table.

query window

Specifies a set of rows that are used in calculations with respect to the current row (event) under examination. The calculation may be for computing a moving set function, a join, or expiring rows from a view. See "Query Windows" on page 272 for details.

rank function

A function that computes the scalar result for each value in a set, with respect to the entire set. A rank function may only be used in the selection list of a SELECT statement. For more information, see "Rank" on page 110.

recent view

A snapshot of the last non-empty current view. A recent view is what appears in the BAM Workbench when editing an object that displays view results. See "View Initialization" on page 357 for details.

scenarios

Test business data models for expected or possible outcomes, and to identify exceptional business conditions. See "Scenarios" on page 318 for detailed information.

scalar expression

An expression without a set function.

scalar function

A function operates within the bounds of a single event and provides a single result for each row, such as the absolute value of a number or concatenation of two strings. These functions may appear in any C-SQL expression. For more information, see "Scalar" on page 109.

set function

A function performs calculations on a column in a set of rows in a view, such as the average value of the cost of some similar product orders. A set function may only be used in the selection list of a SELECT statement. For more information, see "Set" on page 110.

snapshot view

A view that is a replica of a view at the time the query executed; constructed with SELECT *. Note that the view might not reflect those events which have arrived in the system but which have not yet been processed at the time of the query.

stateful view

A view that contains the results of aggregations derived from past events in a single row. A view is stateful if it contains an set function or moving set function in the SELECT clause, or contains a GROUP BY Clause (in which case there is one row for each group), or is derived from a stateful view. See "Stateless and Stateful Views" on page 356 for details.

stateless view

A view that is not a stateful view. See "Stateless and Stateful Views" on page 356 for details.

terminal set function

a set function that has only scalar arguments.

terminal rank function

A rank function that has only scalar arguments.

tumbling set function

A function that performs calculations on a windowed set of the rows in a view. The set of rows to include is determined when a new event arrives, and the set empties when full. For more information, see "Tumbling Windows" on page 286.

user-defined functions

(UDFs) provide a mechanism for extending C-SQL by defining and your own functions for use in formulas, including queries, field expressions, and rules. See "User-Defined Functions" on page 346 for detailed information.

virtual table

A table or view where the rows are derived as they are required in memory.

window

See query window.

Index