Oracle® Data Relationship Management Suite
Administrator's Guide

Release 11.2.x

F13691-07

November 2025

Oracle Data Relationship Management Suite Administrator's Guide, Release 11.2.x

F13691-07

Primary Author: EPM Information Development Team

Contents

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Documentation Feedback

To provide feedback on this documentation, click the feedback button at the bottom of the page in any Oracle Help Center topic. You can also send email to epmdoc_ww@oracle.com.

1 Revision History

The following topics have been updated in this release of the guide:

Topic	Change
Data Types	Added note to both Float and Integer that 0 will export if no default value has been defined. Added information that data types Date, Date/Time, and Time are formatted in the invariant culture.
Function Definitions	Updated the Equals function to say that the comparison is case sensitive.
Creating Dynamic Scripts	Added note that when calculating parent names, any use of special characters must follow the standard JavaScript rules for escaping special characters.
Migrating Data Relationship Management Metadata	Added a note to indicate that the connection string, user ID, and password for external connections do not migrate with migration loads and extracts. Added a new section called "Migrating Core Property Configurations and Settings".
Data Relationship Management Objects	Updated description for the NodeNamePendingInRequest method for RequestItemObject.
Calculated Name and Parent Properties	Added note to clarify behavior when name or parent is manually overridden.
Creating Workflow Models	Added note to Recalculate Task Properties bullet in step 6 to clarify behavior when name or parent is manually overridden.
Managing Dynamic Scripts	Updated description for NodeExists(abbrev) in HierarchyObject Methods table Added 2 new properties to RequestItemDetailObject: • CalcValue • HasCalcValue
Notifications	Various updates to clarify and update notification behavior.
Supported JavaScript Data Types	Added note to clarify using an Array.
Creating a Script Validation for Move	Added new topic Creating a Script Validation for Move
System Preferences	Added a note to description for the FindByProperties system preference. Updated descriptions for SharedNodeDelimiter and SharedNodeSequenceSeparator system preferences.

Topic	Change
Function Definitions	Clarified the local use of several functions.
Troubleshooting and Tips	Added a new section "Troubleshooting and Tips" to the Getting Started chapter. Added workaround information about pasting into fields. Added application performance information.
Validation Classes	Added recommendation that UniqueProp validation use indexed properties.

2
About Data Relationship Management Suite

Oracle Data Relationship Management Suite consists of:

- Oracle Data Relationship Management
- Oracle Data Relationship Management Read Only Access
- Oracle Data Relationship Steward
- Oracle Data Relationship Governance
- Oracle Data Relationship Management Analytics
- Oracle Data Relationship Management for Oracle Hyperion Enterprise Planning Suite
- Oracle Data Relationship Management for Oracle Hyperion Financial Close Suite

3
Getting Started

Related Topics

- Administering Data Relationship Management Applications
- Accessing Data Relationship Management
- Troubleshooting and Tips

Administering Data Relationship Management Applications

Oracle Data Relationship Management uses applications to manage data and service user requests for accessing and changing data. A single Data Relationship Management installation can support one or more applications. Each application uses its own system metadata and security configuration to manage and access data. The same application can support multiple data sets and multiple users with various levels of access to common and restricted sets of data. However, all system metadata within an application is shared and administered by the same users. Any changes to system metadata take effect immediately and all users and data may be affected. If different user groups need to be isolated from any metadata changes being made by another group, it is recommended that each group use a separate application.

Applications are created in the Configuration Console which is accessible from the Data Relationship Management primary application server. For more information on creating a new application, see "Creating an Application" in the *Oracle Data Relationship Management Installation Guide*.

A new Data Relationship Management application includes core metadata objects such as property definitions and categories and a default administrative user. This initial configuration enables the default user to perform four tasks to build, populate, and provision the application:

- Create versions and hierarchies
- Define user metadata objects such as queries, compares, imports, blenders, and exports
- Set up and configure system metadata objects including domains, property definitions, validations, and node types
- Add users and configure security to access product features, objects, and data

This guide covers the administration tasks related to system metadata and user security for Data Relationship Management applications. See the *Oracle Data Relationship Management User's Guide* for information on managing versions, hierarchies, and user metadata objects.

Accessing Data Relationship Management

To start the Oracle Data Relationship Management client:

1. Select **Start**, then **Programs**, then **Oracle EPM System**, then **Data Relationship Management**, then **Web Client** .
2. Enter your user name and password.

   User names and passwords are case-sensitive.

3. Select an application and click **Log On**.

For more information, see Changing Passwords.

Changing Passwords

To change a password:

1. From the Oracle Data Relationship Management Home page, select **Preferences**.
2. Click **Change My Password**.
3. Type the current password.
4. Type the new password.

> **Note**
>
> When a user is authenticated natively and the PasswordPolicyEnabled system preference is set to True, a password must contain three of the following elements:

- Uppercase letters
- Lowercase letters
- Numbers
- Special characters

> **Note**
>
> Otherwise, the password is not restricted unless by an external directory when the user is authenticated via Oracle Hyperion Shared Services.

5. Type the new password again.
6. Click **OK**.

Troubleshooting and Tips

Pasting Into Entry Fields

In some cases, content cannot be pasted from the clipboard by using right click and then **Paste**. To workaround this issue, use Ctrl-V or click **Edit** and then select **Paste** to paste content from the clipboard.

Application Performance

In order to maintain application performance, a standard programming practice has been employed to leverage a feature known as String Interning which provides more rapid access to string data. String Interning is where an immutable copy of each string is stored once and is maintained for subsequent access while the application is running. Therefore, as data is accessed and content is managed, the apparent memory footprint of the engine will grow incrementally until the application is restarted.

4

Managing Users

Related Topics

User Permissions

Oracle Data Relationship Management uses three levels of permissions to control user access to product features and data. Some higher-level permissions also include lower-level permissions. If a user is granted higher-level permission, then all lower-level permissions are also granted. For example, if a user is granted a Level 1 permission, they are also granted all Level 2 and 3 permissions below it.

Version Permissions

Table 4-1 Version Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage Versions—User has access to Version and Hierarchy menu options	**Browse Versions**—Users have access to any version that they are granted rights to in Node Access Groups	NA
	Create Versions—Users can manage (update/delete) any version of which they are the owner. User has access to Version menu options. **Note:** The user who creates a version is the owner until a user with Manage Versions permission changes the owner.	NA
	Manage Hierarchies—Users have access to Hierarchy menu options.	**Browse Hierarchies**—Users have access to any hierarchy that they are granted rights to in Node Access Groups. Users have access to Node menu options if they have Edit node access or greater.

Table 4-1 (Cont.) Version Permissions

Permission Level 1	Permission Level 2	Permission Level 3
		Create Hierarchies—Users can manage (update/delete) any hierarchy of which they are the owner. Users have access to Hierarchy menu options. Users can disable node types for any hierarchy of which they are the owner. **Note:** The user who creates a hierarchy is the owner until a user with Manage Hierarchies permission changes the owner.

Request Permissions

Table 4-2 Request Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage Requests—Users can delete any request in the system that has not already been committed.	**Create Requests**—Users can query any request in the system and can manage (update/delete) any request of which they are the owner.	NA
Workflow Participant—Users can participate in requests using governance workflow models.	NA	NA

Query Permissions

Table 4-3 Query Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage System Queries—Users have access to system queries and to Query menu options. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	**Manage User Queries**—Users have access to view and run User and Standard queries. Users do not have access to Query menu options for Standard Queries. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	**Run Query**—Users can view and run any Standard query. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security. Users have access to Node menu options if they have Edit node access or greater.
	Manage Standard Queries—Users have access to Query menu options for Standard queries. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	NA

Compare Permissions

Table 4-4 Compare Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage System Compares—Users have access to system compares and Compare menu options. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	**Manage User Compares**—Users have access to view and run User and Standard compares. Users do not have access to Compare menu options for Standard Compares. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	**Run Compare**—Users can view and run any Standard compare. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security. Users have access to Node menu options if they have Edit node access or greater.
	Manage Standard Compares—Users have access to Compare menu options for Standard compares. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	NA

Import Permissions

Table 4-5 Import Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage System Imports—Users have access to system imports and Import menu options. Users have restricted access to Property selector based on Property Category security.	**Manage User Imports**—Users have access to view and run User and Standard imports. Users do not have access to Import menu options for Standard Imports. Users have restricted access to Property selector based on Property Category security.	**Run Import**—Users can view and run any Standard import. Users have restricted access to Property selector based on Property Category security.
	Manage Standard Imports—Users have access to Import menu options for Standard imports. Users have restricted access to Property selector based on Property Category security.	NA

Blender Permissions

Table 4-6 Blender Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage System Blenders—Users have access to system blenders and Blender menu options. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	**Manage User Blenders**—Users have access to view and run User and Standard blenders. Users do not have access to Blender menu options for Standard Blenders.	**Run Blender**—Users can view and run any Standard blender. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.

Table 4-6 (Cont.) Blender Permissions

Permission Level 1	Permission Level 2	Permission Level 3
	Manage Standard Blenders—Users have access to Blender menu options for Standard blenders. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	NA

Export Permissions

Table 4-7 Export Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage System Exports—Users have access to system exports and Export menu options. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	**Manage User Exports**—Users have access to view and run User and Standard exports and books. Users do not have access to Export menu options for Standard exports and books. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security	**Run Export**—Users can view and run any Standard exports. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.
	Manage Standard Exports—Users have access to Export menu options for Standard exports and books. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	NA

Script Permissions

Table 4-8 Script Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Run Action Script—Users can run action scripts. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	NA	NA

Audit Permissions

Table 4-9 Audit User Transaction Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Audit User Transactions—Users can query any transactions that they performed. Transactions can include data and metadata changes and logged actions such as Login and running asynchronous operations. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	NA	NA

Table 4-10 Audit Data Transaction Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Audit Data Transactions—Users can query any transactions for data objects they have access to in Permissions or Node Access Groups. Transactions can include transactions performed by the user and changes made by other users. For node-level transactions, users can query transactions for a node and all of its descendants (Include Child Nodes option), assuming the user also has read access to all descendants. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.	NA	NA

Table 4-11 Audit System Transaction Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Audit System Transactions—Users can query any transactions that they performed. Transactions can include data and metadata changes and logged actions such as Login and running asynchronous operations.	NA	NA

Application Permissions

Table 4-12 Application Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage Application	**Manage Categories**	**Browse Categories**—Users have access to any property category that they are granted rights to in Property Category security.
	Manage Properties	**Browse Properties**—Users have access to all properties for the property categories that they are granted rights to in Property Category security.
		Manage Property Lists—User can manage lists of values and lookup tables for property definitions.
	Manage Validations	NA
	Manage Node Types	NA
	Manage Preferences	NA

Access Permissions

Table 4-13 Access Permissions

Permission Level 1	Permission Level 2	Permission Level 3
Manage Access	**Manage Users**—Users cannot edit or delete their own user profile.	NA
	Manage Roles—Users cannot edit their own role assignment.	NA
	Manage Access Groups—Users cannot edit their own Node Access Group assignment.	NA
	Manage Property Access—Users cannot edit their own Property Category assignment.	NA

User Roles

Oracle Data Relationship Management permissions are assigned to users using Roles. Each user role is associated with a set of permissions that provide access to product features or data. A user can be assigned one or more roles which grants them the combined permissions from all roles. If a user is assigned two roles that have conflicting levels of access, the user is granted the higher level of access.

Data Relationship Management provides the following user roles with assigned permissions marked:

Table 4-14 User Roles - Permissions

Permissions			User Roles							
Level 1	**Level 2**	**Level 3**	**Access Manager**	**Anonymous User**	**Application Administrator**	**Data Creator**	**Data Manager**	**Interactive User**	**Workflow User**	**Governance User**
Manage Versions							X			
	Browse Versions		X	X	X	X		X	X	X
	Create Versions					X				
	Manage Hierarchies						X			
		Browse Hierarchies	X	X	X	X		X	X	X
		Create Hierarchies				X				
Manage Requests							X			
	Create Requests				X				X	
Manage System Queries					X					
	Manage User Queries					X	X	X		
		Run Query		X					X	
	Manage Standard Queries						X			
Manage System Compares					X					
	Manage User Compares					X	X	X		
		Run Compare		X					X	
	Manage Standard Compares						X			

Table 4-14 (Cont.) User Roles - Permissions

Permissions			User Roles							
Level 1	**Level 2**	**Level 3**	**Access Manager**	**Anonymous User**	**Application Administrator**	**Data Creator**	**Data Manager**	**Interactive User**	**Workflow User**	**Governance User**
Manage System Imports					X					
	Manage User Imports					X	X			
		Run Import								
	Manage Standard Imports						X			
Manage System Blenders					X					
	Manage User Blenders					X	X			
		Run Blender								
	Manage Standard Blenders						X			
Manage System Exports					X					
	Manage User Exports					X	X	X		
		Run Export		X					X	
	Manage Standard Exports						X			
Run Action Script					X	X	X	X		
Audit User Transactions			X		X	X	X	X	X	
Audit Data Transactions					X	X	X	X		

Table 4-14 (Cont.) User Roles - Permissions

Permissions			User Roles							
Level 1	**Level 2**	**Level 3**	**Access Manager**	**Anonymous User**	**Application Administrator**	**Data Creator**	**Data Manager**	**Interactive User**	**Workflow User**	**Governance User**
Audit System Transactions			X		X					
Manage Application					X					
	Manage Categories									
		Browse Categories	X	X		X	X	X	X	X
	Manage Properties									
		Browse Properties	X	X		X	X	X	X	X
		Manage Property Lists					X			
	Manage Validations									
	Manage Node Types				X					
	Manage Preferences									
Manage Access										
	Manage Users		X							
	Manage Roles		X							
	Manage Access Groups		X							
	Manage Property Access		X							

Table 4-14 (Cont.) User Roles - Permissions

Permissions			User Roles							
Level 1	**Level 2**	**Level 3**	**Access Manager**	**Anonymous User**	**Application Administrator**	**Data Creator**	**Data Manager**	**Interactive User**	**Workflow User**	**Governance User**
Workflow Participant										X

Analytics Roles

Oracle Data Relationship Management Analytics roles can be combined to support multiple functions. For example, a user with the Analytics User, Governance Manager, and Data Manager roles would have access to all dashboards and the admin console. A user with the Access Manager and Application Administrator roles would have access to all reports.

Table 4-15 Analytics Roles for Dashboards and Admin Console

Role	Dashboards and Admin Console					Permissions
	Request	Model	Change	Growth	Admin Console	
Analytics User			X	X		Browse Version and Hierarchies
Governance Manager	X	X				Browse Version and Hierarchies
Application Administrator					X	N/A
Data Manager					X	N/A

Table 4-16 Analytics Roles for Reports

Role	Reports						
	User Role Assignment	Access Group Membership	Hierarchy Access Group Assignment	Workflow Access Group Assignment	Object Access Group Authorization	User Login Activity	Metadata Object Usage
Access Manager	X	X		X	X	X	
Application Administrator			X	X			X
Data Manager			X	X			

Creating Users

When you create users, you define a unique name and assign roles. If a user is not assigned the Data Manager role, node access groups and property categories can be assigned to the user to control their access to data.

> **Note**
>
> The @@ prefix on a user ID indicates an internal only user. You cannot create users with this prefix. Other @@ users include @@SYSTEM and @@STANDARD.

To create users:

1. On the Home page, select **Administer**.
2. From **New**, select **User**.
3. Enter a unique user name and the full name of the user.

   > **Note**
   >
   > Department, Phone, and Email Address are optional. Data governance workflow users must have an email address configured to receive email notifications.

4. If mixed authentication is enabled for a Oracle Data Relationship Management application, select the authentication method for the user.
   - **Internal**—User is authenticated within Data Relationship Management.
   - **CSS (External)**—User is authenticated externally via Oracle Hyperion Shared Services.
5. **Optional:** Select from the following options:
   - **Password does not expire**—PasswordDuration system preference setting is ignored.
   - **Login session does not expire**—IdleTime system preference setting is ignored.

     > **Note**
     >
     > If this option is selected, the maximum allowable idle time is 24 hours. After 24 hours of idle time, the login session expires.

   - **User is exempt from lockout measures**—lockout restrictions are disregarded for this user.
6. On the **Roles** tab, select roles from the **Available** list to assign to the user. Use the arrows to move roles to the **Selected** list.

Note

For additional information on roles, see User Roles.

7. On the **Node Access Groups** tab, select groups from the **Available** list to assign to the user. Use the arrows to move the groups to the **Selected** list.
8. On the **Property Categories** tab. select categories from the **Available** list to assign to the user. Use the arrows to move the categories to the **Selected** list
9. For each category in the selected list, do the following:
   a. Click in the **Action** column and set the user's access (Read or Edit) to the category.
   b. Select in the **Action** column to save the change.
10. Click .

    The Change Password dialog box is displayed.
11. Enter a password for the user.
12. Enter the password again.
13. **Optional:** Select **User must change password at next login** to require the user to change their password the next time they log in.
14. Click **OK**.

User Authentication

Oracle Data Relationship Management supports users that are natively authenticated by the application using stored password information or users that are authenticated by an external user directory. Each Data Relationship Management application is configured to support one or both types of users.

You set up application authentication on the Authentication Settings tab of the Data Relationship Management Console. For more information, see the *Oracle Data Relationship Management Installation Guide*.

Values defined for the following system preferences determine the characteristics of user passwords and when passwords expire for internal authenticated users:

- PasswordPolicyEnabled—If enabled, the password must contain three of the following elements:
  – Uppercase letters
  – Lowercase letters
  – Numbers
  – Special characters
- PasswordMaxLength—Determines the maximum character length for passwords.
- PasswordMinLength—Determines the minimum character length for passwords.
- PasswordDuration—Determines the number of days a password is valid.
- PasswordWarningPeriod—Indicates how many days before (-) or after (+) the password expiration date to warn users to change their password before no longer allowing them to log in. A negative value, for example -3, indicates the user is warned at login during the 3

days prior to password expiration. A positive value, for example 5, indicates the user is warned at login during the 5 days after their password has expired. After the five-day period, the user cannot login without changing the password.

Note

Changes to the PasswordDuration and PasswordWarningPeriod values do not affect users until the next password change. For example, if PasswordDuration is set to 30 days and the password for User1 was changed 26 days ago, the password expires in 4 days. If you change the PasswordDuration value to 60 days, the password for User1 still expires in 4 days. After the user changes the password, the new password expires in 60 days.

Modifying Users

You can change a user password, lockout or unlock a user, or change role, group, or category assignments.

Changing Passwords

To change a user password:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Users**.
3. Select a user and click .
4. Click .
5. Enter a new password for the user.
6. Enter the password again.
7. **Optional:** Select **User must change password at next login** to require the user to change their password the next time they log in.
8. Click **OK**.

Locking Out Users

You can lockout a user to prevent their access to a Oracle Data Relationship Management application. When you lockout a user, you can provide a custom reason for the lockout. This reason is displayed to the user when attempting to log into the application.

To lock out a user:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Users**.
3. Select a user and click .
4. Click .
5. Enter a reason for the lockout.

6. Click **OK**.

Unlocking Users

Unlocking a locked out user will enable their access to the application.

To unlock a user:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Users**.
3. Select a user and click ✎.
4. Click 🔓.
5. Click **OK**.

Changing User Roles and Assignments

To change user roles and assignments:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Users**.
3. Select a user and click ✎.
4. On the **Roles** tab, select roles from the **Available** list to assign to the user. Use the arrows to move roles to the **Selected** list.
5. On the **Node Access Groups** tab, select groups from the **Available** list to assign to the user. Use the arrows to move the groups to the **Selected** list.
6. On the **Property Categories** tab. select categories from the **Available** list to assign to the user. Use the arrows to move the categories to the **Selected** list.
7. For each category in the selected list, do the following:
   a. Click ✎ and set the user's access (Read or Edit) to the category.
   b. Select 💾↶ to save the change.
8. Click 💾.

Deleting Users

Users that are no longer active can be deleted from an application. When a user is deleted, all of the user-level metadata objects associated with the user are also deleted. These metadata objects include queries, compares, imports, blenders, exports, and books.

To delete a user:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Users**.
3. Select a user and click ✖.
4. Click **Delete this Item** to confirm the deletion.

Viewing User Login Status

For each user, you can view login statistics and information:

- The date and time of the user's last valid login
- The number of invalid login attempts
- Whether the user is locked out
- The date and time the user was locked out
- The reason for the lockout

To view user login status:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Users**.
3. Select a user and click .
4. Select the **Login Status** tab.

System Defined Users

Oracle Data Relationship Management applications include four default users which are added during the creation of an application repository.

- **ADMIN**—The default administrative user for an application. The password for this user is initially configured during the repository creation process.
- **@PROCESS**—An internal user set up to handle inter-process communication between server components. This user is not accessible or configurable in the Web client. Transactions are logged for this user each time an application engine is started.
- **@STANDARD**—An internal user set up to manage user metadata objects in the Standard object access group. This user is not accessible or configurable in the Web client.
- **@SYSTEM**—An internal user set up to manage user metadata objects in the System object access group. This user is not accessible or configurable in the Web client.

Common User Provisioning

The Common User Provisioning feature enables users and groups to be provisioned to Oracle Data Relationship Management applications using Oracle Hyperion Shared Services. This configuration allows Data Relationship Management users to be provisioned in a common location along with other Oracle EPM applications. Common User Provisioning also eliminates the need to separately provision users in the Data Relationship Management application. Provisioning information can be synchronized from Shared Services to Data Relationship Management on-demand or a scheduled basis.

When a synchronization takes place, the following actions are performed in Data Relationship Management:

- Add or update users
  - User name
  - Full name

  - Email address
- Assign roles to users
- Assign users to node access groups
- Assign users to property categories
- Remove user roles (if de-provisioned in Shared Services

When you enable Common User Provisioning, all external Data Relationship Management users and their roles are managed in Shared Services and cannot be managed in Data Relationship Management.

Prerequisites

Common User Provisioning is disabled by default in Oracle Data Relationship Management and should only be turned on after completing the following prerequisite steps:

1. Add Data Relationship Management user roles in Oracle Hyperion Shared Services—See "Configuring Shared Services Database with Data Relationship Management User Roles" in *Oracle Data Relationship Management Installation Guide*.
2. Register Data Relationship Management applications with Shared Services—See "Configuring EPM Registry Settings" in *Oracle Data Relationship Management Installation Guide*.
3. Enable Common User Provisioning—See "Configuring Common User Provisioning" in *Oracle Data Relationship Management Installation Guide*.

Provisioning Users and Groups

Any user or group accessible in Oracle Hyperion Shared Services can be provisioned for a Oracle Data Relationship Management application using Common User Provisioning. Groups (containing groups and/or users) and individual users may be provisioned for a Data Relationship Management application. Users and groups provisioned for a Data Relationship Management application in Shared Services, are synchronized in Data Relationship Management when a synchronization task is run. Users can be provisioned separately to multiple registered Data Relationship Management applications.

See "Provisioning Users and Groups" in the Oracle EPM System User Security Administration Guide.

Synchronizing Data Relationship Management Users and Group Membership

Full synchronization of user and group changes from Oracle Hyperion Shared Services to Oracle Data Relationship Management application can be performed manually or scheduled to run in the background. The synchronization creates or updates users in the Data Relationship Management application and updates group membership on node access groups or property categories that are configured to be managed externally.

The results for a synchronization display how many users were created and updated, how many node access groups were updated, and how many property categories were updated. A list of error and warning messages that were generated while running the synchronization are also displayed. You can copy the results and paste to an external editor for further review or usage.

Manual Synchronization

In Oracle Data Relationship Management, when Common User Provisioning is enabled, a user with the Access Manager role can manually synchronize users and groups managed in Oracle Hyperion Shared Services. The results of the job are displayed and can also be viewed on the Jobs page of the Audit task.

To manually synchronize users and groups:

1. On the Home page, select **Administer**.
2. From the toolbar, select (Common User Provisioning Synchronize).

Scheduled Synchronization

In Oracle Data Relationship Management, when Common User Provisioning is enabled, synchronization can be scheduled to run in the background at a specified time every 24 hours. The results of the scheduled job can be viewed by navigating to the job on the Jobs page of the Audit task.

- For information on viewing jobs, see "Viewing Job History" in *Oracle Data Relationship Management User's Guide*.
- For information on scheduling synchronization, see "Configuring Common User Provisioning" in *Oracle Data Relationship Management Installation Guide*.

Partial Synchronization

Partial, real-time synchronization is performed automatically in these scenarios for users and groups managed in Oracle Hyperion Shared Services:

- User Login—Provisioning information for the individual user being authenticated is automatically synchronized before a session is created.
- Node Access Group Membership—User membership for an individual node access group is automatically synchronized when the group is saved.
- Property Category Membership—User membership for an individual property category is automatically synchronized when the category is saved.

5

Managing Node Access Groups

Oracle Data Relationship Management controls granular user access to hierarchy nodes and their properties using node access groups. You can assign users to groups that are granted access to specific nodes in a subset of hierarchies within a Data Relationship Management version. Node access groups use inheritance to assign similar access to descendant nodes of a hierarchy node where an access level has been explicitly assigned. This level of access can be overridden at a lower level or can be locked to prevent overrides.

Typically, node access groups represent functional areas of an organization, and a user may require assignment to multiple groups. If assigned access levels conflict, the highest security level is used.

There are two types of node access groups. The group type controls the type of data access that can be assigned to users of that group. Each node access group can be of only a single group type.

- Interactive—Users have direct access to browse, search, and modify data based on the level of access assigned
- Workflow—Users have restricted access to browse, search, and modify data using governance workflows based on the level of access assigned

Table 5-1 Interactive Group Type–Node Access Levels

Level	Description	Example Usage
Read	Enables read-only access—no changes permitted	View and report
LimitedInsert	Enables insertion of a node for which the user has (at least) global insert privilege.	Insert
Edit	Enables property values to be edited	Edit
Insert	Enables nodes to be inserted, moved, or, removed	Edit, insert, copy, move, remove
Inactivate	Enables nodes to be inactivated and reactivated	Edit, insert, move, remove, inactivate, reactivate
Add	Enables nodes to be added or deleted	Edit, insert, copy, move, remove, inactivate, reactivate, add, delete

Keep the following information in mind:

- Access levels are cumulative; assignment of the Edit access level implies that the Read Only and LimitedInsert access levels are granted. Assignment of the Add access level implies that all other access levels are granted.
- Node access group security is only applied at the hierarchy level. Node access groups do not control access to global lists of nodes such as orphans.
- Access levels are assigned separately for limb and leaf nodes which allows you to define a different level of access for each. This capability is useful when a user should be able to

maintain the roll-up structure of a hierarchy but not edit any properties of leaf nodes or when a user can insert leaf nodes to an existing roll-up structure but not reorganize the structure itself.

- Node access groups are defined only by a user with the Access Manager role.
- Node access groups use local inheritance for access assignment to related nodes. A node access group can be defined as global in order to use global inheritance based on the level of access assigned to a controlling hierarchy.
- Global node access groups can be created and must have a controlling hierarchy defined for each version. This is done by assigning controlled node access groups to a hierarchy. See the see the *Oracle Data Relationship Management User's Guide* for more information.
- Interactive and Workflow node access groups handle the visibility of nodes in hierarchies differently. An interactive access group provides users visibility to the entire hierarchy if the group has access to any node in the hierarchy. In contrast, a workflow access group provides users limited visibility to only nodes in hierarchies to which they have been assigned access. For both group types, members of the group cannot view hierarchies to which they have not been assigned access.

Workflow Group Type Node Access Levels

Users with the Governance User role use the Workflow node access levels to determine their access to data.

Table 5-2 Workflow Group Type–Node Access Levels

Level	Description
Notify	Enables notification of change requests for a node
Submit	Enables nodes to be submitted as part of a change request
Approve	Enables nodes to be approved as part of a change request
Enrich	Enables nodes to be enriched as part of a change request
Commit	Enables changes for a node to be committed to Oracle Data Relationship Management

Workflow node access levels are cumulative for hierarchy access but are also filtered by workflow stage.

Table 5-3 Workflow Node Access Levels by Hierarchy Access

Hierarchy Access	Stage Access			
Access	**Submit**	**Approve**	**Enrich**	**Commit**
Notify	Notify	Notify	Notify	Notify
Submit	Submit	Notify	Notify	Notify
Approve	Submit	Approve	Notify	Notify
Enrich	Submit	Approve	Enrich	Notify
Commit	Submit	Approve	Enrich	Commit

Creating Node Access Groups

To create a node access group:

1. On the Home page, select **Administer**.
2. From **New**, select **Node Access Group**.
3. Enter a name, label, and description for the group.

   Note

   The node access group will be assigned to the Custom namespace. The Fully Qualified Name for the group must be unique. The Label field is filled in automatically after entering the name. The node access group label is a user-friendly descriptor that is displayed for all features aside of application administration. Multiple node access groups can have the same Label for convenience purposes.

4. Select a **Group Type** for the node access group.
   - **Interactive**—To use interactive access levels; see Interactive Node Access Levels.
   - **Workflow**—To use workflow-oriented access to versions, hierarchies, and nodes in the context of submitting, enriching, approving, committing, and being notified of requests. See Workflow Node Access Levels.
5. **Optional**: Select **Global** to make the group a global node access group.

   Note

   Global node access groups must have a controlling hierarchy defined in every version where the group will be used. After a group is created, you can assign it to a single hierarchy in each version as a controlled node access group.

6. If using Common User Provisioning, from **External Group** select a user group provisioned to the Oracle Data Relationship Management application in Oracle Hyperion Shared Services. Users in this external group will be assigned membership to the node access group when a synchronization from Shared Services takes place.
7. Select users from the **Available** list to assign to the group. Use the arrows to move users to the **Selected** list.
8. Click .

Editing Node Access Groups

To edit a node access group:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Node Access Groups**.
3. Select a group and click .

4. Select users from the **Available** list to assign to the group. Use the arrows to move users to the **Selected** list.
5. Click .

Deleting Node Access Groups

To delete a node access group:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Node Access Groups**.
3. Select a group and click .
4. Click **Delete this Item** to confirm the deletion.

> **Note**
>
> Deleting a node access group removes the assignment of the group from the users as well as from any hierarchy nodes.

Assigning Node Access Group Security

Node Access Group security is applied to data by a user with the Data Manager role.

> **Note**
>
> Before assigning node access group security, ensure that appropriate node access groups are created and appropriate users are assigned to the groups.

To set node access group security:

1. Open a version and hierarchy, and select a node.
2. From **Nodes**, select **Assign**, then **Node Access**.
3. In the Property Grid, select the Leaf Access or Limb Access category.
4. Assign the level of access for each node access group.

   The level of access available for assignment for each node access group is based on its group type (interactive or workflow).
5. Click **Save**.

> **Note**
>
> You must assign both Limb and Leaf Workflow NAG access to at least one Leaf and one antecedent Limb node to a value other than "None" to be able to visualize node(s) to choose in the DRG Node Selector. Typically Limb WNAG access is set on the Top Node in a hierarchy.

6 Managing Object Access Groups

Object access groups in Oracle Data Relationship Management determine which metadata objects users have access to, including exports, books, imports, blenders, compares, queries, version variables, and external connections.

Table 6-1 Types of Object Access Groups

Object Access Group Type	Description	Permissions
User	Each user has a core object access group for their personal metadata objects.	A user has Run and Manage permissions to their own object access group.
Standard	A core object access group named Standard is available for all public objects.	All users have implicit Run permission to objects in the Standard object access group. Only users with Manage Standard [Object] role permissions have Manage permission for the Standard object access group.
System	A core object access group named System is available for all system operation/integration objects.	Only users with Data Manager or Application Administrator roles have Manage permission for the System object access group.
Custom	Custom object access groups	Only users with Access Manager role can create, edit, or delete custom object access groups. Users with Run permission may execute objects in the group.

Custom object access groups provide a specific group of users access to a subset of user metadata objects – queries, compares, imports, blenders, exports, and books. Object access groups define a list of users and node access groups and set the permission level (Run or Manage) for each user and node access group. Metadata objects are assigned to object access groups at the time they are created, and they may subsequently be copied or moved to a different group.

- Run—Users can run objects in the group but cannot edit and save changes to the objects
- Manage—Users can create, edit, or delete objects in the group and run them

Guidelines for using object access groups are:

- An object access group enables users to be members of the group either directly or through their node access group assignments. Both are not required.
- Users and node access groups may be assigned to more than one object access group.
- Each user in the object access group is assigned either Manage or Run permission.
- A user's permission assignment in the object access group may override the user's role security. For example, an Interactive User role with Manage permission in an object access group may create or modify objects within the object access group.

- Core object access groups such as User, Standard, and System are managed implicitly based on user existence and their role assignments.
- When saving or copying a user metadata object, the user must assign the object to an object access group for which that user has Manage permission.
- A user metadata object may be assigned to only one object access group.
- Data Manager role users have implicit Manage permission to the core Standard object access group and may be explicitly assigned to a custom object access group.
- Application Administrator role users have implicit Manage permission for all standard, system, and custom object access groups. These users require the ability to migrate metadata objects for any object access group.

Creating Object Access Groups

To create a custom object access group:

1. On the Home page, select **Administer**.
2. From **New**, select **Object Access Group**.
3. Enter a name for the group. A description is optional.
4. On the **Users** tab, select users from the **Available** list to assign to the group. Use the arrows to move users to the **Selected** list.

   Note

   By default, each user is granted Run access. To change a user's access, click ✎. Then from **Access**, select **Manage**.

5. On the **Node Access Groups** tab, select node access groups from the **Available** list to assign to the group. Use the arrows to move node access groups to the **Selected** list.

   Note

   By default, each node access group is granted Run access. To change a group's access, click ✎. Then from **Access**, select **Manage**.

6. Click 💾.

Editing Object Access Groups

To edit a custom object access group:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Object Access Groups**.
3. Select a group, and then click ✎.
4. On the **Users** and **Node Access Groups** tabs, make changes to selected users and groups and to access permissions.

5. Click .

Deleting Object Access Groups

To delete an object access group:

1. On the Home page, select **Administer**.
2. Under **Security**, expand **Object Access Groups**.
3. Select a group, and then click .
4. Click **Delete this Object Access Group** to confirm the deletion.

> **Caution**
>
> When an object access group is deleted, all metadata objects assigned to it are also deleted. This operation cannot be undone

7 Managing Domains

Domains are used to manage referential integrity for multiple sets of nodes from different sources within the same Oracle Data Relationship Management application. A domain is a registered list of nodes of a common type which enables consistent management of these nodes in different versions within the same application. A domain provides a simple method for:

- Qualifying node names to ensure uniqueness
- Sharing identifying properties across versions
- Restricting certain types of changes such as renaming, promoting, demoting, and deleting nodes
- Assigning validations to ensure consistency of business rules regardless of version

Domain nodes are global nodes in a version with membership to a domain. Domain nodes cannot be renamed and cannot be removed from a domain after being assigned as a member. A domain node must have a unique name, regardless of domain assignment. The name of a domain node may represent the natural identifier of the node or may be qualified with a prefix or suffix to ensure referential integrity when used with nodes of different domains in the same version. The domain node description and inactive status/date are shared by a domain node in any version where it exists.

Creating Domains

To create a domain:

1. From the Home page, select **Administer**.
2. From **New**, select **Domain**.
3. Enter the following information:
   - **Name**
   - **Description** (optional)
   - **Qualifier** (optional)—Text used for fully qualifying a node name. No two domains can use the same qualifier text. Select **Prefix** or **Suffix** to denote the location of the qualifier.

     Note

     After a domain has nodes assigned to it, the qualifier text cannot be changed.

   - **Delimiter** (optional)—A single, optional character used to separate the domain qualifier text from the node name.

> **Note**
>
> After a domain has nodes assigned to it, the delimiter cannot be changed.

- **Allow Node Delete**—Select if you want to give users the ability to delete nodes from the version.
- **Allow Leaf Edit**—Select if you want to give users the ability to change the leaf system property value for nodes in the domain.

4. From the **Available Validations** list, select the node-level validations to be enforced for members of the domain and move them to the **Selected Validations** list.

> **Note**
>
> Domain-level validation assignments override assignment values for the same validation that were set at the node or inherited from an ancestor node, hierarchy, or version level assignment.

5. Click .

Editing Domains

A domain may be edited after it is created, with two exceptions:

- The name cannot be changed
- The qualifier and delimiter cannot be changed after nodes have been assigned to it

To edit a domain:

1. From the Home page, select **Administer**.
2. Select a domain and click .
3. Make changes to the domain and click .

Deleting Domains

A domain may be deleted. The domain node records are also removed when the domain is deleted.

> **Note**
>
> If a domain with nodes assigned is deleted, all nodes that are assigned to the domain revert back to non-domain nodes.

To delete a domain:

1. From the Home page, select **Administer**.
2. Select a domain and click .

3. Select **Delete this Domain**.

8

Managing Property Categories

Related Topics

- Property Categories
- Creating Property Categories
- Editing Property Categories
- Deleting Property Categories

Property Categories

Property categories enable the grouping of Oracle Data Relationship Management properties and are used to control the assignment of security privileges to sets of properties. Core properties available by default are only located in a single property category. Custom properties created by application administrators can be associated with multiple property categories.

Data Relationship Management includes the core property categories described in the following table.

Table 8-1 Property Categories

Category	Description
System	Properties related to the basic identifying characteristics of a node, such as ID, name, and description. The only change that can be made to this category is assigning the read-only flag for individual users. Users with read access cannot edit values but can view them. Properties cannot be assigned to this category.
Shared Info	Provides information about which nodes are primary/shared, a list of related shared nodes, and identifies whether the primary node is missing. This category is only displayed when Shared Nodes is enabled via system preferences. **Note:** All properties in this category are read only.
Stats	Properties that provide statistical information about a node such as number of children and number or siblings **Note:** All properties in this category are read only.
Validation	Validations assigned for the node—one property for each validation
Leaf Access	Node security groups and their leaf access levels for the node—one property for each group
Limb Access	Node security groups and their limb access levels for the node—one property for each group

> **Note**
>
> Not all property categories are visible to all users because user access can be restricted to specific categories and the node types can be filtered. The Validation, Leaf Access, and Limb Access categories are available only to users assigned the Data Manager role and are only accessible when assigning validations or node access group security.

Creating Property Categories

To create a property category:

1. From the Home page, select **Administer**.
2. From **New**, select **Property Category**.
3. Enter a name and description for the property category.
4. If using Common User Provisioning, from **External Group - Edit** and **External Group - Read**, select a user group provisioned to the Oracle Data Relationship Management application in Oracle Hyperion Shared Services. Users in these external groups will be assigned membership to the property category with the specified level of access (edit or read) when a synchronization from Shared Services takes place.
5. On the **Properties** tab, select properties from the **Available** list to assign to the property category and use the arrows to move the properties to the **Selected** list.

   > **Note**
   >
   > You can use **Ctrl+Click** or **Shift+Click** to select multiple properties. Double-click a property to select or deselect it.

6. Use the arrows to reorder the selected properties or click to alphabetize the selected properties.
7. On the **Users** tab, select users from the **Available** list to assign to the property category and use the arrows to move the users to the **Selected** list.
8. Select the row for a user in the selected list and click in the **Action** column.
9. From the **Access** column, select Read or Edit to assign the user a level of access to the property category.
10. Click in the **Action** column to save the change or to discard the change.
11. Click .

Editing Property Categories

To edit a property category:

1. From the Home page, select **Administer**.
2. Select a property category and click .

3. On the **Properties** tab, select properties from the **Available** list to assign to the property category and use the arrows to move the properties to the **Selected** list.

> **Note**
>
> You can use **Ctrl+Click** or **Shift+Click** to select multiple properties. Double-click a property to select or deselect it.

4. Use the arrows to reorder the selected properties or click to alphabetize the selected properties.
5. On the **Users** tab, select users from the **Available** list to assign to the property category and use the arrows to move the users to the **Selected** list.
6. Select the row for a user in the selected list and click in the **Action** column.
7. From the **Access** column, select Read or Edit to assign the user a level of access to the property category.
8. Click in the **Action** column to save the change or to discard the change.
9. Click .

Deleting Property Categories

To delete a property category:

1. From the Home page, select **Administer**.
2. Under **Metadata**, expand **Property Categories**.
3. Select a property category and click .
4. Select **Delete this Item** to confirm the deletion.

> **Note**
>
> The deletion of a property category does not result in the deletion of properties associated with the category. These properties remain available within the application.

9

Managing Property Definitions

Property definitions are used to manage the attributes of versions, hierarchies, and nodes in Oracle Data Relationship Management. Properties can store a variety of different data types including text, numeric, date, and references to other data objects. Properties can store explicit values, use inheritance to automatically assign values to descendant nodes, or be calculated based on a formula or lookup table. Property categories can be used to group and organize properties into related sets to simplify their usage and control user access.

System-defined properties that are available by default are used with standard product functionality. User-defined property definitions can be created by application administrators to manage additional attributes that are necessary to support business or system integration requirements.

Property definitions in Data Relationship Management can come from a variety of sources. For example, properties can be:

- System-defined in Data Relationship Management
- User-defined properties created by an application administrator
- Loaded from application templates used with other Oracle products
- Loaded from another Data Relationship Management application or environment using the Migration Utility

Namespaces

Namespaces are used in property definitions to avoid conflicts where properties from different sources have similar names and need to remain separate for data integrity purposes. Property names are differentiated using a namespace prefixing convention.

Table 9-1 Property Definition Example Using Namespaces

Field	Example
Fully Qualified Name	Custom.AccountType
Namespace	Custom
Name	AccountType
Label	AccountType

There are special rules in Data Relationship Management that apply to namespaces to ensure that conflicts do not occur:

- System-defined properties use the "Core" namespace.
- User-defined properties use the "Custom" namespace.
- Other namespaces are reserved for use by Data Relationship Management application templates for other Oracle products.

Data Types

Property data types are described in the following table.

Table 9-2 Property Data Types

Property Data Type	Description
Associated Group	Associated node group. Points to multiple nodes. The nodes point back to the Associated Group node and to each other. Analogy: Fraternity. **Note:** This data type should only be used with global node level properties. **Caution:** Associated node properties that are loaded by an import may not correctly point to all other nodes as a result of their not yet existing in the version based on the order in which nodes are imported.
Associated Node	Associated node. Points to a single other node. The node pointed to points back to the Associated Node node. Analogy: Marriage. **Note:** This data type should only be used with global node level properties. **Caution:** Associated node properties that are loaded by an import may not correctly point to all other nodes as a result of their not yet existing in the version based on the order in which nodes are imported.
Associated Nodes	Associated node list. Points to multiple nodes. The nodes pointed to point back to the Associated Nodes but not each other. Analogy: Friends. **Note:** This data type should only be used with global node level properties. **Caution:** Associated node properties that are loaded by an import may not correctly point to all other nodes as a result of their not yet existing in the version based on the order in which nodes are imported.
Boolean	True or False
Date	Date values are formatted in the invariant culture. This allows for a predictable response and action can be taken to re-format the result, if desired. **Caution:** The default, maximum, and minimum values must be entered in English (United States) format.
Date/Time	Date and time values are formatted in the invariant culture. This allows for a predictable response and action can be taken to re-format the result, if desired. **Caution:** The default, maximum, and minimum values must be entered in English (United States) format.

Table 9-2 (Cont.) Property Data Types

Property Data Type	Description
Float	Floating point value is formatted based on the regional settings associated with the user's session. **Note:** If a default value is not defined, then 0 is output for the value when exported.
Formatted Memo	Formatted memo — retains all formatting (spaces, tabs, new lines, and so on) to the text. Also allows for hyperlink text to be included in the formatted memo. See the Hyperlink data type for details on formatting URLs for hyperlinks. **Note:** Non URL text is not suppressed when both text and hyperlink is used in property value.
Global Node	Points to a node in a version; when value is assigned it shows node name only in the value field of the property grid
Group	List of comma-delimited items
Hierarchy	Points to a hierarchy
Hierarchy Group	Points to a hierarchy group. Hierarchy group properties allow hierarchies to be grouped in multiple ways based on the context in which you want to view them. You can group hierarchies within the same version in different ways based on usage.
Hyperlink	Allows for hyperlink capability for URL text. Multiple URL input is separated by Carriage Return-Linefeed (CRLF, or 0x0D0A) with no spaces. Entered URLs are displayed as navigable hyperlinks. Only the parsed, delimited URLs or formatted URLs are displayed. URLs should follow this format: [url=*http_URL*]*URL_Title*[/url] where *http_URL* specifies the hyperlink text and *URL_Title* specifies the text displayed to the user. For example, this markup example: `[url=https://support.oracle.com]Oracle Support[/url]` would render in the property grid as My Oracle Support
Integer	Integer value If a default value is not defined, then 0 is output for the value when exported.
Leaf Node	Points to a leaf node in a hierarchy. When value is assigned it shows hierarchy name and node name in the value field of the Property Grid.
Limb Node	Points to a limb node in a hierarchy. When value is assigned it shows hierarchy name and node name in the value field of the Property Grid.
List Group	Check list of items. Multiple items can be selected from the list.

Table 9-2 (Cont.) Property Data Types

Property Data Type	Description
Memo	Memo field — formatting is not saved and data is merged into a single line of text. Also hyperlink in the memo. See the Hyperlink data type for details on formatting URLs for hyperlinks. **Note:** Non URL text is not suppressed when both text and hyperlink is used in property value.
Multiple Node	Points to multiple nodes
Node	Points to a node in a hierarchy; when value is assigned, it shows hierarchy name and node name in the value field of the property grid.
Node Properties	Points to the properties of a node
Property	Points to a property
Range List	Defines a range of values; accepts only integer values
Sort	Integer value that is used for sorting
Sort Property	Points to a Sort property
Standard Query	Points to a standard query
String	String value
Time	Time values are formatted in the invariant culture. This allows for a predictable response and action can be taken to re-format the result, if desired. **Caution:** The default, maximum, and minimum values must be entered in English (United States) format.
Version	Points to a version

External Lookups

External Lookup Properties are properties that access an external data source for their list of selectable values. The external data source is accessed using external operations. The external lookup property type allows for return of a recordset from Oracle or SQL Server databases. Use the results of an external lookup to select an item from an external list of values for use as a property value or to calculate request item property values using data from an external source. External lookups for property lists are accessible in Data Relationship Management and Data Relationship Governance.

Creating Properties

To create a property definition:

1. On the Home page, select **Administer**.
2. From **New**, select **Property Definition**.
3. Enter a name for the property.

Note

The property is assigned to the Custom namespace. The Fully Qualified Name and Label fields are filled in automatically after entering the name. The Fully Qualified Name for the property must be unique. The property label is a user-friendly descriptor that is displayed for property definitions for all features aside of application administration. Multiple properties can have the same Label as long as they are not in the same namespace. The property Description is an optional, long descriptor that is displayed at the bottom of the Property Editor.

4. Define parameters for the property:

Note

Not all parameters below are displayed. The parameters displayed depend on the selected data type.

- **Data Type**—See Property Data Types

  You can restrict the list of nodes displayed to a user by selecting a data type: Associated Group, Associated Node, Associated Nodes, Global Node, Leaf Node, Limb Node, Multiple Node, or Node. After you select a data type, the **Constraints** tab is displayed.
- **Property Level**—Level of property definition:
  - **Local node**—Property values are managed for nodes in a specific hierarchy and accessible only at this level.
  - **Global node**—Property values are managed for nodes in a version but also accessible at a local node level.
  - **Hierarchy**—Property values are managed for hierarchies but also accessible at a local node level.
  - **Version**—Property values are managed for versions but also accessible at a global or local node level.

  Note

  If defining a global node inherited property, you must define a controlling hierarchy for the global property. You do with on the Home page on the Hierarchies tab by assigning controlled properties to a hierarchy.
- **Property Type**
  - Defined—Values are defined by the user and stored.
  - Lookup—Lookup based on another property and a lookup table.
  - Derived—Calculated by using a Deriver class.

> **Note**
>
> Derived properties using the Script deriver class may be used for version, hierarchy, and node properties. The Formula deriver class may only be used for global or local node properties.

- External Lookup—Lookup using an external data source.

> **Note**
>
> Values are retrieved from an external data source in real-time. If multiple values are returned, a specific value must be selected for the property.

- **Default Value**—Default value for the property
- **Domain**—For any property where the data type is Node, Limb Node, LeafNode, MultiNode, Associated Node, Associated Nodes, or Associated Group (all of which represent a node or nodes stored as the value), a Domain drop-down is available. The drop-down contains all the domains defined in the system and you can optionally select one of the existing domains.
- **Column Width**—Width for fixed-width columns if the property type is Defined.
- **Minimum Value/Length**—Value or length for the property based on data type.
- **Maximum Value/Length**—Value or length for the property based on data type.

5. Select from these options:

- **Inherited**—Defines the property as Inheriting

> **Note**
>
> This option has no effect on the Derived property type except in the special case where property derivers, such as AncestorProp or DualAncestorProp, are used and the property is global. In such cases, although the property is not literally inheriting values, enable the Inherited option to allow the specification of a controlling hierarchy.

- **Overrideable**—Allows property to be overridden in the property grid.

> **Note**
>
> This option is enabled only for the Derived property type.

- **List**—Allows property values to be selected only from a predefined list of values.

> **Note**
>
> Property values stored for a list property can be limited to only values in the list using the EnforceListProps system preference.

> **Note**
>
> A list of values can be used for a defined property or a derived, overrideable property.

- **Hidden**—Hides the property in the property grid.
- **Indexed**—Creates an index for the property to improve performance of searches, property queries, and validations. This option is available only for defined, string data type properties.

> **Note**
>
> Indexed properties can increase memory usage on the application server and should only be used for properties most likely to be used in searches, queries, and validations that check uniqueness.

6. Do any of the following:
   - To assign a property to categories, select categories from the **Available** list and move them to the **Selected** list.
   - If you selected the **Defined** property type along with the **List** option, on the **List Values** tab do the following:
     a. Click **Add** and enter a value to the list.
     b. Click **Save** in the Action column for the row.

     > **Note**
     >
     > Use Move or Delete for each row to reorder or delete list values. Use Edit or double-click a row to edit it and Cancel to cancel edits.
   - If you selected the **Lookup** property type, select the **Lookup Table** tab and do the following:
     a. Click **Add** to enter a new key-value pair to the list.
     b. Click **Save** in the Action column for the row.

     > **Note**
     >
     > Use Move or Delete for each row to reorder or delete list values. Use Edit or double-click a row to edit it and Cancel to cancel edits.
   - If you selected a data type that allows hierarchy constraints, select the **Constraints** tab and do the following:
     a. Select a property from **Hierarchy Group Property** and then select a hierarchy group.

        In the node selector, users will see nodes only from hierarchies that belong to the selected hierarchy group.

> **Note**
>
> Only the default Core property type is supported by Oracle Data Relationship Management Analytics.

b. **Optional:** Select **Enforce Constraint on Server Property Update** to validate this constraint when the property is updated via the Web client, imports, action scripts, or the Web Service API.

- If you selected the **Derived** property type, select the **Parameters** tab and define a formula or script for the derived property.

  For more information on formulas, see Creating Formulas. For more information on scripts, see Creating Dynamic Scripts.

- If you selected the **External Lookup** property type, select the **External Lookup** tab and enter the following information:
  - **External connection**—Select a database or Web service connection
  - **Operation**—Select the external operation to perform
  - For each parameter configure:
    - **Parameter source type**—Select Literal or Property.
    - **Source**—If **Literal** was selected for source type, then enter a literal value in the Param Source column. When the external operation is called for this External Lookup property, the literal value is passed in for the current parameters. If **Property** was selected for source type, then select a property to provide the parameter value for the external operation. When the External Lookup is executed, the parameter value comes from the selected property on the current node or request item.
  - In **Column/Property Mappings**, select which result column in the selected lookup result will supply the value for the external lookup property. Click **Add** to add additional columns which can be mapped to different properties, so that when the external lookup value is selected, other property values get updated automatically.

    The first Column/Property mapping is automatically defined and cannot be deleted. This mapping is for the current property. A column must be selected, and defaults to the first column stored on the operation. You can modify the column value for the first row but not the property value. For additional mappings, you can select and edit the Column Name and the Result Column.

7. Click .

Using Hierarchy Constraints

Hierarchy constraints can limit the hierarchies and nodes available for viewing and selection when updating a node data type property value. A hierarchy constraint is an optional configuration for property definitions which use a node data type. The hierarchy constraint feature uses hierarchy groups and hierarchy group properties, which must be configured before you can assign hierarchy constraints.

You can use a hierarchy constraint with the following data types:

- Associated Group
- Associated Node

- Associated Nodes
- Global Node
- Leaf Node
- Limb Node
- Multiple Node
- Node

> **Note**
>
> The Associated Group, Associated Node, and Associated Nodes node data types may require additional consideration when setting up a hierarchy constraint because associated nodes create a cross-reference. If a hierarchy constraint is defined, care should be taken that the hierarchy group include all hierarchies that may be associated with one another. An example is a cross-reference between nodes in Employee and Cost-Center hierarchies. It may be necessary to create a separate hierarchy group property and hierarchy group to be used for hierarchy constraints.

Editing Property Definitions

If a property definition is modified from a Defined property type to a non-editable type such as Derived or Lookup, then the following conditions apply:

- The confirmation message when switching to a non-stored property type is modified to state that pending updates to change request items may be affected.
- Pending property updates for in-flight requests are no longer displayed, validated or committed for items with that task assigned.

To edit a property definition:

1. On the Home page, select **Administer**.
2. Under **Metadata**, expand **Property Definitions**.
3. Expand **Core** or **Custom** depending on the type of property definition.
4. Double-click a property.
5. Modify any parameters that can be edited.

   > **Caution**
   >
   > If you change the Property Type from a defined value (RWDerived or Defined) to a value that does not allow storage (Derived or Lookup), defined property values are deleted and this data will be lost. Before making this type of change, you must confirm that the potential for data loss is acceptable.

   For more information see Creating Properties.
6. Click .

Deleting Properties

If a property definition is deleted from Oracle Data Relationship Management, then the following conditions apply:

- The dependency check for property definitions is modified to include workflow metadata references, and the user must confirm the deletion. Property definition dependencies for workflow metadata consist of the following
  - Workflow Task Properties
  - Workflow Task Validation Properties
  - Change Request Item Details
- Upon confirmation, if a property is deleted, then each dependent reference to the property is also deleted, including assignment to workflow tasks, pending updates to in-flight requests, and historical change requests.
- As with interactive deletion of a property definition, transaction history is always retained.

To delete a property:

1. From the Home page, select **Administer**.
2. Under **Metadata**, expand **Property Definitions**.
3. Select a property and click ✖.
4. Select **Delete Property Definition** to confirm the deletion.

Caution

The deletion of a property definition will also result in the deletion of all values stored for the property as well as the removal of the property from all metadata objects where it was being used.

10

Managing Validations

Validations enable business rules to be enforced on versions, hierarchies, nodes, and properties. Validations can be run in either real time or batch, or both modes. Real-time validations are run at the time of modification and prevent changes from being saved if the action would violate the rules being enforced. Batch validations can be explicitly run before or after edits are made to identify data conditions that are invalid and need to be addressed.

Validation Classes

Validation classes allow different types of business rules to be enforced. Some validation classes can be used generically while other classes are used for specific purposes. Validations can be created from a set of existing validation classes. Many business rules on nodes can be enforced with a validation class that uses a query for its logic. This enables validations to leverage queries that have been created for analysis purposes to also manage data integrity. Rules for versions and hierarchies or special cases for nodes can be accomplished using other validation classes. A few of the validation classes are used for product testing purposes only and should not be used in a production environment.

Table 10-1 Validation Classes

Validation Class	Level	Description	Parameters
BoolNodeInHier	Node	Verifies that the specified boolean property has no True values in the specified hierarchy	Property, Hierarchy
ContainAllProp	Global Node	Verifies that the specified hierarchy contains all nodes where the specified property is True	Hierarchy, Property
ContainAllWith	Global Node	Verifies that the specified hierarchy contains all nodes for which the specified property has the specified value	Hierarchy, Property, Value
CustPropQuery	Node	Verifies using predefined query and expected result Only a local property query can be used.	Property query name, Failure value
DateRangeCheck	Node	Verifies that the From Date is earlier than or equal to the To Date	From Date Property, To Date Property
Formula	Node	Verifies a node using business logic expressed in a formula. A formula result of False results in a validation failure.	Formula

Table 10-1 (Cont.) Validation Classes

Validation Class	Level	Description	Parameters
GlobalPropQuery	Global Node	Verifies using predefined query and expected result	Property query name, Failure value
HierContainsRef	Node	Hierarchy contains a reference to the node when a Boolean property is True, or if the node is a leaf node and a third Boolean property is True.	Hierarchy name, Boolean property for all nodes, Boolean property for leaf nodes
HierFail	Hierarchy	Automatically fails at hierarchy level for testing purposes	none
InvalidNameLength	Node	Verifies that the node name is not equal to a specified length.	Length
MaxChildren	Version	Verifies that the number of children per node do not exceed specified limit	Maximum number of children
MaxHierNodes	Hierarchy	Verifies that the number of nodes in the hierarchy does not exceed specified limit	Maximum number of nodes
MaxVersionNodes	Version	Verifies that the number of nodes in the version does not exceed specified limit	Maximum number of nodes
MergeEquiv	Merge	Verifies that the affected node and merge node have the same value for the specified property	Global node property
MergePropSet	Merge	Verifies that if the affected node property value is set (overridden), the merge node property value is set for the specified property (Property values need not be the same)	Property
MixedKids	Node	Checks for nodes with both limb and leaf children.	None
NoBoolBranch	Node	Verifies that the specified boolean property is set to True at least once on a specified branch	Property
NodeFail	Global Node	Automatically fails nodes at the version level for testing purposes	none
NodeFailRandom	Node	Automatically fails the specified percentage of nodes for testing purposes	Failure percentage
NoDefaults	Node	Verifies that no default values are used for the specified property	Property

Table 10-1 (Cont.) Validation Classes

Validation Class	Level	Description	Parameters
NoPropBranch	Node	Verifies that the specified property is set at least once on a specified branch	Property
PropEquivBool	Node	Property equivalency when a third boolean property is True.	Boolean property to evaluate, First Property, Second Property
PropLength	Node	Verifies that the specified property is at least minimum length and no more than maximum length	Property, Minimum Length, Maximum Length
PropRemove	Remove	Prevents the removal of a node if the property or properties specified (in the prop1, prop2 and prop3 parameters) are equal to the specified values (in the value1, value2, value3 parameters).	Property1, Property2, Property3, Value1, Value2, Value3
RequiredField	Node	Verifies that, for all nodes for which the specified property has a specified value, each property in the required list has a value: • If the Reject Default Records flag is True, each property in required list must have a value other than the default • If the Reject Default Records flag is False, then default values are acceptable	Property, Value, Reject Default Records, Required Properties
Script	Node, Hierarchy, Version, Global Nodes, Move, Remove, Merge	Verifies data using a dynamic script. A return value of True passes the validation. A return value of False results in failure of the validation.	Script
SingleBoolBranch	Node	Verifies that the specified boolean property is set to True only once per branch	Property
SinglePropBranch	Node	Verifies that the specified property is set only once per branch	Property
StrandedParent	Node	Verifies that all limb nodes have children	none
StrPropEqual	Node	Fails for all nodes for which the specified property equals the specified value	Property, Value

Table 10-1 (Cont.) Validation Classes

Validation Class	Level	Description	Parameters
UniqueProp	Node	Verifies that the specified property has no duplicate values within a hierarchy If Include Defaults is False, then nodes with the default value are not included. If Exclude Shared is True, then shared nodes are not considered when checking uniqueness of property values.	Property, Include Defaults, Exclude Shared It is recommended that the UniqueProp validation use indexed properties.
UniquePropBranch	Node	Verifies that the specified property has unique value within a branch	Property
VersionFail	Version	Automatically fails at the version level for testing purposes	none
VersionUnique2Prop	Global Node	Verifies that specified properties have no duplicate values within a version If Include Defaults is False, then nodes with the default value are not included. If Exclude Shared is True, then shared nodes are not considered when checking uniqueness of property values.	First property, Second property, Include Defaults, Exclude Shared
VersionUniqueProp	Global Node	Verifies that the specified property has no duplicate values within a version If Include Defaults is False, then nodes with the default value are not included. If Exclude Shared is True, then shared nodes are not considered when checking uniqueness of property values.	Property, Include Defaults, Exclude Shared

Validation Levels

The validation level defines the scope of a business rule. For node validations, the level can also include the type of action that needs to be performed in order for the validation to run. The following table defines each validation level and indicates:

- Whether the validation can run in batch mode, real-time mode, or both.
- Where the validation gets assigned.
- On which object the validation operates.

Table 10-2 Validation Levels

Validation Level	Runs in Batch or Real-time	Where Assigned	Operates On
Node—Reviews node relationships and properties to ensure criteria are met. Use to determine whether a node level string property value has a valid length.	Real-time or Batch	Version, Hierarchy, or Node	Local Node
Hierarchy—Reviews properties in a hierarchy to ensure criteria are met. Can be assigned and run at the hierarchy or version levels. Use to ensure that a hierarchy has no more than 10,000 nodes.	Batch	Version or Hierarchy	Hierarchy
Version—Reviews the properties of a version. Use to ensure that a version contains no more than 100,000 nodes.	Batch	Version	Version
Global Node—Assigned at a version level. Validates every node in the version regardless of hierarchy, including orphans. Only properties defined as global are reviewed. Use to ensure that all nodes within a version have a unique property value.	Batch	Version	Global Node
Merge—Runs when an operation requiring a merge (for example, a delete or an inactivate) is performed. Assigned at the version level. Use to ensure that a leaf node is merged only into another leaf node.	Real-time	Version	Global Node
Move—A validation triggered when an attempt is made to move a node. Assigned at the hierarchy level. Use to prevent moving of cost centers within a hierarchy.	Real-time	Hierarchy	Local Node

Table 10-2 (Cont.) Validation Levels

Validation Level	Runs in Batch or Real-time	Where Assigned	Operates On
Remove—Similar to the Move level. Runs when an attempt is made to remove or delete a node from a hierarchy. Can be used to prevent specified types of nodes from being deleted. Use to prevent the deletion of cost center nodes from a hierarchy.	Real-time	Version or Hierarchy	Global Node

Creating Validations

To create a validation:

1. On the Home page, select **Administer**.
2. From **New**, select **Validation**.
3. Enter a name for the validation.

   Note

   The validation will be assigned to the Custom namespace. The Fully Qualified Name for the validation must be unique. The Label field is filled in automatically after entering the name. The validation label is a user-friendly descriptor that is displayed for validations for all features aside of application administration. Multiple validations can have the same Label as long as they are not in the same namespace.

4. Enter the message to display to the user if the validation fails.
5. Select a validation class. See Validation Classes.

   Note

   The valid levels are populated depending on the class selected.

6. For classes that can be run in real time at the node level, select a level that includes a type of action.
7. Select from the following options for the validation:
   - RealTime—Runs when a change is made
   - Batch—Runs when explicitly requested
   - Inherited—Runs for selected node and its descendants

> **Note**
>
> Depending on the validation class you select, some of these options may not be available or parameters are displayed for which you may need to edit values.

8. Define the parameters for the selected validation class.

   See Validation Classes for the parameters for each validation class. For more information on creating formulas, see Creating Formulas. For more information on creating scripts, see Creating Dynamic Scripts.

9. Click 💾.

Creating a Script Validation for Move

To create a script validation for a Move:

1. On the Home page, select **Administer**.
2. From **New**, select **Validation**.
3. From **Class**, select **Script**.

By default, validation level is Node and run mode is Batch.

4. Under **Run this Validation**, select **Real Time**.

This enables the validation to be triggered on a particular action (such as Move).

5. From **Level**, select **Move**.

> **Note**
>
> The Level option is above the Real Time option that you selected in step 4.

6. Save the validation.

Assigning Validations

After you create validations, you can assign them to versions, hierarchies, domains, and nodes. Multiple validations can be assigned at the same time.

> **Note**
>
> When assigned at a domain level, validations are inherited by all nodes that are members of that domain. When assigned at the version level, validations are inherited by all hierarchies and nodes within the version. When assigned at the hierarchy level, validations are inherited by all nodes within the hierarchy.

For information on assigning validations to domains, see Managing Domains. For information on assigning validations to versions, hierarchies, and nodes, see the *Oracle Data Relationship Management User's Guide*.

Editing Validations

To edit a validation:

1. On the Home page, select **Administer**.
2. Under **Metadata**, expand **Validations**.
3. Select a validation and click ✎.
4. Make changes to the validation.

   Note

   The Class, Level, and Mode of Operation parameters cannot be modified after a validation has been saved.

5. Click **Save**.

Deleting Validations

When you delete a validations, all validation assignments to versions, hierarchies, and nodes are also deleted.

To delete a validation:

1. From the Home page, select **Administer**.
2. Under **Metadata**, expand **Validations**.
3. Select a validation and click ✖.
4. Select **Delete this Item** to confirm the deletion.

11

Managing Formulas

Formulas enable you to define complex logic for derived properties and validations using a native formula language in Oracle Data Relationship Management. Formulas are composed of functions and string literals and must follow specific syntax rules.

For more information, see:

- Creating Properties
- Managing Validations

Working with Functions

Function names are case-insensitive and should be immediately followed by parentheses, regardless of whether parameters are required.

Function parameters must be of the expected type and number. Parameters can be nested functions or string literals. If parameters are of incorrect type, an error is reported. In the case of too few parameters, a list index out of bounds error is reported. In the case of too many parameters, additional parameters are ignored.

Special Characters

In certain functions for which parameter values contain special characters (for example: comma, space, tab), use brackets ([]). For example, `FlipList(PropValue(Custom.NodeList),[comma])` performs the FlipList function on the comma-delimited list returned from the function call `PropValue(Custom.NodeList)`.

The following functions can take `comma`, `space`, or `tab`, in brackets ([]), for the Delimiter parameter: `ArrayCount`, `ArrayIndex`, `ArrayItem`, `FlipList`, `Intersection`, `ListContains`, `PadList`, `RangeListContains`, `IsRangeListSubset`, `MinList`, `MaxList`, `AvgList`, `SumList`, `SortList`, `ListDistinct`, `ListNodePropValues`, and `ListNodesWith`.

The `ReplaceStr` function, which requires parameters for the old and new patterns, can take `comma`, `space`, `tab`, `crlf`, `cr`, `lf`, `openparen`, or `closeparen`, in brackets ([]), in addition to normal text strings.

Note

Parameter values that contain literal commas will result in this syntax error, "Invalid number of parameters". A comma-delimited list passed in as the result of a function call is a valid use and will be handled as expected. For example:

Invalid syntax: `FlipList(a,b,c,[comma])`

Valid syntax: `FlipList(PropValue(Custom.NodeList),[comma])` where `Custom.NodeList` value = a,b,c

Literals

Any value that is not a valid function name followed by parentheses is considered a literal. A literal can be a string, integer, floating-point, or boolean literal. In a string literal, spaces are treated as a character. Therefore, do not use extra spaces in formulas unless they are necessary to derive the appropriate result. You can use the Remove Spaces option to strip spaces from the formula before saving.

Format String Parameter

Format strings passed to the string formatting routines contain two types of objects — literal characters and format specifiers. Literal characters are copied verbatim to the resulting string. Format specifiers get a property value from the specified property and apply formatting to it. Only one specifier can exist in the format string.

Format specifiers use the following form:

```
"%"["-"][width]["."prec]type
```

Table 11-1 Format String Characters

Character	Description
%	Indicates start of a format specifier
["—"]	Left justification indicator (optional) Left justifies the result by adding blanks after the value. The default is to right-justify the result by adding blanks before the value.
[width]	Width specifier (optional) Sets the minimum field width for a conversion. If the resulting string is shorter than the minimum field width, it is padded with blanks to increase the field width.
["." prec]	Precision specifier (optional)
type	Conversion type character Conversion characters may be specified in uppercase or lowercase. For all floating-point formats, the actual characters used as decimal and thousand separators are obtained from the DecimalSeparator and ThousandSeparator global variables or their TFormatSettings equivalent. Valid values for type are listed in the following table.

Table 11-2 Format String Type Values

Type Value	Description
d	Decimal The property value must be an integer. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.
u	Unsigned decimal Similar to d but no sign is output.
e	Scientific The property value must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd". The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string; a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.
f	Fixed The property value must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative. The number of digits after the decimal point is given by the precision specifier in the format string; a default of two decimal digits is assumed if no precision specifier is present.
g	General The property value must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string; a default precision of 15 is assumed if no precision specifier is present. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format.

Table 11-2 (Cont.) Format String Type Values

Type Value	Description
n	Number The property value must be a floating-point value. The value is converted to a string of the form "-d,ddd,ddd.ddd...". The "n" format corresponds to the "f" format, except that the resulting string contains thousand separators.
m	Money The property value must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, DecimalSeparator, and CurrencyDecimals global variables or their equivalent in a TFormatSettings data structure. If the format string contains a precision specifier, it overrides the value given by the CurrencyDecimals global variable or its TFormatSettings equivalent.
s	String The property value must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the property value is a string that is longer than this maximum, the string is truncated.
x	Hexadecimal The property value must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

Date-Time Format Strings

Date-time format strings specify the formatting of date-time values (such as TDateTime) when they are converted to strings. Date-time format strings are composed from specifiers that represent values to be inserted into the formatted string. Some specifiers (such as "d") format numbers or strings. Other specifiers (such as "/") refer to locale-specific strings from global variables. The case of the specifiers is ignored in formats, except for the "am/pm" and "a/p" specifiers.

Specifier	Display
c	Date followed by time **Note:** The time is not displayed if the date-time value indicates midnight precisely.
d	Day as a number without a leading zero (1–31)

Specifier	Display
dd	Day as a number with a leading zero (01–31)
ddd	Day as an abbreviation (Sun-Sat)
dddd	Day as a full name (Sunday-Saturday)
ddddd	Short format of date
dddddd	Long format of date
e	Year in the current period/era as a number without a leading zero (Japanese, Korean, and Taiwanese locales only)
ee	Year in the current period/era as a number with a leading zero (Japanese, Korean, and Taiwanese locales only)
g	Period/era as an abbreviation (Japanese and Taiwanese locales only)
gg	Period/era as a full name (Japanese and Taiwanese locales only)
m	Month as a number without a leading zero (1–12) **Caution:** If the "m" specifier immediately follows an "h" or "hh" specifier, the minute rather than the month is displayed.
mm	Month as a number with a leading zero (01–12) **Caution:** If the "mm" specifier immediately follows an "h" or "hh" specifier, the minute rather than the month is displayed.
mmm	Month as an abbreviation (Jan-Dec)
mmmm	Month as a full name (January-December)
yy	Year as a two-digit number (00–99)
yyyy	Year as a four-digit number (0000–9999)
h	Hour without a leading zero (0–23)
hh	Hour with a leading zero (00–23)
n	Minute without a leading zero (0–59)
nn	Minute with a leading zero (00–59)
s	Second without a leading zero (0–59)
ss	Second with a leading zero (00–59)
z	Millisecond without a leading zero (0–999)
zzz	Millisecond with a leading zero (000–999)
t	Time using the format given by the ShortTimeFormat global variable
tt	Time using the format given by the LongTimeFormat global variable
am/pm	Uses the 12-hour clock for the preceding "h" or "hh" specifier, and displays "am" for any hour before noon, and "pm" for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.

Specifier	Display
a/p	Uses the 12-hour clock for the preceding "h" or "hh" specifier, and displays "a" for any hour before noon, and "p" for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding "h" or "hh" specifier
/	Date separator character given by the regional settings
:	Time separator character given by the regional settings
'xx'/"xx"	Characters enclosed in single or double quotation marks are displayed as-is and do not affect formatting.

Formula Evaluation

You can test formulas when you create or modify a property definition or validation. The formula is evaluated using the supplied property values to calculate the result of the formula. This process may find logic or implementation errors in the formula that a simple syntax validation may miss. The formula result and any formula error or status message is displayed.

Formulas are evaluated left to right, with evaluation of functions and string literals performed as they are encountered. By this method, nested functions are evaluated before additional parameters that are displayed to the right of the nested function. Functions can be nested explicitly in the formula or they can be implicitly nested by retrieving the value of another formula property. Circular references (property formulas that refer to the property itself, either explicitly or implicitly) should be avoided in most cases. Oracle Data Relationship Management detects and prevents harmful circular references, but they should not be used unless they are necessary and well understood.

Formula Syntax Checks

Formula syntax is verified for the following before a formula is saved:

- Function names are correct.
- Property names are correct.
- An equal number of open and close parentheses are present.
- The actual number of parameters is at least the expected number of parameters for each function

Functions such as `Concat` can take any number of parameters. The parameter count validation verifies that the actual number of parameters is equal to or greater than the expected number of parameters. Thus too many parameters do not generate an error, but too few parameters do.

The syntax validation does not evaluate the formula, therefore errors may occur if invalid constants are entered. For example: `IntToStr(ABC,3)` passes the syntax validation, but generates an error in the Oracle Data Relationship Management application. You must evaluate each formula to avoid this type of error prior to saving.

Property Names in the Syntax Check

In order to accurately perform a syntax validation on property names, functions that require property names are partially evaluated for those rare cases in which a property name is not a literal but is the result of a function.

Consider these examples:

- The formula `PropValue(Concat(Core.Abbrev))` is valid, but the `Concat` function has to be evaluated (not just validated for syntax) to verify the property name.
- The formula `PropValue(If(NodeIsLeaf(),Core.Abbrev,Custom.Label))` is valid, but the If function has to be evaluated to verify the property name.

If the property name in question comprises only part of the formula, only the parts needed to determine property names are evaluated. For example, in the formula `Add(PropValue(Concat(Core.,I,D)),If(NodeIsLeaf(),0,1)`, the only part of the formula evaluated for the syntax validation is the `Concat` function and its parameters.

The fact that these formula parts are evaluated becomes significant in cases such as `PropValue(PropValue(NodeType))`. For this formula, the syntax validation fails unless a value is supplied for the *Custom.NodeType* property.

Considerations for Using Formulas

Data Type Conversion

Some functions require that data values be of a specific data type to be properly evaluated. For example, functions that perform mathematical calculations require that input arguments are integer or floating point values, whereas string manipulation functions require that string values be provided as input. In some cases, data values must be converted from one data type to another to be successfully derived. Oracle Data Relationship Management provides a set of functions to handle data type conversions within formulas.

Property Level Restrictions

Generally, property definitions created to manage data at a lower level of granularity can reference other properties that manage data at a higher level of granularity.

- Local Node—May refer to other local node, global node, hierarchy, or version properties
- Global Node—May refer to other global node or version properties
- Hierarchy—May refer to other hierarchy or version properties (Lookup only)
- Version—May refer to other version properties (Lookup only)

Referencing Properties from Other Nodes

It is common for a derived property or validation to evaluate or retrieve a property value from a different node than the current node for which the formula is being calculated. Data Relationship Management provides several functions that enable you to access property values from nodes within the same version.

- NodePropValue
- ParentPropValue
- HierNodePropValue

- AncestorProp
- DualAncestorProp
- AscNodeProp
- ReplacePropValue
- ListPropValues
- ListNodePropValues

Referencing Local Node Properties from Global Node Properties

Global node properties do not require a hierarchy context to return a value, whereas local node properties do require a hierarchy to be specified. Derived properties or validations that are calculated for a global node cannot reference local node property values using the standard PropValue or NodePropValue functions. Global node properties may reference local node property values using the HierNodePropValue function whereby a particular hierarchy must be specified to retrieve the value of the property for a specific local node in the hierarchy.

Nesting Functions

Combining functions into the same formula is referred to as nesting functions. The output of one function is used as an input argument for another function in the formula. When evaluating nested functions, Data Relationship Management executes the innermost function first and then works its way outward. Functions may be nested explicitly within the same formula or nested implicitly by using one formula that refers to a property that uses a different formula.

Using Properties as Variables for Other Properties

Data Relationship Management enables you to use a combination of nested functions, references to other properties or nodes, and literal values, which may result in lengthy or complex formulas. You can use separate property definitions to modularize formula logic and simplify the formula syntax required to achieve the same results. This approach may significantly improve the ease of maintenance for these formulas.

In addition, formulas may evaluate the same data or perform the same calculation multiple times within the same property definition or across multiple property definitions for a given node. When this logic is embedded in a much larger formula or implemented within property definitions, these checks and calculations are performed multiple times, which may affect the performance for operations that require the properties to be calculated. You can minimize redundant processing by isolating the duplicate formula logic within a separate property definition.

Using Recursion to Traverse Hierarchy Relationships

Business rules for nodes at lower levels of a hierarchy may require the evaluation of property values from ancestor nodes above them. One way to allow these property values to be referenced by lower-level nodes is to enable inheritance on the property definition that manages the values that must be referenced. However, in many cases, using an inheritance for a property definition is not appropriate.

You can use specific hierarchical formula functions with a self-reference to the current property definition to recurse up a branch of a hierarchy to retrieve or evaluate property values for ancestor nodes.

ParentPropValue—Use this function to recurse up a branch of ancestors in the current hierarchy. For example: `If(Equals(Integer,PropValue(Core.Level),1),Label Only,ParentPropValue(Essbase.DataStorage))`

HierNodePropValue—Use this function to recurse up a branch of ancestors in another hierarchy. For example:
`If(Equals(Boolean,PropValue(Custom.PlanPoint),True),Abbrev(),HierNodePropValue(Geography,HierNodePropValue(Geography,Abbrev(),Core.Parent),Custom.PlanMember))`

Creating Formulas

Formulas are created in the formula editor which is available on the Parameters tab for creating or editing derived property definitions and validations.

To create a formula:

1. You can enter a text formula or insert functions and properties in the following ways on the **Parameters** tab:
   - To insert a function, place your cursor in the formula and click **Insert Function**. A list of functions is displayed. Expand a function to view its input parameters. Enter the parameter values and click **OK**.
   - To insert a property, place your cursor in the formula and click **Insert Property**. A list of properties is displayed. Select a property and click **OK**.
2. Make selections from the following options:
   - **Remove Spaces**—Selected by default. If selected, all spaces in the formula are removed when the formula is evaluated and when the property is saved. To preserve spaces to be evaluated as literal values in the formula, disable this option.
   - To evaluate the formula, select an option:
     - **Evaluate with Selected Node**—Click ⋯ and select a node. The node's current property values are used in the formula. Click **Evaluate**. The result is displayed at the bottom of the formula designer.
     - **Evaluate with Scratch Pad**—Enter property values manually. Values can also be copied from a node and then modified for the evaluation. In the Copy From Node, click ⋯ and select a node to display its property values in the grid. Use the filter row below the column headings to filter the list of properties. Use the Edit buttons in the Action column to modify property values for evaluation with the formula. Click Evaluate. The Evaluation Result is displayed at the bottom of the formula designer.
3. To test the formula, click **Evaluate**.

Function Definitions

Following is an alphabetical listing of available functions used with derived formula property definitions.

Abbrev

Description

Returns the name (Abbrev) of the current node.

Syntax

```
Abbrev(): String
```

Example

```
Abbrev()
```

Return value is the name of the node.

Add

Description

Adds two specified integer values and returns the result.

Syntax

```
Add(Int1,Int2:Integer):Integer
```

Example

```
Add(1,4)
```

Return value is 5.

AddedBy

Description

Returns the value of the Added By change tracking property.

Syntax

```
AddedBy():String
```

Example

```
AddedBy()
```

Returns the name of the user who added the current node to the version.

AddedOn

Description

Returns the value of the Added On change tracking property as a date/time.

Syntax

```
AddedOn():Date/Time
```

Example

```
AddedOn()
```

Returns the date and time at which the current node was added to the version.

AddFloat

Description

Adds two specified float values and returns the result.

Syntax

```
AddFloat(Float1,Float2:Float):Float
```

Example

```
AddFloat(2.14,3.75)
```

The return value is 5.89.

AncestorProp

Description

Returns a property value of the first ancestor where a property equals a specified value.

This function is local in scope and will not function properly if used in a global context.

> **Note**
>
> If the current node is valid for the criteria, then it will be returned.

Syntax

```
AncestorProp(Operator:String,Property:String,Value:String,FromTop:Boolean,Retu
rnProp:String)
```

Operator is the operator to use when comparing the property with the value. Valid values: =, <, >, >=, and <=.

Property is the name of the property to use.

Value is the value to compare.

FromTop specifies whether to search from the top node of the hierarchy. If False, the search is performed starting from the current node.

ReturnProp is the name of the property to return.

And

Description

Returns True if all specified Boolean expressions evaluate to True.

Syntax

```
And(Expression1,Expression2,...ExpressionN:Boolean):Boolean
```

Example

```
And(1,T,True)
```

Return value is True.

ArrayCount

Description

Returns the number of items in a specified list (array).

Syntax

```
ArrayCount(List:String,Delimiter:String):Integer
```

List specifies the list of strings in which to search.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

Note

You must use the name of the delimiter (not the character) and include brackets around the name.

Example

```
ArrayCount(Diet Cola;Root Beer;Cola,[comma])
```

Return value is 3.

ArrayIndex

Description

Returns the position of the first occurrence of the specified item within the list (array). Returns zero (0) if the item is not found.

Syntax

```
ArrayIndex(Item:String,List:String,Delimiter:String):Integer
```

Item specifies the string value to test.

List specifies the list of strings in which to search.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

Note

You must use the name of the delimiter (not the character) and include brackets around the name.

Example

```
ArrayIndex(Cola,Diet Cola;Root Beer;Cola,[comma])
```

Return value is 3.

ArrayItem

Description

Returns the item in the list (array) at the specified index position.

Syntax

```
ArrayItem(List:String,Delimiter:String,Index:Integer):String
```

List specifies the list of strings in which to search.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

Note

You must use the name of the delimiter (not the character) and include brackets around the name.

Index is the position of the string in the list. A negative value indicates the last item in the list.

Example

```
ArrayItem(Diet Cola;Root Beer;Cola,;,3)
```

Return value is Cola.

AscNodeProp

Description

Returns a property value of the associated node referenced by the specified property.

Syntax

```
AscNodeProp(LookUpProperty,ReturnProperty)
```

LookupProperty is the name of the property that points to the node. Property must be datatype Node or AscNode.

ReturnProperty is the name of the property of the associated node to return. Property must be global.

AvgList

Description

Returns the average of the items in a list, ignoring blank items. Returns a blank string if the list contains an item not of the specified item type.

Syntax

```
AvgList(InputList:String,Delimiter:String,ItemType:String):String
```

InputList specifies the list to use.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

ItemType indicates the expected item data type for list members. Valid values: integer, float, and datetime. The default value is float.

Example

```
AvgList(1;2;3,[comma],Integer)
```

Return value is 2.

BoolToStr

Description

Returns a Boolean value converted to True or False. Returns False if the input does not represent a Boolean value.

Syntax

```
BoolToStr(Expression:Boolean):String
```

Example

```
BoolToStr(1)
```

Return value is True.

Changed

Description

Returns the value of the Node Changed change tracking property as a Boolean.

Syntax

```
Changed()
```

ChangedBy

Description

Returns the name of the user who last updated the current node in the version.

Syntax

```
ChangedBy():String
```

Example

```
ChangedBy()
```

ChangedOn

Description

Returns the value of the Changed On change tracking property.

Syntax

```
ChangedOn():Date/Time
```

Example

```
ChangedOn()
```

Returns the date and time at which the current node was last updated in the version.

Concat

Description

Concatenates two or more specified strings into one and returns the result.

Syntax

```
Concat(Item1,Item2,... ItemN:String):String
```

Example

```
Concat(Abbrev,-,Descr())
```

If current node name is 100 and current node description is Colas, then return value is 100–Colas.

ConcatWithDelimiter

Description

Concatenates two or more strings into one delimited list and returns the result.

Syntax

```
ConcatWithDelimiter(Delimiter:String,SkipBlanks:Boolean,Items:String)
```

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

SkipBlanks indicates whether to skip blank values in the list of strings. Valid values: 1, 0, T, F, t, f.

Items specifies the list of strings to concatenate.

Example

```
ConcatWithDelimiter([comma],1,Item1,Item2,Item3,Item4)
```

Return value is Item1; Item2; Item3; Item4.

Decode

Description

Returns the input string with all instances of [openparen], [closeparen], [comma], [tab], [space], [crlf], [cr], and [lf] replaced by the appropriate character.

Note

This function is for upgrading property definition names that use special characters. These special characters can cause parsing issues with derived property formulas. This function is used primarily to convert existing properties using deprecated deriver classes to the Formula deriver class.

Syntax

```
Decode(CodedString:String):String
```

CodedString is the string value on which to perform the function.

DefaultProp

Description

Returns the default value for the property.

Syntax

```
DefaultProp(Property:String)
```

Property is the name of the property to use.

Descr

Description

Returns the description of the current node.

Syntax

```
Descr():String
```

Example

If current node description is Colas, then return value is Colas.

Divide

Description

Divides two specified integer values and returns the result.

Syntax

```
Divide(Int1,Int2:Integer):Integer
```

Example

```
Divide(200,10)
```

Return value is 20.

DivideFloat

Description

Divides two floating-point numbers (float) and returns the result.

Syntax

```
Divide(Float1,Float2:Float):Float
```

Example

```
DivideFloat(2.535,1.5)
```

The return value is 1.69.

DualAncestorProp

Description

Returns a property value of the first ancestor where two properties equal the specified values.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
DualAncestorProp(Operator1:String,Property1:String,Value1:String,Operator2:Str
ing,Property2:String,Value2:String,FromTop:Boolean,ReturnProp:String):String
```

Operator1 is the operator to use when comparing the first property and value. Valid values: =, <, >, >=, and <=.

Property1 is the name of the first property to check.

Value1 is the first value to compare.

Operator2 is the operator to use when comparing the second property and value. Valid values: =, <, >, >=, and <=.

Property2 is the name of the second property to check.

Value2 is the second value to compare.

FromTop specifies whether to search from the top node of the hierarchy. If False, the search is performed starting from the current node.

ReturnProp is the name of the property of the ancestor to return.

Equals

Description

Returns True if two specified values are equal. This function is case-sensitive.

Syntax

```
Equals(ParamType:String,Param1:String,Param2:String):Boolean
```

ParamType is the data type to use for comparing values. Valid values: string, integer, float, date. The default value is integer.

Param1 is the first value to compare.

Param2 is the second value to compare.

Example

```
Equals(integer,01,1)
```

Return value is True.

FlipList

Description

Returns a string representing the reverse of the specified list.

Syntax

```
FlipList(List,Delimiter:String):String
```

List specifies the list of strings to flip.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

Note

You must use the name of the delimiter (not the character) and include brackets around the name.

Example

```
FlipList(DietCola;Orange Soda;Root Beer;Lemonade,[comma])
```

Return value is Lemonade,Root Beer,Orange Soda,Diet Cola.

FloatToStr

Description

Returns a float value converted to a string. Returns zero (0) if the input value does not represent a float.

Syntax

```
FloatToStr(Float1:Float):String
```

Example

```
FloatToStr(1.001)
```

Return value is 1.001.

Format

Description

Formats the value using a specified format string parameter type identifier and parameter value of the specified type. This function is limited to one value parameter.

Syntax

```
Format(Format:String,ParamType:String, ValueToFormat:String):String
```

Format is the format to apply.

ParamType is the data type to use for comparing values. Valid values: string, integer, float, date. The default value is integer.

ValueToFormat is the value on which to perform the function.

Example

```
Format('%8.2f',Float,123.456)
```

Return value is 123.46.

FormattedDate

Description

Returns the value of a date property formatted using the specified format string.

Syntax

```
FormattedDate(PropertyName:String,FormatString:String): String
```

PropertyName is the name of the property to use.

FormatString specifies the date format to apply.

GreaterThan

Description

Compares two values and returns True if the first value is greater than the second value.

Syntax

```
GreaterThan(Value1:Integer,Value2:Integer,ParamType:String):Boolean
```

Value1 is the first value to compare.

Value2 is the second value to compare.

ParamType is the data type to use for comparing values. Valid values: string, integer, float, date. The default value is integer.

Example

```
GreaterThan(1,2)
```

The return value is False.

GreaterThanOrEqual

Description

Compares two values and returns True if the first value is greater than or equal to the second value.

Syntax

```
GreaterThanOrEqual(Value1:Integer,Value2:Integer,ParamType:String):Boolean
```

Value1 is the first value to compare.

Value2 is the second value to compare.

ParamType is the data type to use for comparing values. Valid values: string, integer, float, date. The default value is integer.

Example

```
GreaterThanOrEqual(2,2)
```

The return value is True.

HasCharacters

Description

Returns True if the specified Input contains characters from the Character Classes, Special Characters, or Characters listed in CharList.

Syntax

```
HasCharacters(Input:String,CharList:String):Boolean
```

Input is the string value to test.

CharList is a list of characters to test, including optional special values. Special character values are enclosed in brackets and separated by a comma. Valid values: [alpha], [numeric], [whitespace], [punctuation], [uppercase], [lowercase], [comma], [space], [tab], [crlf], [cr], [lf], [openparen], and [closeparen].

HasChildWith

Description

Returns True if the specified expression is True for any child of the current node.

Syntax

```
HasChildWith(Expression:Boolean):Boolean
```

Example

```
HasChildWith(GreaterThan(ID(),200))
```

If the current node has any children with an ID greater than 200, then return value is True.

HasParentNode

Description

Returns True if the current local node has a parent node.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
HasParentNode():Boolean
```

Example

```
HasParentNode()
```

If the node is a child of the top node of a hierarchy or any descendant node, then the return value is True.

HasSiblingWith

Description

Returns True if the specified expression is True for any sibling of the current node.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
HasSiblingWith(Expression:Boolean):Boolean
```

Example

```
HasSiblingWith(PropValue(Leaf))
```

If any of the children are leaves, then the return value is True.

HierNodePropValue

Description

Returns the value of the specified property of the specified node in the specified hierarchy.

Syntax

```
HierNodePropValue(HierAbbrev:String,NodeAbbrev:String,PropAbbrev:String):Strin
g
```

HierAbbrev is the name of the hierarchy to use.

NodeAbbrev is the name of the node to use.

PropAbbrev is the name of the property to use.

Example

```
HierNodePropValue(Assets,1000,Description)
```

If the description for node 1000 in the Assets hierarchy is "Banking", then the return value is Banking.

ID

Description

Returns the ID of the current node.

Syntax

```
ID():Integer
```

Example

```
ID()
```

If the current node ID is 2000, then the return value is 2000.

If

Description

Returns the value of the TrueResult parameter if the specified expression evaluates to True. Otherwise, it returns the value of the FalseResult parameter.

Syntax

```
If(Expression:Boolean, TrueResult:String,FalseResult:String):String
```

Expression is a Boolean expression to evaluate.

TrueResult is the string value to return if the condition is True.

FalseResult is the string value to return if the condition is False.

Example

```
If(Equals(String,Descr(),),Abbrev(),Concat(Abbrev,-,Descr()))
```

If the node name is Colas and the current node description is blank, then the return value is Colas.

If the node name is 100 and the current node descriptions is Colas, then the return value is 100–Colas.

InheritedPropOrigin

Description

Returns the name of the node from where an inherited property value originates. If the specified property is global, then the origin hierarchy is also returned. Returns False if the specified property is not inheriting, or if the node or property is not found.

This function can be local in scope if a local property is passed in the parameters.

Syntax

```
InheritedPropOrigin(PropAbbrev:String,Node:String):String
```

Example

```
InheritedPropOrigin(Custom.AccountType,Abbrev())
```

PropAbbrev is the name of the property to use.

Node is the name of the node to use.

InRange

Description

Returns True if the specified value falls within a specified range of values. If the input parameter is a string, the Min and Max parameters specify a string length range to check. For other types, Min and Max specify a numeric or date value range to check.

> **Note**
>
> If MinExclusive/MaxExclusive is True, then values equal to the Min/Max are included in the range, otherwise they are excluded.

Syntax

```
InRange(DataType:String,Input:String,Min:String,Max:String,MinExclusive:String
,MaxExclusive:String):Boolean
```

DataType is the data type to use. Valid values: string, integer, float, and datetime.

Input is the string value to test.

Min is the minimum value for length or range check.

Max is the maximum value for length or range check.

MinExclusive specifies whether to exclude the Min value from the range to check.

MaxExclusive specifies whether to exclude the Max value from the range to check.

Example

```
InRange(Integer,5,1,10,False,False)
```

Return value is True.

InternalPrefix

Description

Returns the non-numeric prefix from the name of the current node.

Syntax

```
InternalPrefix()
```

Intersection

Description

Returns the set of items common to both specified lists of values. The ordering of the results is based on how the items appear in the first list specified.

Syntax

```
Intersection(List1:String,List2:String,Delimiter:String):String
```

List1 specifies a list of strings in which to search.

List2 specifies a list of strings in which to search.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

Example

```
Intersection(A;B;C;D;E,C;E;F;A,[comma])
```

The return value is A,C,E.

IntToStr

Description

Returns the specified integer value converted to a string data type. Returns zero (0) if the input value does not represent an integer.

Syntax

```
IntToStr(Int1:Integer):String
```

Example

```
IntToStr(12345)
```

The return value is 12345.

InvertedLevel

Description

Returns the maximum depth of descendants below the current node.

Syntax

```
InvertedLevel()
```

IsAlpha

Description

Returns True if the specified string contains only alphabetical characters (case-insensitive).

Syntax

```
IsAlpha(String:String):Boolean
```

Example

```
IsAlpha(A23D)
```

The return value is False.

IsAlphaNumeric

Description

Returns True if the specified string contains only alphabetical or numeric characters (not case-sensitive).

Syntax

```
IsAlphaNumeric(String:String,AllowBlanks:Boolean):Boolean
```

String is the string value to test.

AllowBlanks specifies whether a blank string should be treated as numeric. Default is False.

Example

```
IsAlphaNumeric(ABC123,True)
```

Returns True.

IsBlank

Description

Returns True if the specified input value is an empty string (zero length).

Syntax

```
IsBlank(Input:String):Boolean
```

Example

```
IsBlank(Descr())
```

Returns True if the node description is blank.

IsBottomNode

Description

Returns True if the specified node has no child nodes. Returns False if the node is not found.

Syntax

```
IsBottomNode(Node:String):Boolean
```

Node is the name of the node to use.

Example

```
IsBottomNode(Abbrev)
```

Returns True if the node does not have children.

IsDataType

Description

Returns True if the input value matches the specified data type.

Syntax

```
IsDataType(DataType:String,Input:String):Boolean
```

DataType is the data type to use. Valid values: boolean, string, integer, float, and datetime.

Input is the string value to test.

Example

```
IsDataType(123,Integer)
```

Returns True.

IsDefinedPropVal

Description

Returns True if the specified property for the specified node has a defined (overridden) value. Returns False if the node or property is not found.

This function can be local in scope if a local property is passed in the parameters.

Syntax

```
IsDefinedPropVal(PropAbbrev:String,Node:String):Boolean
```

PropAbbrev is the name of the property to use.

Node is the name of the node to use.

Example

```
IsDefinedPropVal(Custom.AccountType,Abbrev())
```

Returns True if the Account Type property has a defined (overridden) value.

IsNodeAbove

Description

Returns True if the first node is an ancestor of the second node in the current hierarchy. Returns False if Node1 or Node2 is not found.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
IsNodeAbove(Node1:String,Node2:String):Boolean
```

Node1 is the name of the first node to use.

Node2 is the name of the second node to use.

Example

```
IsNodeAbove(Parent,Child)
```

Returns True if node parent is an ancestor of the child node.

IsNodeBelow

Description

Returns True if the first node is a descendant of the second node in the current hierarchy. Returns False if Node1 or Node2 is not found.

Syntax

```
IsNodeBelow(Node1:String,Node2:String):Boolean
```

Node1 is the name of the first node to use.

Node2 is the name of the second node to use.

Example

```
IsNodeBelow(Child,Parent)
```

Returns True if node child is descendant of the parent node.

IsNumeric

Description

Returns True if the specified value contains only numeric characters (0-9).

Syntax

```
IsNumeric(String: String,AllowBlanksAsNumeric:Boolean):Boolean
```

String is the string value to test.

AllowBlanksAsNumeric specifies whether to allow a blank value to be considered a string. The default value is False.

Example

```
IsNumeric(12345)
```

The return value is True.

IsRangeListSubset

Description

Returns True if the specified value is a subset of the specified range list.

Syntax

```
IsRangeListSubset(RangeList:Range List,SubsetRangeList:Range
List,Delimiter:String):Boolean
```

RangeList is a list of integer ranges to search, separated by the specified delimiter.

SubsetRangeList is a subset list of integer ranges to search, separated by the specified delimiter.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

Note

You must use the name of the delimiter (not the character) and include brackets around the name.

Length

Description

Returns the number of characters in the specified string value.

Syntax

```
Length(String:String):Integer
```

Example

```
Length(Desc())
```

If the description for the current node is Colas, then the return value is 5.

LessThan

Description

Compares two values and returns True if the first value is less than the second value.

Syntax

```
LessThan(Value1:Integer,Value2:Integer,ParamType:String):Boolean
```

Value1 is the first value to compare.

Value2 is the second value to compare.

ParamType is the data type to use for comparing values. Valid values: string, integer, float, date. The default value is integer.

Example

```
LessThan(1,2)
```

The return value is True.

LessThanOrEqual

Description

Compares two values and returns True if the first value is less than or equal to the second value.

Syntax

```
LessThanOrEqual(Value1:Integer,Value2:Integer,ParamType:String):Boolean
```

Value1 is the first value to compare.

Value2 is the second value to commpare.

ParamType is the data type to use for comparing values. Valid values: string, integer, float, date. The default value is integer.

Example

```
LessThanOrEqual(3,3)
```

The return value is True.

ListAncestors

Description

Returns a comma-delimited list of the names of the current node's ancestors, starting from the top node. Returns a blank string if the current node is not a local node.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
ListAncestors(SortOrder:String):String
```

SortOrder specifies the sort order for the return list of nodes. Supported sort order values:

- [hier]—Default value for local context. The list of nodes is returned in the standard hierarchy sort order for the current hierarchy.
- [alpha]—The list of nodes is returned sorted by node name.
- [nodeid]—Limited use for legacy compatibility. The list of nodes is returned sorted numerically on the node ID of each node in the return list.

> **Note**
>
> You must use brackets around the SortOrder parameter.

Example

```
ListAncestors([alpha])
```

If A, B, C, and D are children of Z, Z is a child of Y, and the current node is D, then the return value is Z,Y.

ListChildren

Description

Returns a comma-delimited list of children for the current node.

Syntax

```
ListChildren(SortOrder:String):String
```

SortOrder specifies the sort order for the return list of nodes. Supported sort order values:

- [hier]—Default value for local context. The list of nodes is returned in the standard hierarchy sort order for the current hierarchy.
- [alpha]—The list of nodes is returned sorted by node name.
- [nodeid]—Limited use for legacy compatibility. The list of nodes is returned sorted numerically on the node ID of each node in the return list.

> **Note**
>
> You must use brackets around the SortOrder parameter.

Example

```
ListChildren([alpha])
```

If A, B, C, and D are children of Z and the current node is Z, then the return value is A, B, C, D.

ListContains

Description

Returns True if the specified list contains the specified value.

Syntax

```
ListContains(List:String,Item:String,Delimiter: String):Boolean
```

List specifies the list of strings in which to search.

Item specifies the string value on which to perform the function.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

Example

```
ListContains(PropValue(NodeList),Colas,[comma])
```

The return value is True.

ListDescendants

Description

Returns a comma-delimited list of descendants for the current node.

Syntax

```
ListDescendants(SortOrder:String):String
```

SortOrder specifies the sort order for the return list of nodes. Supported sort order values:

- [hier]—Default value for local context. The list of nodes is returned in the standard hierarchy sort order for the current hierarchy.
- [alpha]—The list of nodes is returned sorted by node name.
- [nodeid]—Limited use for legacy compatibility. The list of nodes is returned sorted numerically on the node ID of each node in the return list.

> **Note**
>
> You must use brackets around the SortOrder parameter.

Example

```
ListDescendants([hier])
```

If A, B, C, and D are children of Z, Z is a child of Y, and the current node is Y, then the return value is Z, A, B, C, D.

ListDistinct

Description

Returns a distinct list of items from a specified list, with duplicates removed.

Syntax

```
ListDistinct(InputList:String,Delimiter:String):String
```

InputList specifies the list to use.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

Example

```
ListDistinct(A;B:C;A;D,[comma])
```

The return value is A,B,C,D.

ListNodePropValues

Description

Returns a list of property values for the specified property for a specified list of nodes. Returns a blank string in the list, for any node that cannot be found.

This function can be local in scope if a local property is passed in the parameters.

Syntax

```
ListNodePropValues(NodeList:String,Delimiter:String,PropAbbrev:String):String
```

NodeList is a comma-delimited list of node names.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

Note

You must use the name of the delimiter (not the character) and include brackets around the name.

PropAbbrev is the name of the property to use.

Example

```
ListNodePropValues(100;200;300,[comma],Core.Leaf)
```

Returns True,True,True if nodes 100, 200, and 300 are leaf nodes.

ListNodesWith

Description

Returns a list of nodes from the specified node list where the specified expression evaluates to True.

Syntax

```
ListNodesWith(NodeList:String,Delimiter:String,Expression:String):String
```

NodeList is a comma-delimited list of node names.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

Note

You must use the name of the delimiter (not the character) and include brackets around the name.

Expression is a Boolean expression to evaluate.

Example

```
ListNodesWith(100;200;300,[comma],NodeIsLeaf())
```

Returns True,True,True if nodes 100, 200, and 300 are leaf nodes.

ListRelatedNodesWith

Description

Returns a list of nodes related to the current node where the specified expression evaluates to True.

This function is local in scope if the relationship parameter is Ancestors or Siblings.

Syntax

```
ListRelatedNodesWith(Relation:String,Expression:String,SortOrder:String,Max:In
teger):String
```

Relation can be:

- Ancestors—Local properties can be referenced in the specified expression
- Siblings—Local properties can be referenced in the specified expression
- Children—Local and global properties can be referenced in the specified expression
- Descendants—Local and global properties can be referenced in the specified expression

Expression is a Boolean expression to evaluate.

SortOrder specifies the sort order for the return list of nodes. Supported sort order values:

- [hier]—Default value for local context. The list of nodes is returned in the standard hierarchy sort order for the current hierarchy.
- [alpha]—The list of nodes is returned sorted by node name.
- [nodeid]—Limited use for legacy compatibility. The list of nodes is returned sorted numerically on the node ID of each node in the return list.

> **Note**
>
> You must use brackets around the SortOrder parameter.

Max is an integer value indicating the maximum number of nodes to return. Zero or no value indicates no limit, and all nodes are returned.

Example

```
ListRelatedNodesWith(children,True,[alpha],1000)
```

Returns 100,200,300 if the nodes are children of the current node.

ListSiblings

Description

Returns a comma-delimited list of siblings (peers) of the current node.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
ListSiblings(SortOrder:String):String
```

SortOrder specifies the sort order for the return list of nodes. Supported sort-order values:

- [hier]—Default value for local context. The list of nodes is returned in the standard hierarchy sort order for the current hierarchy.
- [alpha]—The list of nodes is returned sorted by node name.
- [nodeid]—Limited use for legacy compatibility. The list of nodes is returned sorted numerically on the node ID of each node in the return list.

Example

```
ListSiblings([alpha])
```

If A, B, C, and D are children of Z and the current node is B, then the return value is A, C, D.

LowerCase

Description

Returns the specified string value converted to lower case.

Syntax

```
LowerCase(String:String):String
```

Example

```
LowerCase(HOBBES)
```

The return value is hobbes.

LTrim

Description

Returns the specified value with all spaces trimmed from the beginning of the string.

Syntax

```
LTrim(String: String): String
```

Example

```
LTrim("   101203")
```

The return value is 101203.

MaxList

Description

Returns the maximum item from the specified list, ignoring blank items. Returns a blank string if the list contains an item not of the specified type.

Syntax

```
MaxList(InputList: String,Delimiter: String,ItemType: String)
```

InputList specifies the list to use.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

ItemType indicates the expected item data type for list members. Valid values: integer, float, and datetime. The default value is float.

Example

```
MaxList(1;2;3,[comma],Integer)
```

Return value is 3.

MinList

Description

Returns the minimum item from the specified list, ignoring blank items. Returns a blank string if the list contains an item not of the specified type.

Syntax

```
MinList(InputList:String,Delimiter:String,ItemType:String)
```

InputList specifies the list to use.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]

- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

ItemType indicates the expected item data type for list members. Valid values: integer, float, and datetime. The default value is float.

Example

```
MinList(1;2;3,[comma],Integer)
```

Return value is 1.

Modulus

Description

Returns the modulus (remainder) of the division of two specified integers.

Syntax

```
Modulus(Dividend: Integer, Divisor: Integer): Integer
```

Dividend is the numerator of the fraction being divided.

Divisor is the denominator of the fraction being divided.

Example

```
Modulus(5,2)
```

The return value is 1.

Multiply

Description

Multiplies two specified integers and returns the result.

Syntax

```
Multiply(Int1: Integer, Int2: Integer): Integer
```

Example

```
Multiply(2,5)
```

The return value is 10.

MultiplyFloat

Description

Multiplies two specified floating-point numbers (float) and returns the result.

Syntax

```
Multiply(Float1: Float, Float2: Float): Float
```

Example

```
MultiplyFloat(4.76,2.3)
```

The return value is 10.948.

NextSibling

Description

Returns the next sibling for the current node based on the sort order used for the current hierarchy.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
NextSibling(): String
```

Example

```
NextSibling()
```

If A, B, C, and D are children of Z and the current node is B, then the return value is C.

NodeAccessGroups

Description

Returns a comma-delimited list of node access groups for the current user for the current node.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
NodeAccessGroups(): String
```

Example

```
NodeAccessGroups()
```

The return value is Accounts, Finance.

NodeExists

Description

Returns True if the specified node exists.

Syntax

```
NodeExists(NodeAbbrev: string): Boolean
```

NodeAbbrev is the name of the node to use.

Example

```
NodeExists(2000)
```

If node 2000 exists, then the return value is True.

NodeInHier

Description

Returns True if the specified node exists in the specified hierarchy.

Syntax

```
NodeInHier(NodeAbbrev, HierAbbrev: string): Boolean
```

NodeAbbrev is the name of the node to use.

HierAbbrev is the name of the hierarchy to use.

Example

```
NodeInHier(2000,Assets)
```

If the node 2000 is in the Assets hierarchy, then the return value is True.

NodeIsLeaf

Description

Returns True if the current node is a leaf node.

Syntax

```
NodeIsLeaf(): Boolean
```

Example

```
NodeIsLeaf()
```

If the current node is a leaf, then the return value is True.

NodeIsValidForPropertyHiers

Description

Returns True if the specified node satisfies the hierarchy constraint for the specified property. Also returns True if the property does not store node values or if no constraint is defined for the property.

This function can be local in scope if a local property is passed in the parameters.

Syntax

```
NodeIsValidForPropertyHiers(NodeAbbrev: String, PropAbbrev: String): Boolean
```

NodeAbbrev is the name of the node to use.

PropAbbrev is the name of the property to use.

NodePropValue

Description

Returns the value of the specified property of the specified node in the current hierarchy for a local node or in the current version for a global node.

This function can be local in scope if a local property is passed in the parameters.

Syntax

```
NodePropValue(NodeAbbrev: String, PropAbbrev: String): String
```

NodeAbbrev is the name of the node to use.

PropAbbrev is the name of the property to use.

Example

```
NodePropValue(2000,Abbrev())
```

Return value is 2000.

Not

Description

Returns the Boolean opposite of the specified Boolean expression.

Syntax

```
Not(Expression: Boolean): Boolean
```

Example

```
Not(NodeIsLeaf())
```

If the node is a limb, then the return value is True.

Now

Description

Returns the current system date and/or time.

Syntax

```
Now([DateTimeType: String]): DateTime
```

DataTimeType is optional for specifying which date portion to return. Valid values: Date, Time, Datetime. The default value is Date.

Example

```
Now()
```

Returns the current date and time; for example 3/25/2010 9:20:44 AM.

```
Now(Time)
```

Returns only the current time; for example 9:20:44 AM.

```
Now(Date)
```

Returns only the current date; for example 3/25/2010.

NumChildWith

Description

Returns the number of children for the current node where the specified expression evaluates to True.

Syntax

```
NumChildWith(Expression: Boolean): Integer
```

Example

```
NumChildWith(NodeIsLeaf())
```

If the node has two leaf children, then the return value is 2.

NumDescendantsWith

Description

Returns the number of descendants for the current node where the specified expression evaluates to True.

Syntax

```
NumDescendantsWith(Expression: Boolean): Integer
```

Example

```
NumDescendantsWith(NodeIsLeaf())
```

If the node has two children and each child has 10 leaf children, then the return value is 20.

Or

Description

Returns True if any of the specified Boolean expressions evaluate to True.

Syntax

```
Or(Expression1, Expression2,... ExpressionN: Boolean): Boolean
```

Example

```
Or(NodeIsLeaf(),Equals(Integer,PropValue(Level),3))
```

If the current node is a leaf or is at level 3 in the hierarchy, then the return value is True.

OrigPropValue

Description

Returns the value of the specified property for the originating node when using the HasSiblingWith or NumDescendantsWith functions.

This function can be local in scope if a local property is passed in the parameters.

Syntax

```
OrigPropValue(PropAbbrev: String): String
```

PropAbbrev is the name of the property to use.

Example

```
HasSiblingWith(GreaterThan(OrigPropValue(ID),ID())
```

If the current node's ID is 200 and it has any siblings with a node ID greater than 200, then the return value is True.

PadChar

Description

Returns a specified string lengthened using a specified pad character. Padding can be on the left or right of the original string. The resulting string is at least as long as the number of digits specified. If the original string is longer than the number of digits specified, the original list is returned.

Syntax

```
PadChar(String: String, PadChar: String; PadLeft: Boolean; NewLength:
Integer): String
```

String is the string value on which to perform the function.

PadChar is the character to use for padding the string.

PadLeft specifies whether to pad the string on the left. Valid values: 1, 0, T, F, t, or f.

NewLength is an integer specifying the length of the result.

Example

```
PadChar(102,0,1,6)
```

The return value is 000102.

PadList

Description

Returns a specified list lengthened using a specified pad character. Padding can be on the left or right of the original list. The resulting list is at least as long as the number of digits specified. If the original list is longer than the number of digits specified, the original list is returned.

Syntax

```
PadList(String, DelimChar, PadChr:String, PadLeft: Boolean,
NewLength:Integer): String
```

StringList is the list of strings to apply padding to, separated by the specified delimiter.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

Note

You must use the name of the delimiter (not the character) and include square brackets around the name.

PadChar is the character to use for padding the string.

PadLeft specifies whether to pad the string on the left. Valid values: 1, 0, T, F, t, or f.

NewLength is an integer specifying the length of the result.

Example

```
PadList(1;2;3;4,;,T,3)
```

The return value is 001;002;003,004.

ParentPropValue

Description

Returns the value of the specified property of the current node's parent node. Returns a blank string if the node has no parent, or if the current node is not a local node.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
ParentPropValue(PropAbbrev: String): String
```

PropAbbrev is the name of the property to use.

Example

```
ParentPropValue(Abbrev)
```

If the parent node name is Colas, then the return value is Colas.

Pos

Description

Returns the position (index) of the first character of the specified substring within the specified string using a case-sensitive search. A zero value is returned if the substring is not found within the string value.

Syntax

```
Pos(SubString: String, String: String): Integer
```

Substring is the string value for which to search.

String is the string value on which to perform the function.

Example

```
Pos(D,ABCDEFG)
```

The return value is 4.

PreviousSibling

Description

Returns the previous sibling for the current node based on the sort order used for the current hierarchy.

This function is local in scope and will not function properly if used in a global context.

Syntax

```
PreviousSibling(): String
```

Example

```
PreviousSibling()
```

If A, B, C, and D are children of Z and the current node is B, then the return value is A.

PropControllingHier

Description

Returns the name of the controlling hierarchy of the specified property in the current version.

Syntax

```
PropControllingHier(PropAbbrev: String): String
```

PropAbbrev is the name of the property to use.

Example

```
PropControllingHier(TimeBalance)
```

The return value is Accounts.

PropDefaultValue

Description

Returns the default value of the specified property definition.

Syntax

```
PropDefaultValue(PropAbbrev: String): String
```

PropAbbrev is the name of the property to use.

Example

```
PropDefaultValue(Currency)
```

The return value is USD.

PropertyCategories

Description

Returns a comma-delimited list of property categories for the current user.

Syntax

```
PropertyCategories(AccessType: String) :String
```

AccessType is the access level for a property category. Valid values: ReadOnly, ReadWrite, or Both.

Example

```
PropertyCategories(Both)
```

The return value is System, All, Essbase, Enterprise, HFM, Planning.

PropMaxValue

Description

Returns the maximum value of the specified property definition.

Syntax

```
PropMaxValue(PropAbbrev: String): Integer
```

PropAbbrev is the name of the property to use.

Example

```
PropMaxValue(Volume)
```

The return value is 10.

PropMinValue

Description

Returns the minimum value of the specified property definition.

Syntax

```
PropMinValue(PropAbbrev: String): Integer
```

PropAbbrev is the name of the property to use.

Example

```
PropMinValue(Volume)
```

The return value is 1.

PropValue

Description

Returns the value of the specified property for the current node.

This function can be local in scope if a local property is passed in the parameters.

Syntax

```
PropValue(PropAbbrev: String): String
```

PropAbbrev is the name of the property to use.

Example

```
PropValue(Volume)
```

The return value is 2.

RangeListContains

Description

Returns True if the specified list of ranges contains the specified value.

Syntax

```
RangeListContains(RangeList: String, Value: Integer, Delimiter: String):
Boolean
```

RangeList is a list of integer ranges to search, separated by the specified delimiter. For example, 1-100, 201-300

Value is an integer value to search for in the list of ranges.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]

- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

Example

```
RangeListContains(PropValue(MyRangeList),1,[Comma])
```

If the property MyRangeList' has a value of 1-10, 101-10000, then the return value is True, because 1 is contained in the specified range. However, RangeListContains(PropValue(MyRangeList),11,[Comma]) returns False, because 11 is not contained in the specified range.

> **Note**
>
> If you change MyRangeList to "1-5,6-10,101-1000", Data Relationship Management replaces this value with "1-10,101-1000", because it verifies RangeList and combines ranges with contiguous boundaries.

ReplacementAbbrev

Description

Returns the replacement (merge) node name for the current node if the node is inactive and a merge node is specified.

Syntax

```
ReplacementAbbrev(): String
```

Example

```
ReplacementAbbrev()
```

ReplacePropValue

Description

Returns the specified property value for the current node's replacement (merge) node if the node is inactive and a merge node is specified.

This function can be local in scope if a local property is passed in the parameters.

Syntax

```
ReplacePropValue(PropAbbrev: String): String
```

PropAbbrev is the name of the property to use.

Example

```
ReplacePropValue(Description)
```

ReplaceStr

Description

Returns the string with instances of the old pattern replaced by the new pattern.

Syntax

```
ReplaceStr(String: String,OldPattern: String,NewPattern: String,ReplaceAll:
Boolean): String
```

String is the string value on which to perform the function.

NewPattern is the string value with which to replace the found string.

OldPattern is the string value to search for.

ReplaceAll specifies whether to replace all occurrences of the search string with the replace string. Valid values: 1, 0, T, F, t, or f.

Example

```
ReplaceStr(A1;A2;A3,A,B,T)
```

The return value is B1;B2;B3.

RTrim

Description

Returns the specified value with all spaces trimmed from the end of the string.

Syntax

```
RTrim(String: String): String
```

String is the string value on which to perform the function.

Example

```
RTrim("100   "))
```

The return value is 100.

SortList

Description

Returns the specified list in a sorted order.

Syntax

```
SortList(InputList: String,Delimiter: String,IgnoreCase: Boolean,ItemType:
String)
```

InputList specifies the list to use.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include square brackets around the name.

IgnoreCase specifies whether to ignore case when sorting. Default value is False.

ItemType indicates the target data type for result list items. Valid values: string, integer, float date, time, and datetime. The default value is string. If nay item cannot be converted to the specified type, the function returns a blank string.

StripPadChar

Description

Removes a specified pad character from the beginning of a specified string and returns the modified value. If the original string contains fewer pad characters than are specified for StripCount, the original string value is returned.

Syntax

```
StripPadChar(String: String, PadChar: String, StripCount: Integer): String
```

String is the string value on which to perform the function.

PadChar is the character to use for padding the string.

StripCount is an integer specifying the number of characters to remove from the string. Zero removes all padded characters.

Example

```
StripPadChar(0003333,0,6)
```

The return value is 3333.

StrToBool

Description

Returns a Boolean value based on the specified string. If the string starts with a Y, T, or 1 (one) regardless of case or following characters, a True value is returned. If the string starts with N, F, or 0 (zero) regardless of case or following characters, a False value is returned.

Syntax

```
StrToBool(String: String): Boolean
```

String is the string value on which to perform the function.

Example

```
StrToBool(0)
```

The return value is False.

StrToFloat

Description

Returns the float value of the specified string. Returns zero (0) for a space or blank string.

If the specified string does not represent a floating point number, an error is returned.

Syntax

```
StrToFloat(String: String): Float
```

String is the string value on which to perform the function.

Example

```
StrToFloat(11.101)
```

The return value is 11.101.

StrToInt

Description

Returns the integer value of the specified string. Returns zero (0) for a space or blank string.

If the specified string does not represent an integer number, an error is returned.

Syntax

```
StrToInt(String: String): Integer
```

String is the string value on which to perform the function.

Example

```
StrToInt(101)
```

The return value is 101.

Stuff

Description

Returns the specified value with the specified characters replaced by the specified string.

Syntax

```
Stuff(PropAbbrev: String, CharsToReplace: String, ReplacementChars: String):
String
```

PropAbbrev is the name of the property to use.

CharsToReplace is the string value to search for.

ReplacementChars is the string value with which to replace the found string.

Example

```
Stuff(Abbrev(),GEO,RIO)
```

If Abbrev is GEO101, then the return value is RIO101.

SubString

Description

Returns a portion of the specified string starting at the specified index and containing the specified number of characters.

Syntax

```
SubString(String: String, Index: Integer, Count: Integer): String
```

SubString is the string value on which to perform the function.

Index is an integer representing the index position to start searching for the substring. Zero indicates the first character position in the string.

Count is a number representing the number of characters to search, beginning from the starting index.

Example

```
SubString(Colas,1,2)
```

The return value is Co.

Subtract

Description

Subtracts the second integer value from the first value and returns the result.

Syntax

```
Subtract(Minuend: Integer,Subtrahend: Integer): Integer
```

Minuend is an integer value

Subtrahend is an integer value.

Example

```
Subtract(10,2)
```

The return value is 8.

SubtractFloat

Description

Subtracts the second floating-point value from the first value and returns the result.

Syntax

```
SubtractFloat(Minuend,Subtrahend: Float): Float
```

Minuend is floating point value

Subtrahend is floating point value.

Example

```
SubtractFloat(8.09,3.76)
```

The return value is 4.33.

SumList

Description

Returns the sum of the items in a list, ignoring blank items. Returns a blank string if the list contains an item not of the specified item type.

Syntax

```
SumList(InputList: String,Delimiter: String,ItemType: String):Integer
```

InputList specifies the list to use.

Delimiter is the character to use to delineate items in the string list. Supported special characters:

- [comma]
- [space]
- [tab]

> **Note**
>
> You must use the name of the delimiter (not the character) and include brackets around the name.

ItemType indicates the expected item data type for list members. Valid values: integer, and float. The default value is float.

Example

```
SumList(1;2;3,;,Integer)
```

Return value is 6.

Trim

Description

Returns the specified value with all spaces trimmed from the beginning and end of the string.

Syntax

```
Trim(String: String): String
```

String is the string value on which to perform the function.

Example

```
Trim("   101   ")
```

The return value is 101.

UpperCase

Description

Returns a string value converted to uppercase.

Syntax

```
UpperCase(String: String): String
```

String is the string value on which to perform the function.

Example

```
UpperCase(smaller)
```

The return value is SMALLER.

UserName

Description

Returns the user name for the current user.

Syntax

```
UserName(): String
```

Example

```
UserName()
```

Return value is the user name.

XOr

Description

Returns True if one and only one of the specified Boolean expressions evaluates to True.

Syntax

```
XOr(Expression1:Boolean, Expression2: Boolean): Boolean
```

Example

```
XOr(NodeIsLeaf(),Equals(Integer,PropValue(Level),3))
```

If the node is either a leaf or is at level 3 in the hierarchy, the return value is True.

Function Groups

The following table groups functions by use.

Table 11-3 Function Groups

Function Group	Functions
Aggregate	• AvgList • MaxList • MinList • SumList

Table 11-3 (Cont.) Function Groups

Function Group	Functions
Change Tracking	• AddedBy • AddedOn • Changed • ChangedBy • ChangedOn • Now
Comparison	• Equals • GreaterThan • GreaterThanOrEqual • InRange • IsBlank • IsRangeListSubset • LessThan • LessThanOrEqual • RangeListContains
Conditional	• And • If • Not • Or • XOr
Data Type	• BoolToStr • FloatToStr • IntToStr • IsDataType • IsNumeric • StrToBool • StrToFloat • StrToInt
List	• ArrayCount • ArrayIndex • ArrayItem • Intersection • ListContains • ListDistinct • ListNodePropValues • ListNodesWith • SortList
Mathematical	• Add • AddFloat • Divide • DivideFloat • Modulus • Multiply • MultiplyFloat • Subtract • SubtractFloat

Table 11-3 (Cont.) Function Groups

Function Group	Functions
Node	• Abbrev • ID • InternalPrefix • NodeExists • NodeInHier • NodeIsLeaf
Property	• AncestorProp • AscNodeProp • DefaultProp • Descr • DualAncestorProp • HierNodePropValue • InheritedPropOrigin • IsDefinedPropVal • NodePropValue • OrigPropValue • ParentPropValue • PropControllingHier • PropDefaultValue • PropMaxValue • PropMinValue • PropValue • ReplacePropValue
Relationship	• Children • HasChildWith • HasParentNode • HasSiblingWith • InvertedLevel • IsBottomNode • IsNodeAbove • IsNodeBelow • ListAncestors • ListChildren • ListDescendants • ListRelatedNodesWith • ListSiblings • NextSibling • NumChildWith • NumDescendantsWith • PreviousSibling • ReplacementAbbrev

Table 11-3   (Cont.) Function Groups

Function Group	Functions
String Manipulation	• Concat • ConcatWithDelimiter • Decode • FlipList • Format • FormattedDate • HasCharacters • IsAlpha • IsAlphaNumeric • Length • LowerCase • LTrim • PadChar • PadList • Pos • ReplaceStr • RTrim • StripPadChar • Stuff • SubString • Trim • UpperCase
User	• NodeAccessGroups • PropertyCategories • UserName

12 Managing Dynamic Scripts

Dynamic scripting enables you to develop business logic for derived properties and validations using JavaScript. Dynamic scripts provide a more robust and better performing alternative to formulas, using a standard scripting language. Scripts allow for better organization and less complexity of logic through the use of multiple statements, variables, and in-line comments. Dynamic scripts also provide support for advanced concepts like looping and regular expressions.

Execution Contexts

There are several contexts for executing a script: property context, validation context, and request item property context. Each context defines different initial parameters and returns a different type of result.

Derived Properties Using Scripts

The Script deriver class enables dynamic scripts to be used by derived properties. Derived properties using scripts are available for versions, hierarchies, and nodes.

Table 12-1 Property Level Descriptions

Property Level	Parameter	Object
Version	version	VersionObject
Hierarchy	hierarchy	HierarchyObject
Global Node	node	NodeObject
Local Node	node	LocalNodeObject

For more information, see:

- Node Derived Properties
- Version and Hierarchy Properties

Node Derived Properties

In this context, you are passed a parameter called node. For global properties, the node is a NodeObject. For local properties the node is a LocalNodeObject. A script for a derived property must return a value and the value must be appropriate to the data type of the property that is being evaluated or executed. If the value returned by a script does not match the property data type, then it will be coerced: for example, a null value being returned for a Boolean property will be treated as false.

Note

Not all Oracle Data Relationship Management property data types have a JavaScript representation. See Data Type Conversions.

Version and Hierarchy Properties

In this context, you use a version parameter referencing a VersionObject or a hierarchy parameter referencing a HierObject. When defining your scripts, a version may not be loaded when the script is evaluated or executed. If a version or hierarchy derived property only accesses other version and hierarchy level properties, then the property is calculated regardless of the version load status. If a version or hierarchy derived property attempts to access node level information, then the version must be loaded or the property calculation will produce an error value. For example, if a version-level property attempts to get the list of orphans, that property will produce an error value when the version is not loaded; after the version is loaded, that same property will produce the correct value.

Validations Using Scripts

The Script validation class enables dynamic scripts to be used with validations. There are several different validation levels and some have different parameters. Following are the validation levels and parameters:

Table 12-2 Validation Levels and Parameters

Level	Parameter	Description
Any level	validation	Provides information about the validation currently executing
Hierarchy	hierarchy	HierarchyObject for the hierarchy being validated
GlobalNode	node	NodeObject for the global node being validated
Node	node	LocalNodeObject for the node being validated
Remove	node	NodeObject for the node being validated
Move	node	LocalNodeObject for the node being moved

Table 12-2 (Cont.) Validation Levels and Parameters

Level	Parameter	Description
	move	An object that contains information about the move: OldParent—LocalNodeObject for the original parent NewParent—LocalNodeObject for the destination parent IsPost/IsPre—Indicates whether this script is running just before the move or just after the move has been completed. The script will usually be run twice, once before the move and once after the move. Values—During the pre-move phase, simple key-value pairs can be stored in this object (for example, Values["key"] = "value"). During the post-move phase, these values are present, enabling you to store information about the pre-move state and compare it to the post-move state. All values are converted to String, Number, or Date objects. Complex objects are not currently supported.
Merge	node	The node being deleted or inactivated
	merge	An object that contains information about the merge: Target—NodeObject for the target of the merge IsInactivate—True if this is an inactivate operation IsDelete—True if this is a delete operation
Version	version	VersionObject for the version being validated

Governance Requests Using Scripts

Dynamic scripts may be used with workflow tasks in a governance request. Scripts are run in the context of a current request item and are used for calculating values to be used by the item, such as the Name or Parent of the node being updated.

Table 12-3 Governance parameters

Parameter	Description
requestitem	Current RequestItemObject for the request being calculated

Enumeration Constants

Certain properties are numbers that correspond to named constants, making your code easier to understand and maintain. For example, instead of using:

`if(nodeProp.PropOrigin == 2)` you can use `if(nodeProp.PropOrigin == PropOrigin.Overridden)`

Property Enumeration Constants

- DataType—Boolean, LeafNode, Date, Time, Float, Integer, Sort, Group, Node, LimbNode, String, Hier, Version, ListGroup, MultiNode, AscNode, AscNodes, AscGroup, Memo, FormatMemo, SortProp, Property, Query, StdQuery, GlobalNode, NodeProps, RangeList, DateTime, Hyperlink, HierarchyGroup
- PropLevel—Node, Hier, Version
- PropOrigin—Default, Inherited, Overridden, InheritedHier, InheritedVer, Derived, InheritedDomain, Unknown
- PropType—Invalid, System, Defined, Lookup, Derived, Stats, Validation, Verification, LimbAccessGroup, LeafAccessGroup, UserSpecific, RWDerived, SharedInfo

Validation Enumeration Constants

- ValidationLevel—Node, Hier, Version, GlobalNodes, Merge, Move, Remove
- ValidationType—None, RealTime, Batch, Both

Request Enumeration Constants

- WorkflowAction—AddLeaf, AddLimb, Update, Inactivate, Insert, Move, Remove, Delete
- WorkflowStageType—Submit, Enrich, Approve, Commit
- WorkflowStatus—None, Draft, Submitted, Calculated, Validated, PushedBack, Pending, Assigned, Claimed, Escalated, DeEscalated, Rejected, Committed

Note

The WorkflowStatus enumeration is used to return the RequestObject.Status current value for a request. However, some values are used internally only. The valid values for RequestObject.Status are: Draft, Submitted, Pending, Claimed, Escalated, Rejected, or Commited.

Supported JavaScript Data Types

Standard JavaScript data types are available, and Oracle Data Relationship Management uses them wherever possible. For example, dates are represented using the Date object. Functions are themselves objects, and a function invoked with *new* creates an object whose prototype points to the function's constructor prototype just as in any ECMA-compliant JavaScript environment.

Note

JavaScript Document Object Model (DOM) objects are not supported in Data Relationship Management scripts.

You must be familiar JavaScript syntax and built-in objects, including what methods are available. Some of the available data types:

- Array—Includes length, pop, push, concat, join, reverse, slice, shift, sort, and so on

  Note

  Due to changes in the JavaScript boxing of items by caching mechanisms, not all Array functions will work as expected or as they did in previous releases. For example, indexOf in JavaScript will compare objects based on memory locations, not the string or text value of items. Therefore, other methodologies should be considered when inspecting arrays. IndexOf() uses "===" comparison in JavaScript and there is not a single definition of "==" that is available. You can use JavaScript design patterns to implement your own specialIndexOf() to provide a "=="-style comparison.

- Boolean—Represents True and False
- Date—Includes Date.parse(), month, day, year, and so on
- Error—Uses try/catch error handling and access error.message
- Function—Supports the standard call and apply functionality
- Math—Includes random, max, pow, round, sin, cos, floor, sqrt, log, and so on
- Number—All numbers in JavaScript are of the floating-point type number
- RegExp—You can use language support for Regular Expressions or access them explicitly
- String—Includes concat, indexOf, lastIndexOf, substr, split, splice, search, replace, toUpperCase, toLowerCase, and so on

Globally available functions like parseInt, parseFloat, isNaN, decodeURI, encodeURI are also available.

Print Function

The print function allows you to output debug information while creating scripts. The results are displayed in the Warnings section of the script editor. Although the print function produces only output in a testing context, the engine must still construct the arguments; therefore, comment out any print statements before saving a script for production use.

Format Function

The Format function provides a much richer string formatting mechanism than standard JavaScript. The first parameter is a string that contains format specifiers surrounded with curly braces. Escape braces by doubling them, for example "{{" becomes "{" in the output. Format specifiers start at zero and increase incrementally. If you omit a specifier from a sequence, the equivalent parameter to the Format function is ignored. For example, "{1}" ignores the first value parameter to Format and uses the second.

There is one shortcut. You can call Format and pass a format specifier without braces and pass only one argument. The result is equivalent to Format("{0:<*specifier*>", <*argument*>)

The format specifiers work similarly to other languages like Java or C#. The syntax is {<*paramnum*>} or {<*paramnum*>;<*format*>}, where *paramnum* is a positive whole integer starting at zero and increasing sequentially. The format param depends on the type of the object passed in as that parameter.

The format parameters generally return values appropriate to the user's locale; for example, in the US "{0:0.00}" returns "1.23" while in Europe it returns "1,23"). Alternately, you can use the escaping support to explicitly override the locale and output the same value for all users. For example, "#\,###\,##0" would format a number using commas as thousands separators in all regions, regardless of culture settings.

Data Type Conversions

Not all Oracle Data Relationship Management property definition data types have corresponding representations in JavaScript. For any that do not have a corresponding representation, the StringValue and Value will be identical and you must ensure that you understand how to parse the string value. If returning a value for a property of one of these data types, you are also responsible for ensuring you return a proper string representation of that data type. If the stored value does not have a valid conversion to the property’s data type, then the value will be undefined.

List properties will return an Array with each element of the array containing objects of the type appropriate to the data type. For example, a Date property marked List would return an Array containing Date objects.

Lookup properties may not always return the data type expected when the lookup target is invalid, the key is not found in the lookup table, or the value in the lookup table is not valid for the data type. For example, if the value for a key-value pair is "TEST," but the data type is Date, then the result will be Undefined.

Following are the Data Relationship Management data types with their corresponding representation in JavaScript.

Table 12-4 Data Type Comparison

Property Definition Data Type	JavaScript Data Type
AscGroup	NodeObject Array
AscNode	NodeObject
AscNodes	NodeObject Array
Boolean	Boolean
Date	Date
DateTime	DateTime
Float	Number
FormatMemo	String
GlobalNode	NodeObject
Group	String Array
Hier	HierObject
Hierarchy Group	String (hierarchy group name)
Hyperlink	String (representing the URL)

Table 12-4 (Cont.) Data Type Comparison

Property Definition Data Type	JavaScript Data Type
Integer	Number
LeafNode	LocalNodeObject
LimbNode	LocalNodeObject
ListGroup	String Array
Memo	String
MultiNode	LocalNodeObject array
Node	LocalNodeObject
NodeProps	PropDefObject array
Query	String (query name)
Property	PropDefObject
Sort	Number
SortProp	PropDefObject
StdQuery	String (query name)
String	String
Time	String
Version	String (version name)

When calling another JavaScript derived property (or a derived property of a different node), because the value returned by that deriver is not converted to its string representation immediately, you can pass complex objects between derivers and delay coercion until the final result is returned by calling *toString()* on that complex object (except as noted for built-in conversions such as from Arrays).

Formatting Numbers

Numbers can only be formatted with a single shortcut character such as "G", or a composite of specifiers, such as "##0,000.0". If you attempt to use a shortcut character in a format specifier larger than one character, it will be copied to the output unchanged (treated as a literal character).

Run your production exports with the appropriate culture selected to ensure the output is correctly formatted.

Table 12-5 Single Character Shortcut Numeric Formats

Format	Description
D	Whole number (with locale-aware negative sign for negative numbers)
D<precision>	Whole number formatted to at least <precision> digits, zero-padded if necessary. For example, 123 with "{0:D5}" will output 12300.
E	Exponential (scientific) notation "1.234E+10"

Table 12-5 (Cont.) Single Character Shortcut Numeric Formats

Format	Description
F	Floating point number "123.456" (with locale-aware decimal separator and negative sign for negative numbers)
F<precision>	Floating point number rounded to <precision> significant digits after the decimal
G	General number format
N	Generalized Numeric format "123,456.789" (with locale-aware group/decimal separators and negative sign for negative numbers)
N<precision>	Generalized Numeric rounded to <precision> digits after the decimal
P	Percent (for 0.20146 outputs "20.14%" with locale-aware group/decimal separators and negative sign for negative numbers)
P<precision>	Percent rounded to <precision> significant digits (for 0.205 "{0:P0" outputs "21%")
X	Hexadecimal (base-16) output "4D2"

Table 12-6 Numeric Format Specifiers

Format	Description
0	Zero placeholder, outputs digit if present, otherwise zero
#	Digit placeholder, outputs digit if present, otherwise does not produce output
.	Locale-specific decimal separator
,	When placed between placeholders, outputs a locale-specific group separator (for 123456789 "{0:#,#}" outputs "123,456,789"). When one or more are placed immediately to the left of the decimal point (or implicit decimal point) the number is divided by 1000 for every comma (for 123456789 "{0:#,##0,,}" outputs "1,235").
%	Multiplies the number by 100 and outputs a locale-specific percentage symbol at the given location
E<sign>0	Exponential notation. At least one zero is required, with the number of zeros specifying the minimum digits in the exponent. <sign> is optional and can be: • + (always output sign +/- as required) • - (output - sign only for negative numbers)
\<char>	Escape character (<char> is treated as literal output)

Table 12-6 (Cont.) Numeric Format Specifiers

Format	Description
;	Section separator. If present, allows definition of different formats for positive numbers, negative numbers, and zeros. • One section "{0:#,#;}"—Identical to no section • Two sections "{0:#,#;-#,0}"—The first section applies to positive numbers and zero, the second section applies to negative numbers • Three sections "{0:#,#;-#,0;zero}"—The first section applies to positive numbers, the second section applies to negative numbers (If empty, the first section is used for negative numbers also), the third section applies to zero
Any other character	Copied to output unchanged

Formatting Dates

Dates can be formatted with a single shortcut character, such as: "G," or a composite of specifiers, such as "HH:mm". If you want to use a single character as a regular specifier and not a shortcut, prefix the string with %. For example: "%m" outputs the unpadded minute instead of the Month+Day.

Table 12-7 Single Character Shortcut Date Formats

Format	Description
t	Short Time "4:05 PM"
T	Long Time "4:05:07 PM"
d	Short Date "3/9/2013"
D	Long Date "Friday, March 09, 2013"
f	Long Date + Short Time "Friday, March 09, 2013 4:05 PM"
F	Long Date + Long Time "Friday, March 09, 2013 4:05:07 PM"
g	Short Date + Short Time "3/9/2013 4:05 PM"
G	Short Date + Long Time "3/9/2013 4:05:07 PM" (default)
m	Month + Day "March 09"
y	Month + Year "March, 2013"
r	RFC 1123 "Fri, 09 Mar 2013 16:05:07 GMT"
s	Sortable Date/Time "2013-03-09T16:05:07"
u	Universal Sortable Date/Time "2013-03-09 16:05:07Z"

Table 12-8 Date Format Specifiers (more than one character)

Format	Description Examples are for 2013-04-05 04:07:09 PM CST
yy	Year "13"
yyyy	Year "2013"
M	Month "4"
MM	Month "04"
MMM	Month "Apr"
MMMM	Month "April"
d	Day "5"
dd	Day "05"
ddd	Day "Sun"
dddd	Day "Sunday"
h	12-Hour "4"
hh	12-Hour "04"
H	24-Hour "16" (if 4 AM "4")
HH	24-Hour "16" (if 4 AM "04")
m	Minute "7"
MM	Minute "07"
s	Seconds "9"
ss	Seconds "09"
f	Fractions of a second (Can be repeated 1-4 times for more precision)
F	Fractions of a second without trailing zeros (Can be repeated 1-4 times)
t	AM or PM designator "P" (blank for 24-hour only cultures)
tt	AM or PM designator "PM" (blank for 24-hour only cultures)
z	GMT offset "-6"
zz	GMT offset "-06"
zzz	GMT offset "-06:00"
:	Time separator (locale-specific)
/	Date separator (locale-specific)
\<char>	Escape character (<char> is treated as literal output), for example: "{0:HH\h}" outputs "16h"
Any other character	Copied to output unchanged

Data Relationship Management Objects

Following are the Oracle Data Relationship Management objects with methods and properties described.

SysObject

One SysObject, called Sys, is automatically created. This object is available in all contexts, and provides general functions as well as information about the Data Relationship Management application. There are no properties for this object.

Table 12-9 SysObject Methods

Name	Description
FormattedDate (value, formatString)	Formats dates according to the Formula system rules. Useful for backward compatibility to exactly match old Formula properties. • value must be a Date object or a valid datetime string • formatString must be a valid formatting string (see the FormattedDate function)
GetNextID(key)	Returns the next available integer ID for a given string key value
GetPropDef(abbrev)	Returns a PropDefObject for the given property name. The name must be the fully qualified name.
GetRequestByID(int)	Returns a workflow request by ID.
GetSysPrefValue(abbrev)	Returns the value of the given system preference (for example, HierNodeSeparator)
InRange(dataType, input, min, max, minExclusive, maxExclusive)	Corresponds to the formula function InRange. Required parameters are dataType, input, and min.
IsNodeAbove(ancestor, child)	Returns True if ancestor is above child in the hierarchy. Returns False if parameters are not LocalNodeObjects or are not in the same hierarchy.
IsNodeBelow(descendant, parent)	Returns True if descendant is below parent in the hierarchy. Returns False if parameters are not LocalNodeObjects or are not in the same hierarchy.

Table 12-9 (Cont.) SysObject Methods

Name	Description
RunFormula(node, propDef, formulaString)	Runs a Data Relationship Management formula and returns the string results • node is either a NodeObject or LocalNodeObject. Your formula string must not make references to local properties when passing in a NodeObject, or an error will result. When passing a LocalNodeObject, you can reference all available global and local properties. • propDef—To be parsed or executed correctly, some formula functions require a property definition. When you use those functions, you must supply a property definition. Generally, the property definition characteristics (like Level, Global vs Local, and Type) must match, but it doesn't have to be the actual property that the formulaString is for. They can be unrelated. In most formulas, you can pass null for this parameter. Syntax is `Sys.GetPropDef(abbrev)`. For example: `Sys.RunFormula(node, Sys.GetPropDef("Custom.MyProp1"), "Concat(Prop value ', PropValue(Custom.MyProp2),' ,is, , valid)");` • formulaString is a legacy Data Relationship Management Formula; white space is considered a literal part of the Formula so it must be removed if necessary. **Note:** This is not considered a best practice and should be used only when necessary to achieve an exact match with legacy behavior. Performance is decreased when using this method.

PropDefObject

There are no methods for this object.

Table 12-10 PropDefObject Properties

Name	Description
Abbrev	The property definition name (including fully qualified namespace)
Cascade	True if the property values are inherited
ColumnWidth	The default export column width
DataType	A DataType enumeration value, for example DataType.String (see Enumeration Constants)
Descr	Description

Table 12-10 (Cont.) PropDefObject Properties

Name	Description
DefaultValue	Default Value of the property definition. The type depends on the data type of the property definition.
EditorLabel	Label
Global	True if property is a Global Node property
Hidden	True if property is hidden from the property grid
ID	ID
Level	A PropLevel enumeration value, for example, PropLevel.Node (see Enumeration Constants)
List	True if prop lets user select from a list of values
ListValues	Array of values from which a user can select
LookupValues	Lookup key-value pairs for a lookup property. Use the Key and Value properties of the objects in this array.
MaxValue	Maximum value
MinValue	Minimum value
Namespace	Namespace of property definition
PropType	A PropType enumeration value, for example, PropType.Defined (see Enumeration Constants)
PropClass	Deriver class (Formula or Script)
ReadOnly	True if property is read-only (such as a Core stats property)

VersionObject

Table 12-11 VersionObject Properties

Name	Description
Abbrev	Name
Descr	Description
HierCount	Number of hierarchies
ID	ID
NodeCount	Number of nodes

Table 12-12 VersionObject Methods

Name	Description
GetHierarchies()	Gets an array of all the hierarchies in the version that are available to the current user
GetGlobalNodes()	Gets an array of all the global nodes (NodeObjects) in the version
GetOrphans()	Gets an array of all the orphans (NodeObjects) in the version

Table 12-12 (Cont.) VersionObject Methods

Name	Description
HierByAbbrev(abbrev)	Gets a HierarchyObject by name
HierByID(id)	Gets a HierarchyObject by ID
NodeByAbbrev(abbrev)	Gets a NodeObject by name
NodeByID(id)	Gets a NodeObject by ID
NodeExists(abbrev)	Returns True if a global node exists with the given name
Prop(abbrev)	Gets the NodePropObject for the given property of the version
PropValue(abbrev)	Gets the value of the given property of the version. The return type depends on the data type of the property definition.

HierarchyObject

Table 12-13 HierarchyObject Properties

Name	Description
Abbrev	Name
Descr	Description
HierarchyUrl	Hierarchy URL
ID	ID
NodeCount	Number of nodes in hierarchy
SharedNodesEnabled	True if shared nodes are enabled
TopNode	The LocalNodeObject top node
Version	The VersionObject
VersionAbbrev	Name of version
VersionID	ID of version

Table 12-14 HierarchyObject Methods

Name	Description
NodeByAbbrev(abbrev)	Gets a NodeObject by name
NodeByID(id)	Gets a NodeObject by ID
NodeExists(abbrev)	Returns True if a local node with the given name exists
Prop(abbrev)	Gets the NodePropObject for the given property of the version
PropValue(abbrev)	Gets the value of the given property of the version. The return type depends on the data type of the property definition.

Common Node Properties and Methods

Some properties and methods are common to both NodeObject and LocalNodeObject although these two objects do not share a prototype chain.

In all cases where the value can differ because of the global or local context, the correct value is returned for that context. For example, when calling GetChildren() on a NodeObject, the resulting Array will contain NodeObjects. When making the same call on a LocalNodeObject, the resulting Array will contain LocalNodeObjects.

Table 12-15 Common Properties for NodeObject and LocalNodeObject

Name	Description
Abbrev	Core.Abbrev
AddedBy	Core.AddedBy
AddedOn	Core.AddedOn
Changed	Core.Changed
ChangedBy	Core.ChangedBy
ChangedOn	Core.ChangedOn
ChildNodeCount	Number of child nodes
Descr	Core.Descr
DomainAbbrev	Core.DomainAbbrev
DomainNodeAbbrev	Core.DomainNodeAbbrev
ID	Core.ID
Inactive	Core.Inactive
IsPrimary	True if the node is the primary for a shared node; False if node is not shared or not the primary
IsShared	True if the node is a shared node
Leaf	Core.Leaf
NodeApproved	Core.NodeApproved
Version	The node’s owner VersionObject
VersionAbbrev	The node’s version name
VersionID	The node’s version ID

Table 12-16 Common Methods for NodeObject and LocalNodeObject

Name	Description
GetChildren(sorted)	Gets an Array of the direct children of this node, optionally in sorted order. Default for sorted is False.
GetDescendants(inclusive, sorted)	Gets an Array of the descendants of this node, optionally including this node and/or in sorted order. Default for inclusive is True. Default for sorted is False.
NodeByAbbrev(abbrev)	Gets a NodeObject by name
NodeByID(id)	Gets a NodeObject by ID

Table 12-16 (Cont.) Common Methods for NodeObject and LocalNodeObject

Name	Description
NodeExists(abbrev)	Returns True if a global node with the given name exists
Prop(abbrev)	Gets the NodePropObject for the given property of the version
PropValue(abbrev)	Gets the value of the given property of the version. The return type depends on the data type of the property definition.

LocalNodeObject

Oracle recommends that you use the various *xxxWith* functions to locate other nodes in the hierarchy. For example *ChildrenWith* executes much faster than calling GetChildren() and iterating the results. Similarly, *GetReferenceInHier* is much faster and easier to use than calling GetReferences() and iterating the results.

Table 12-17 LocalNodeObject Properties

Name	Description
GlobalNode	Global NodeObject for the current node
Hier	HierarchyObject for the hierarchy the node is in
HierAbbrev	Core.HierAbbrev
HierID	Core.HierID
Level	Number representing the node’s level in the hierarchy
MissingPrimary	True if the primary node is not found
NodeUrl	Node URL
Parent	LocalNodeObject for the parent node of this node. Null is returned for the top node of a hierarchy.
ParentNodeAbbrev	Name of parent node
Primary	The primary node for this shared node. If the primary is not in this hierarchy, returns the primary in the first hierarchy in which it occurs. If you need the list of hierarchies in which the primary appears, call GetReferences() on the returned primary node. If a shared node or primary cannot be found, returns null.
PrimaryNotInHier	True if the primary node exists but not in this hierarchy

Table 12-18 LocalNodeObject Methods

Name	Description
AncestorsWith(func, maxResults, searchFromTop, inclusive)	Searches the ancestor chain for nodes that satisfy the given function. This is the fastest way to locate ancestors. Returns an Array of LocalNodeObject results. • Func must be a function that takes a single node argument and returns True if the node should be included in the results or False if it fails the test. • maxResults is optional and defaults to 1. Use 0 for no limit (all nodes that pass the condition). • searchFromTop is optional and defaults to False. Use True to start at the top of the hierarchy. • inclusive is optional and defaults to False. Use True to include the current node in the potential matches (it must still pass the test).
ChildrenWith(func, maxResults)	Searches the node's child list for nodes that satisfy the given function. This is the fastest way to find children. Returns an Array of LocalNodeObject results. • func must be a function that takes a single node argument and returns True if the node should be included in the results or False if it fails the test. • maxResults is optional and defaults to 1. Use 0 for no limit (all children that pass the condition).
DescendantsWith(func, maxResults, inclusive, depthFirst)	Searches the descendant chain for nodes that satisfy the given function. This is the fastest way to find descendants Returns an Array of LocalNodeObject results. • func must be a function that takes a single node argument and returns True if the node should be included in the results or False if it fails the test. • maxResults is optional and defaults to 1. Use 0 for no limit (all nodes that pass the condition). • inclusive is optional and defaults to False. Use True to include the current node in the potential matches (it must still pass the test). • depthFirst is optional and defaults to True. If True, each branch is examined all the way to its tips before backing up the tree and moving to the next branch. If False, all the children of a node are examined first, then each child's nodes are examined, and so on. If you have a good idea of where the node may be in the tree, picking the correct value here can greatly speed up the search.
GetAncestorEnumerator()	Gets a NodeEnumeratorObject that enumerates the ancestor nodes

Table 12-18   (Cont.) LocalNodeObject Methods

Name	Description
GetAncestors(inclusive)	Gets an Array of LocalNodeObject ancestors
GetChildEnumerator(sorted)	Gets a NodeEnumeratorObject that enumerates the child nodes. If sorted is True, then the children will be in sorted order.
GetDescendantEnumerator()	Gets a NodeEnumeratorObject that enumerates the descendant nodes
GetImplicitly SharedDescendants(inclusive)	Gets the child nodes of the primary node this shared node is related to
GetInvertedLevel()	Equivalent to the formula InvertedLevel function
GetReferences()	Gets an Array of LocalNodeObjects that are references for this node (all hierarchies this node appears in)
GetReferenceInHier(hierAbbrev)	Gets the reference to this node in the given hierarchy. If the hierarchy is not accessible or this node does not exist in that hierarchy, then the result will be null.
NextSibling()	Gets the next sibling of this node in the sort order
PreviousSibling()	Gets the previous sibling of this node in the sort order
SiblingsWith(func, maxResults, inclusive)	Searches the node’s siblings for nodes that satisfy the given function. Returns an Array of LocalNodeObject results. • func must be a function that takes a single node argument and returns True if the node should be included in the results or False if it fails the test. • maxResults is optional and defaults to 1. Use 0 for no limit (all ancestors that pass the condition). • inclusive is optional and defaults to False. Use True to include the current node in the potential matches (it must still pass the test).

NodePropObject

Table 12-19   NodePropObject Properties

Name	Description
Abbrev	The name of the property definition
ControllingHierarchy	The HierarchyObject for the property definition's controlling hierarchy in this version. If the property is not a global node property, does not have a controlling hierarchy, or the controlling hierarchy is not found, then the return value will be null.
Locked	True if the value is locked
Origin	A PropOrigin enumeration value, for example, PropOrigin.Overridden (see Enumeration Constants)

Table 12-19 (Cont.) NodePropObject Properties

Name	Description
Owner	The object that this value is associated with (VersionObject, HierarchyObject, NodeObject, or LocalNodeObject)
PropType	A PropType enumeration value, for example, PropType.Defined (see Enumeration Constants)
StringValue	The raw string value of this property. In the case of Derived or RWDerived properties this may be the property definition default value or the overridden value.
Value	The interpreted value of this property (for example, for DataType.Float and DataType.Integer, this value will be a Number object). Not all DataTypes necessarily have a non-string representation.

Table 12-20 NodePropObject Methods

Name	Description
GetPropDef()	Gets the PropDefObject for the node prop

RangeListObject

The RangeListObject represents a RangeList of values and can be used to inspect a RangeList property without having to manually parse strings. A new RangeListObject can also be constructed to return from a derived property of the appropriate data type.

Constructor Example

```
var x = new RangeListObject();


var y = new RangeListObject("1-10,20-25");


var z = new RangeListObject([{start:1, end:10},{start:20, end:25}]);
```

Table 12-21 RangeListObject Constructor Parameters

Parameters	Optional	Description
ranges	True	Range values for initialization. This parameter is optional. Two formats are accepted: • Array—An array where each element of the array is an object that has a start and end property indicating the range. Any object in the array that does not have these properties is ignored. • String—A comma-separated list of string entries. Each entry contains the start and end values separated by a dash (-) or equals (=) symbol.

Table 12-22 RangeListObject Properties

Name	Description
Ranges	An Array of objects. Each object has two properties: • start—The start of the range entry • end—The end of the range entry This property is read-only. To modify the range use the methods below.

Table 12-23 RangeListObject Methods

Name	Description
AddRange(start, end)	Adds a new range to the range list. This may expand an existing range entry or create a new one. To add a single number to the list, use it for both the start and end parameters. Both parameters will be coerced to integers if necessary.
Contains(value)	Returns True if the value is in the range list, otherwise False. value will be coerced to an integer if necessary.
IsSupersetOf(range)	Returns True if the current RangeListObject is a superset of the given RangeListObject. Passing another type of object is an error.
RemoveRange(start, end)	Removes a range from the list. This removal may split an existing range entry into two or remove an entry entirely. To remove a single number from the list use it for both the start and end parameters. Both parameters will be coerced to integers if necessary.

NodeEnumeratorObject

A NodeEnumeratorObject is a more efficient way to operate on a list of nodes. Instead of building the entire list all at once, the enumerator grabs only one node at a time as needed. If you find what you are looking for halfway through the list, you can abandon the enumerator. Properties and methods that return an Array of node objects must build the entire array immediately, whether you access the items at the end of the array or not.

The enumerator starts with a null Current value. You must call MoveNext() to advance the enumerator to the first node in the list.

Note

A good practice is to use the *With* methods like *AncestorsWith* or *SiblingsWith* methods when you need to find only a few nodes out of all possible matches and need to iterate the list only one time. If you need to cycle over the list of ancestor nodes multiple times or you know you will need most or all of the ancestors, then an enumerator may be faster.

Table 12-24 NodeEnumeratorObject Methods

Name	Description
GetCurrent()	The current node (either a NodeObject or LocalNodeObject depending on the context).
MoveNext()	Advances the enumerator to the next node. Returns False if there are no more nodes to enumerate.

ValidationObject

Table 12-25 ValidationObject Properties

Name	Description
Abbrev	Name of the validation (including fully-qualified namespace)
Descr	Description
EditorLabel	Label
Cascade	True if validation assignment is inherited
ValidationClass	Name of validation class
ValidationLevel	A ValidationLevel enumeration value, for example ValidationLevel.Node (see Enumeration Constants).
ValidationType	A ValidationType enumeration value, for example ValidationType.Batch (see Enumeration Constants).

Validation Scripts

- The validation script returns a JavaScript object that contains a property named "success". If the script returns a Boolean value or a non-Boolean object (for example, Number or String), then its value is converted to Boolean using standard JavaScript conversion rules and then assigned to the success property. The script can optionally return a JavaScript array of values in a property named parameters. The array values are substituted into the failure message of the validation using string substitution.

- You can return a Boolean value (True or False). If you return True, the validation succeeds; otherwise it fails. If you do not return a value, it is considered the same as returning False.
- If you return a non-Boolean object, such as Number or String, it is converted to Boolean then returned. Standard JavaScript conversion applies. Numbers equal to zero, empty strings, and null or undefined objects are interpreted as false. All other values are true.
- If you return a complex object that contains a property named "success," that success property is converted to Boolean and used as the return value of the validation. You can optionally return an Array of values in a property named "parameters." This is a JavaScript Array object, that needs to be populated and then used in the parametrized Failure Message. The parameters are substituted into the failure message of the validation using string substitution. You should return the correct number of values corresponding to the placeholders in the failure message. If you return extra parameters they are ignored. If you do not return enough parameters, the missing parameters are considered empty strings.

RequestObject

The RequestObject represents a governance request, including request header, and items. The Items property represents a list of the request items added to the request. A key attribute is the Version property, the target version for the request including its hierarchies and nodes, all accessible via the relevant script objects.

Table 12-26 RequestObject Properties

Name	Description
ID	ID
Title	Title of request
Version	Target version for request
ModelName	Workflow model for request
StageName	Current stage of request
StageType	WorkflowStageType enumeration value, for example, WorkflowStageType.Submit (see Enumeration Constants)
Status	WorkflowStatus enumeration value, for example, WorkflowStatus.Submitted (see Enumeration Constants)
Items	List of RequestItemObject added to the request

RequestItemObject

The RequestItemObject represents an individual request item for a governance request, including information about the current task and the node being updated, along with the details (property values) for the item. The Request property provides access to the full request object for the item, including header properties and other items.

The NodeNamePendingInRequest method is used for identifying potential node name conflicts with other in-flight requests for the target version, returning True if an item on another pending request contains an Add item for the same node name.

Table 12-27 RequestItemObject Properties

Name	Description
ItemID	Item ID

Table 12-27 (Cont.) RequestItemObject Properties

Name	Description
RequestID	Request ID
Request	Request object to which the item belongs
NodeName	Core.Abbrev of node being updated
Description	Core.Descr of node being updated
HierarchyName	Hierarchy of node being updated
ParentName	Core.Parent of node being updated
TaskName	Workflow task name of request item
TaskAction	WorkflowAction enumeration value, for example, WorkflowAction.AddLimb (see Enumeration Constants)
TaskDomain	Domain name (if any) of workflow task
ItemDetails	List of RequestItemDetailObject for request item

Table 12-28 RequestItemObject Methods

Name	Description
NodeNamePendingInRequest(name)	Accepts a parameter of a specific node name to test. Returns True if an in-flight request other than the current one for the version contains an AddLimb/Leaf item with the specified name.

RequestItemDetailObject

The RequestItemDetailObject represents an individual request item detail for a governance request, corresponding to a single property value.

Table 12-29 RequestItemDetailObject Properties

Name	Description
CalcValue	Calculated value of property
HasCalcValue	Returns True if the value is calculated
Modified	Returns True if the value was modified in the request
PropertyName	Name of property
Value	Value of property

Execution Environment

The Oracle Data Relationship Management engine is a multithreaded, multimachine environment and scripts may execute simultaneously on multiple threads and across machines. While you can create values and store them in the global scope, you should not rely on this behavior because when your script executes on another thread that global value will not be present. Similarly, global values are not updated across Data Relationship Management engine instances or machines. In addition, becauseData Relationship Management supports multiple live versions, if you rely on calculating a value for a node and storing that value in the global scope, you may produce incorrect values if a different script accesses the property for another node.

Note

For the same reason that you should not store variables in the global scope, you should also avoid modifying the built-in Data Relationship Management object prototypes, because you cannot be sure that your modifications have been made across all engine instances and threads.

Setting Script Timeouts

To prevent excessive engine locks, scripts that take too long to execute without returning a value are terminated based on a time-out setting. The script time-out can be set for each property definition and validation.

The time-out is per execution context, so if an export is exporting the script property of 100 nodes and the time-out for the property is set to 30 seconds, then the export may take up to 50 minutes, because each node can take 30 seconds to evaluate its property. However if a script property calls another script property, then it does not increase the time-out. For example, if PropA has a 10 second time-out, PropB has a 20 second time-out, and PropA calls PropB which then starts a long-running calculation, when 10 seconds have elapsed, the evaluation of PropA is terminated because its original time-out was exceeded.

Preventing Infinite Loops

A script that results in an infinite loop (also known as a stack overflow) is a serious error which can cause a server process to terminate unexpectedly. Although Data Relationship Management attempts to prevent such scripts from executing, you should exercise caution when writing self-referencing, or recursive, scripts. Always test new scripts in a development environment before deploying to production.

A simplified example of a script that will loop infinitely is shown below. Because the script includes a call to itself, but never terminates execution, the engine executing the function will eventually terminate due to lack of resources. Lastly, because the script never calls the Data Relationship Management engine, there is no chance to catch the overflow and stop the script.

```
function badFunc(a) { badFunc(a); }


badFunc("oops");
```

Performance Considerations

For the best performance, avoid referencing formula-derived properties from a script, and vice versa. Scripts in general offer the best opportunity for performance tuning optimization, compared to formulas, due to considerations such as just-in-time (JIT) compilation for native hardware, including 64-bit processors. Scripts are also tuned by the JIT compiler for actual execution characteristics and will run faster over time.

Creating Dynamic Scripts

Dynamic scripts are created in the script editor which is available on the Parameters tab for derived property definitions and validations.

The script editor is also available when calculating a Name or Parent during a governance workflow task.

> **Note**
>
> When calculating parent names, any use of special characters must follow the standard JavaScript rules for escaping special characters. For more information, see "Naming Nodes" in *Oracle Data Relationship Management User's Guide*.

To create a dynamic script:

1. Enter the script in the text area.

   > **Note**
   >
   > To insert a property, place your cursor in the script and click **Insert Property**. A list of properties is displayed. Select a property and click **OK**.

2. Make selections from the following options:
   - **Script Timeout**—The number of seconds until the script times out.
   - To evaluate the script with a selected node, click ⋯ and select a node. The node's current property values are used in the script. Click **Evaluate**. The result is displayed at the bottom of the script designer.
3. To test the script, click **Evaluate**.

13
Managing Node Types

Node types enable hierarchy nodes to be viewed and managed differently based on their relationships and attribution. Nodes of a specific node type share the same characteristics:

- Properties
- Validations
- Glyph

A hierarchy can have nodes of different node types and the same node can be of different node types in different hierarchies. Examples of node type usage include GL accounts, cost centers, consolidation entities, product groups, forecast points, and so on.

To categorize nodes by node type:

1. Determine the node types that are necessary to categorize nodes within a hierarchy.
2. Identify properties that are relevant (or not relevant) to each node type.
3. Identify validations that are relevant (or not relevant) to each node type.
4. Optionally, assign a glyph to each node type.

Defining Node Types

To define a node type:

1. On the Home page, select **Administer**.
2. From **New**, select **Node Type**.
3. Enter a name and description for the node type.
4. **Optional**: Select a glyph to use for the node type
5. On the **Properties** tab, select properties from the Available list to associate with the node type. Use the arrows to move properties to the **Selected** list.
6. On the **Validations** tab, select validations from the Available list to associate with the node type. Use the arrows to move validations to the **Selected** list.
7. Click **Save**.

Editing Node Types

To edit a node type:

1. On the Home page, select **Administer**.
2. Under **Metadata**, expand **Node Types**.
3. Select a node type and click ✎.
4. Do any of the following:
   - Edit the description.

- Change the glyph to use for the node type
- Add or remove properties
- Add or remove validations

5. Click **Save**.

Deleting Node Types

To delete a node type:

1. On the Home page, select **Administer**.
2. Under **Metadata**, expand **Node Types**.
3. Select a node type and click ✖.
4. Click **Delete this Item** to confirm the deletion.

Working with Node Glyphs

Glyphs are images that are associated to node types and are displayed as the icon for a node in the Oracle Data Relationship Management user interface. You can create new glyphs and modify existing glyphs. You can also delete glyphs that you no longer want to use. Glyphs must be provided in a PNG format.

To add a node glyph:

1. On the Home page, select **Administer**.
2. From **New**, select **Glyph**.
3. Enter a name for the glyph and add a description.
4. Click **Browse** and select the PNG file.
5. Click **Upload**.
6. Click **Save**.

To modify a node glyph:

1. On the Home page, select **Administer**.
2. Under **Metadata**, expand **Glyphs**.
3. Select a glyph and click ✎.
4. Click **Browse** a select the different PNG file.
5. Click **Upload**.
6. Click **Save**.

To delete a glyph:

1. On the Home page, select **Administer**.
2. Under **Metadata**, expand **Glyphs**.
3. Select a glyph and click ✖.
4. Click **Delete this Item** to confirm the deletion.

14

Working with System Preferences

System Preferences enable administrative users to edit settings that control the behavior of Oracle Data Relationship Management.

System Preferences

The following table describes Oracle Data Relationship Management system preferences.

Table 14-1 System Preferences

System Preference	Type	Description
AllowAsOf	Boolean	True forces capture of core actions and creates a baseline version to allow the creation of As-Of versions. If this preference is set to False, As-Of versions cannot be created. Default value is True. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
AllowNextIDGeneration	Boolean	True enables automatic Next ID generation. Default value is False.
AllowNextIDKeyCreation	Role	List of roles allowed to create a new key in NextID feature. Default values are Interactive User, Data Creator, Data Manager.
AllowPru	Boolean	True enables the pruning option which allows a non-admin user to remove a node that has children. If False, a non-admin user cannot remove a node that has children. Default value is True.
AllowRelaxedMove	Boolean	When a node is moved, True allows the new parent to take precedence over any conflicting parental relationships for the node in other hierarchies. Default value is False.
AllwSpac	Boolean	True allows spaces in node names. Default is True.

Table 14-1 (Cont.) System Preferences

System Preference	Type	Description
AnalyticsNodeCountUpdateTime	String	Specifies a time of day, in local time using 24-hour format, when the node counts for versions and hierarchies for all loaded, normal versions are to be updated. For example, 2:15 PM would be entered as "1415". The default time is 3:00 AM.
ApprovalGroups	String	Comma-delimited list of approval groups.
ApprovalGroupTrackProperties	String	Delimited list of approval properties tracked by groups.
ApprovalPropertyByApprovalGroup	String	Global boolean approval property by approval group.
AuthMethod	String	User authentication method: • Internal—Users are only authenticated within Data Relationship Management. • CSS (External)—Users are only authenticated externally. Requires access to Shared Services. • Mixed—Users are authenticated internally or externally based on a setting for each individual user. Default value is Internal. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
CopyLcl	Boolean	True copies local values when a node is copied. Default value is True.
DefaultCurrentVersion	Version	Default current version. This preference can be set using the Make Default option for versions.
DefaultPreviousVersion	Version	Default previous version. This preference can be set using the Make Default option for versions.
DefaultPropCopyMode	String	Default property copy mode. Valid values are Overridden, Selected, and ForceAll. Default value is Overridden.
EnablePropCopyOptions	Role	List of roles allowed access to the property copy options. Default values are Interactive User, Data Creator, Data Manager.
EnforceListProps	Boolean	True allows updates to a List Property with values from the pre-defined list only. Default value is True.

Table 14-1 (Cont.) System Preferences

System Preference	Type	Description
FiltrChr	String	Set of characters for the Replace function on the Output Option screen of exports.
FindByProperties	Property	List of properties available to search with when browsing a hierarchy. The properties displayed are those to which a user has access. Also, the properties displayed may not be applicable to all hierarchies. **Note:** The ADMIN user cannot be added to custom Property Categories in Data Relationship Management. As a consequence, if a property listed in the FindByProperties system preference is not added to a Property Category that ADMIN is already a member of, then ADMIN will not be able to perform a Find with that property in the Hierarchy Browse window.
FindWildCardAppend	Boolean	True appends an asterisk (*) to the Find criteria when Exact Match is not selected. Default value is False.
FindWildCardPrepend	Boolean	True prepends an asterisk (*) to the Find criteria when Exact Match is not selected. Default value is False.
GlobalPropLocalOverride	Property	List of properties to exclude from local checks on global properties. These are used when GlobalPropLocalSecurity is enabled. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
GlobalPropLocalSecurity	Boolean	True enforces local security on global properties. Changes to global properties are checked against local security (node access levels) for the user for all hierarchies where the node exists. Default value is False. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
HierSep	String	Hierarchy and node separator character. Default value is tilde (~).

Table 14-1 (Cont.) System Preferences

System Preference	Type	Description
IdleTime	Integer	Number of minutes to session time out on the application server. Default value is 60. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
Inactivate	Role	List of user roles allowed to inactivate nodes. Default value is all roles.
InactiveChanges	Role	List of roles allowed to change inactive nodes. Default values are Data Manager, Application Administrator, Access Manager.
InvDescr	String	List of invalid characters for node description property.
InvName	String	List of invalid characters for node name. **Note:** Characters in this list cannot be used as the delimiter with shared nodes.
JobResultsMaxSize	Integer	For jobs that are run using the Client File option, maximum size (in bytes) of results saved to the Job History. Job results exceeding this size are not saved to the Job History. The default value is 10,000,000 bytes. A negative value indicates that all results, regardless of size, are saved to the Job History. **Caution:** Disabling JobResultsMaxSize by setting to a negative value is strongly discouraged because this can significantly impact performance for large jobs. **Note:** JobResultsMaxSize does not apply to Exports run using the Server File or Database Table options.
JobResultsRetentionAge	Integer	Number of days to retain archived job result detail in history. A value of zero indicates that job results are never purged from history. **Note:** Job results are purged to manage database size. Disabling the purge may result in significant database growth over time.
LeafEdit	Role	List of roles allowed to change the Leaf property. Default values are Data Manager, Data Creator, Application Administrator, Access Manager.

Table 14-1 (Cont.) System Preferences

System Preference	Type	Description
LockoutInactivity	Integer	Maximum number of days of inactivity before a user is locked out. Default value is 30; zero indicates no maximum.
LockoutInvalidLogins	Integer	Maximum number of invalid logins before a user is locked out. Default value is 6; zero indicates no maximum.
LossLevel	String	Loss level to capture. Valid values are: • Defined • All Default value is Defined. Selecting All can significantly impact system performance for removed or deleted nodes with many property values. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
LRUPropertyCacheSize	Integer	Maximum size for the LRU property cache. The LRU Property cache stores calculated values that may be accessed multiple times. Generally, the default for this preference should be used and should not be changed.
MaxDescr	Integer	Maximum number of characters for node description. Valid values are 12 to 255. Default value is 80.
MaxFileNameLength	Integer	MaxFileNameLength is a value between 8 and 255 indicating the length of the filename (not including the path) for server-written output files (Exports, Version Backups, etc.)
MaxLeaf	Integer	Maximum number of characters for the leaf name. Valid values are 3 to 20. Default value is 255.
MaxLimb	Integer	Maximum number of characters for the limb name. Valid values are 3 to 20. Default value is 255.
NodeApprovedSecurity	Role	List of roles allowed to view and update. the NodeApproved system property for nodes
PasswordDuration	Integer	Number of days that a user password is valid. Valid values are 1 to 9999. Default value is 30.

Table 14-1 (Cont.) System Preferences

System Preference	Type	Description
PasswordMaxLength	Integer	Maximum number of characters for user password. Valid values are 0 to 255. Zero indicates no minimum. Default value is zero.
PasswordMinLength	Integer	Minimum number of characters for user password. Valid values are 0 to 9999. Zero indicates no minimum. Default value is 6.
PasswordPolicyEnabled	Boolean	True requires the password to contain three of the following elements: • Uppercase letters • Lowercase letters • Numbers • Special characters Default value is True.
PasswordWarningPeriod	Integer	Positive or negative number to indicate how many days before (-) or after (+) the password expiration date to warn users to change their password before no longer allowing them to log in. Valid values are -30 to 30. Default value is 1.
RenameLeaf	Role	List of roles allowed to rename leaf nodes. Default values are Data Manager, Application Administrator, Access Manager.
RenameLimb	Role	List of roles allowed to rename limb nodes. Default value is all roles.
ReqMerge	Boolean	True requires merge for inactivates or deletes when UseMerge is enabled. Default value is False.

Table 14-1 (Cont.) System Preferences

System Preference	Type	Description
SharedNodeDelimiter	String	Specifies the delimiter between the node name and the shared node suffix. The SharedNodeDelimiter character should not be used anywhere that would affect node names. Default value is colon (:). **Caution:** Different characters must be used when setting up the SharedNodeDelimiter and SharedNodeSequenceSeparator system preferences. For example, if the SharedNodeDelimiter is a colon, the SharedNodeSequenceSeparator character cannot be a colon. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
SharedNodeIdentifier	String	Specifies the identifier to be used after the shared node delimiter. Default value is Shared. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
SharedNodeMaintenanceEnabled	Boolean	True enables shared nodes. Default value is False. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
SharedNodeNamingType	String	Specifies the alternate name for shared nodes. Valid values are: Suffix or Prefix. Default is Suffix **Note:** A change to this preference requires a restart of the Data Relationship Management application.
SharedNodeSequenceLength	Integer	Specifies the length of the uniqueness key when using numeric sequence type. Default value is 3. **Note:** A change to this preference requires a restart of the Data Relationship Management application.

Table 14-1 (Cont.) System Preferences

System Preference	Type	Description
SharedNodeSequenceSeparator	String	Specifies the separator character to be placed after the shared node identifier. Default value is dash (-). **Caution:** Different characters must be used when setting up the SharedNodeDelimiter and SharedNodeSequenceSeparator system preferences. For example, if the SharedNodeDelimiter is a colon, the SharedNodeSequenceSeparator character cannot be a colon. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
SharedNodeSequenceType	String	Specifies the type of uniqueness key. Valid values are Numeric or Ancestors. Default is Numeric. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
SortLimbsFirst	Boolean	True controls the sorting of limb nodes first followed by leaf nodes. If False, limb and leaf nodes can be sorted together. This preference affects hierarchy exports, display, and node lists. Default value is True.
TopNodeParentString	String	Used in Import and Export to denote parent value for a top node. Default value is None.
TransactionLevels	String	List of transaction levels to capture. Turning on As-Of or specifying result or loss actions forces core actions to be captured. Valid values are: • Logged Action • Core Action • Result Action • Loss Action **Note:** Transactions at the Admin level are always logged regardless of this system preference. Default values are Logged Action, Core Action, Result Action, Loss Action. **Note:** A change to this preference requires a restart of the Data Relationship Management application.
UpName	Boolean	True uses uppercase always for the node name Default value is False

Table 14-1 (Cont.) System Preferences

System Preference	Type	Description
UseChangeApproval	Boolean	True enables change approval. Default value is False.
UseMerge	Boolean	True enables use of Merge methodology for inactivated and deleted nodes. **Note:** If ReqMerge is True, then the system requires a merge node to be specified. If ReqMerge is False, then a merge node is optional unless the node approved property is True. The node approved property is set to True when a version is finalized or when it is specifically set to True by a user with appropriate access. Default value is False.
ValSec	Boolean	True checks node access group security to determine whether a user can run batch validations for a node. Default value is False.
WarnHL	Integer	Maximum number of nodes to be displayed for lists such as Descendants, Children, Query Results, and so on. Minimum value is 1000. If set to a value less than 1000, then 1000 nodes are displayed. Default value is 5000.

For more information, see:

- Setting Transaction History Logging Levels
- Setting Up Change Approval

Setting Transaction History Logging Levels

You must have application administrator privileges to set Oracle Data Relationship Management Transaction History logging levels. Set the TransactionLevels system preference to specify the action types to capture in the transaction history.

Local Security for Global Properties

You use two system preferences — GlobalPropLocalSecurity and GlobalPropLocalOverride — to control local security on global properties.

To set Transaction History logging levels:

1. In the Data Relationship Management Web client, select **Administer**.
2. Under **Metadata**, expand the **System Preferences** and edit the **TransactionLevels** preference.
3. In **TransactionLevels**, select transaction level types:

- **Logged Action** records basic logging information, such as users logging in and logging out.
- **Core Action** records actions that change the version, hierarchy, or node information, such as Add Node, Change Property, or Move Node.
- **Result Action** records actions that result from core actions. For example, if the "clear all below" core action is performed, then properties are cleared from individual nodes. Clearing properties from the individual nodes is a result action.
- **Loss Action** records loss of data due to a core action. For example, when a node is deleted, the defined properties for that node are deleted, which is a loss action. Loss actions are controlled by the LossLevel system preference.

> **Note**
>
> If the Loss Action is specified, or if the AllowAsOf system preference is turned on, then Core Actions are tracked, even if not set in the TransactionLevels system preference.

4. Set the LossLevel preference:
   - **Defined**—Only values that are specifically set at the node are tracked when the node is deleted.
   - **All Items**—Derived, default, and inherited values are tracked in the LossAction.
5. Stop and restart the application, or restart the Data Relationship Management service.

Setting Up Change Approval

The change approval system in Oracle Data Relationship Management enables you to define approval groups and tie them to an approval flag that is triggered by a set of properties or special actions. This allows normal users to make changes and approvers to run a query and then set the approval flag as needed.

The systems preferences that determine the behavior of the change approval in Data Relationship Management are:

- UseChangeApproval—Set to True to turns on use of change approval.
- ApprovalGroups—A comma-delimited list of the names for the approval groups used in the system.
- ApprovalGroupTrackProperties—If UseChangeApproval is True, defines properties that are tracked that will trigger a change of the approval flag to False for this group. The format is xxx[a,b,c],yyy[d,e,f]... where xxx and yyy are sales groups defined in the ApprovalGroups preference and a,b,c,d,e,f are property names. For example, Sales[Custom.SalesGroup,{NodeMove}],Treasury[Custom.AccountDescription,{NodeAdd}].

  Special actions that can be included in the property list are:
  - {NodeAdd}—Triggers the Approval Needed mechanism on an added node.
  - {NodeInactivate}—Triggers the Approval Needed mechanism on an inactivated node.
  - {NodeReactivate}—Triggers the Approval Needed mechanism on a reactivated node.
  - {NodeInsert}—Triggers the Approval Needed mechanism on an inserted node.
  - {NodeRemove}—Triggers the Approval Needed mechanism on a removed node.

  - {NodeMove}—Triggers the Approval Needed mechanism on a moved node.
- ApprovalPropertyByApprovalGroup—If UseChangeApproval is True, defines the global, boolean property to set to False if any of the trigger properties are changed or the special actions are used. The format is xxx:bbbb,yyy:cccc…where xxx and yyy are sales groups defined in the ApprovalGroups preference and bbbb and cccc are the names for the global, boolean properties to be used to store the approval flag for the groups, for example, Sales:Custom.SalesApprovedFlag,Treasury:Custom.TreasuryApprovedFlag.

Configuring System Preferences

To configure System Preferences:

1. On the Home page, select **Administer**.
2. Under **Metadata**, expand **System Preferences**.
3. Select a system preference and click ✎.
4. Modify the value and click **Save**.

15

Working with External Connections

Application administrators can define and configure common connections to external file systems, databases, and Web services. Imports, exports and books can share file and database connections to minimize maintenance of connectivity information. Database and Web service connections can be configured with external operations to lookup data in an external system or commit data changes to an external system. External connections enable the application server to directly access, read, or write data to these resources.

> **Note**
>
> You must set up external resources before defining external connections.

External Operations

External operations can be defined for Web service or database external connections. External operations are configured as either lookup or commit. Lookup operations read data from an external system. Commit operations write data to an external system. Database and Web service connections can support multiple operations. For more information, see External Commits and External Lookups.

Defining External Connections

To define an external connection:

1. On the Home page, select **Administer**.
2. From **New**, select **External Connection**.
3. Enter a name and description.
4. From **Object Access**, select **Standard**, **System** or a custom group.
5. Select a connection type: **Server File**, **FTP**, **Database**, or **Web Service**.
6. Do one of the following:

   - If you selected **Server File**, enter a UNC path to the server and click .

     > **Note**
     >
     > The Windows user account used by the Oracle Data Relationship Management application server is automatically used for Server File connections. The default Windows user account used for the **Oracle DRM Server Processes** Windows service is **Local System account**. The account used for the service must be able to access the UNC path for proper Server File connectivity. Additionally, the UNC path must have the appropriate permissions for the service account to read and write files.

- If you selected **FTP**, enter the following information:
  - Host Server
  - User ID
  - User Password
  - Click .
- If you selected **Database**:
  - Select the **Data Access Provider**: Oracle, SqlServer, or OleDb.
    * Enter a Database Connection Timeout value
    * Enter a Database Command Timeout value
  - Enter the **Connection String**.
  - Enter your user ID and password

    Note

    To establish a writable external connection, the administrator must have SELECT, INSERT, and DELETE access. A user who has only SELECT access can establish a read-only external connection to tables and views.

  - Click .
  - On the **Allowed Objects** tab, to filter a large list, do any of the following:
    * Select or enter a schema/owner, using wildcards if needed.
    * Enter the name of an object, using wildcards if needed.
    * Select **Include Views** to include views where the privilege is at least SELECT. Note that views are always read only.
    * Select **Include Read-Only Tables** to include tables where the privilege is at least SELECT but does not include both INSERT and DELETE.
    * Click and then select objects from the **Available** list. Use the arrows to move objects to the **Selected** list.
    * **Optional:** To use the **Quick Add** section, enter the schema/owner and name of the object that you want to add and click the arrow to move it to the **Selected** list.
  - To add an external operation, click the **External Operations** tab, click **Add**, and then do the following:
    * Enter **Name** for the operation. The name must be unique for the parent External Connection.
    * Enter **Description** text describing the purpose of the operation.
    * Select the **Operation Type**—Lookup or Commit. This selection is used to filter the list of operations available for selection with the External Lookup and External Commit features.
    * Select the **Database Operation Type**—Statement or Stored Procedure.

        * If you selected **Statement**, click **Add**, and then do the following:
            * Enter parameters to be passed in when calling the operation:
                * **Parameter Name**—Name of the parameter. No white space is allowed.
                * **Parameter Description**—Description of the parameter
                * **Test Value**—Value used for testing the operation. The value is stored for reuse.
            * In the **SQL Statement** field, enter a single SQL statement to be executed. You can use substitution parameters in the SQL statement to pass runtime values. Substitution parameter formatting is <%ParamKey%>, where <% and %> denote a substitution parameter and ParamKey is the name of the parameter to be used for substitution. For example, <%TopNode%>.
            * Click to test the operation. The Rollback option rollbacks any changes made to the database by the script. Rollback is selected by default. When an operation is tested, the parameter's test values are inserted into the statement and executed. Click the **Result** tab to view the results of the test.
        * If you selected **Stored Procedure**:
            * Enter the **Stored Procedure Name** to execute, may include package name as prefix.
            * Enter **Name** for the operation. The name must be unique for the parent External Connection.
            * Enter **Description** text describing the purpose of the operation.
            * View the list of parameters for the stored procedure. Select True for Results Param to return the parameter in Data Relationship Management operation result. Only one parameter may be selected as a result parameter. Result parameters are only returned for Lookup operations. For Commit operations, success or failure only is indicated.
            * **Test Value**—Value used for testing the operation. The value is stored for reuse.
            * Click to test the operation. The Rollback option rollbacks any changes made to the database by the stored procedure. Rollback is selected by default. When an operation is tested, the parameter's test values are inserted into the stored procedure and executed. Click the **Result** tab to view the results of the test.
- If you selected **Web Service**:
    - Select the **Protocol**: HTTP or HTTPS.
    - Enter the **Hostname**
    - Enter the **Port**—If port 0 is specified, standard ports 80 and 443 are used for HTTP and HTTPS respectively
    - Select the **Authentication Type**—If set to Basic, then User ID and Password can be saved.
    - Enter **User ID** and **Password**.

- To add an external operation, click **Add** and then do the following:
  - Enter **Name** for the operation. The name must be unique for the parent External Connection.
  - Enter **Description** text describing the purpose of the operation.
  - Select the **Operation Type**—Lookup or Commit. This selection is used to filter the list of operations available for selection with the External Lookup and External Commit features.
  - On the **Request** tab, click **Add**, and then enter parameters to be passed in when calling the operation:
    - **Parameter Name**—Name of the parameter. No white space is allowed.
    - **Parameter Description**—Description of the parameter
    - **Test Value**—Value used for testing the operation. The value is stored for reuse.
  - From **HTTP Action** select GET, POST, PUT, or DELETE.

    > **Note**
    >
    > Only POST and PUT allow sending HTTP Body content.

  - Enter the HTTP **URI** for the Web service message.
  - Enter the raw content of the **HTTP Header**.
  - Enter the text content of the **HTTP Body**.
  - **Response** tab—Displays the full outgoing and incoming messages for the Web service operation. Parameters used in the outgoing message will have their test values inserted into the request. The HTTP body of the incoming message returned by the Web service is expected to be in XML or JSON format. For external lookup operations, the incoming message needs to be converted to a tabular format (rows and columns) for use with external lookup properties. To handle this conversion, XPath expressions can be used. The List Identifier Expression parameter identifies the elements in the incoming message which are the rows of the result set. The Result Columns identify the attributes of the row elements which are displayed as columns in the result set.

    To preview the results of the List Identifier Expression and Result Columns configurations, click the **Preview** tab. The results are displayed in a data grid.

  You can use substitution parameters in the URI, HTTP Header, and HTTP Body to pass runtime values to the external operation. Substitution parameter formatting is <%ParamKey%>, where <% and %> denote a substitution parameter and ParamKey is the name of the parameter to be used for substitution. For example, <%TopNode%>.

  To test the configuration, click . The HTTP Request is built and sent to the endpoint. The user-interface automatically switches to the **Response** tab and displays the full outgoing message and incoming response. Parameters used in the outgoing message will have their test values inserted into the request.

7. Click to validate the selected items to verify that they are accessible at the appropriate level through the connection user name and password.

8. Click to save the external connection.

Editing External Connections

To edit an external connection:

1. On the Home page, select **Administer**.
2. Under **Metadata**, expand **External Connections**.
3. Select an external connection and click .
4. Make changes as required.
5. Click to save the external connection.

Deleting External Connections

When you delete an external connection, all import and export profiles using the connection are affected.

To delete an external connections:

1. From the Home page, select **Administer**.
2. Under **Metadata**, expand **External Connections**.
3. Select an external connection and click .
4. Select **Delete this Item** to confirm the deletion.

16 Configuring Governance Workflows

Governance workflows are formalized processes used to control the entry, approval, validation, and commitment of changes to nodes, relationships, and property values.

Application administrators define workflow tasks and workflow models to govern change requests submitted by business users and remediation requests submitted by data stewards.

It is recommended that you read "Governance Workflows" in the *Oracle Data Relationship Management User's Guide* for additional information on governance workflow concepts.

Managing Workflow Tasks

A workflow task is a single set of changes performed by a user for a local node within the context of a request. Request items in requests are controlled by workflow tasks.

A workflow task consists of an action type, instructions for the user, properties to be viewed or edited, and validations. The action type for a workflow task specifies the basic type of action being performed, such as adding, moving, or updating nodes. Each action type defines rules regarding the selection of nodes and parents, application of property updates, and the actions to perform when the request is validated and committed.

Note

The following actions are not supported in workflow requests:

- Merging nodes
- Annulling nodes
- Reactivating nodes
- Inserting orphan nodes
- Adding domain nodes where the domain is different from the parent

Task Properties

Workflow task properties can be configured to control which properties are displayed for request items, whether they are editable, and if values are required. Editable properties may be configured as required. Default properties for an action type may not be removed from the task.

Task and Property Instructions

You can add instructions on the request page to help guide users through the creation, enrichment, and approval of a request item. Instructions can be defined for workflow tasks and their properties. Task instructions are displayed for the originating task of a request item when an item is viewed in a Submit, Enrich, or Commit stage. Task instructions for an Update workflow task assigned to an Approve or Enrich stage are displayed instead of the originating task instructions. Task property instructions can be displayed for individual request item properties.

Hyperlinks can be included in task and property instructions. URLs can be inserted directly into the instructions field or the URL can use the syntax `[url=http_URL]URL_Title[/url]` where `http_URL` specifies the hyperlink text and `URL_Title` specifies the text displayed to the user. For example, this example: [url=http://support.oracle.com]Oracle Support[/url] would render in the property grid as Oracle Support.

Task Validations

Task validations are optional, node-level validations which must be successfully executed for request items before a request can be submitted or approved for a particular workflow stage. Validations configured to run in batch mode are available for selection as task validations. Task validations can be associated with task properties in order to link the validation messages with specific properties which may need to be corrected.

Calculated Name and Parent Properties

The Name and Parent properties used in workflow tasks identify the node and hierarchy location for which changes are being made. The values of these properties are often manually defined by a user or loaded from a source file. The Calculate Name and Calculate Parent options available for workflow tasks can be used to calculate the values of these properties using a dynamic script instead of having to define or load the values explicitly.

The Calculate Name option is available for workflow tasks using the Add Leaf or Add Limb action types. The Calculate Parent option is available for those tasks as well as Insert and Move tasks. The calculation logic of the script(s) may access the following data sources:

- NextID function
- Properties of the version for the request
- Hierarchies and their properties in the version
- Nodes and their properties
- Hierarchy relationships between nodes
- Properties of the request
- Request items and their properties
- Request item tasks and their action type

Calculation of the Name and Parent properties takes place when a governance request is calculated in the stage where a request item is added using a workflow task with these options enabled. The values may be recalculated in the originating stage for a request item or in a later stage which has been configured to recalculate these properties.

Note

If a workflow model has been set up to allow Recalculated Task Properties and the calculated Name or Parent is manually overridden, then the Name or Parent will not be calculated again during that stage or any subsequent stage.

External Commits

External commits can be optionally configured on workflow tasks to immediately synchronize approved changes in a governance request to an external target system when the request is committed. For example, an external operation can run an SQL statement that inserts, updates, or deletes data, or it can invoke a SOAP or REST Web service to create, update, or delete data in an external system. When you use external commits in Oracle Data Relationship Governance, external data updates can be initiated after a Data Relationship Governance request is successfully committed. The external data source is accessed using external operations defined for database and Web service connections.

After a Data Relationship Governance request has been successfully committed, external operations for each item are performed as configured by each item's task.

- Operations are performed synchronously in the order defined, by item and task.
- Operations are performed in the context of the local node for the request item, allowing output parameters to be based on properties which may not be selected for the task.
- If an error occurs during an external operation, the error message is added to the request item as an External Commit Failure.
- Request activity is updated with success or failure after each external operation.
- If a Commit Status property is defined for the external operation, then that property will be updated to True if the operation completed with no errors, and updated to False if the operation completed with errors.
- If any external operations did not complete successfully, then Data Managers and Commit stage participants are notified.

Creating Workflow Tasks

To create a workflow task:

1. On the Home page, select **Administer**.
2. From **New**, select **Workflow Task**.
3. Enter a name for the workflow task.
4. From **Action Type**, select the type of action for the task:
   - **Add Leaf**—Adds a leaf node with global and local properties
   - **Add Limb**—Adds a limb node with global and local properties
   - **Delete**—Updates a node's global/local properties and deletes the node
   - **Inactivate**–Updates a node's global and local properties and inactivates the node
   - **Insert**—Inserts a node into a hierarchy and updates its global/local properties
   - **Move**—Moves a node to a different parent and updates its global/local properties
   - **Remove**—Updates a node's global/local properties and removes the node
   - **Update**—Updates global and local properties for a node

> **Note**
>
> If users intend to upload items to a request from a file, the following properties are required to be defined in the task (and the files to be uploaded by the users):
>
> – For Add actions: Name, Parent, Description
> – For Insert actions: Name, Parent
> – For Move actions: Name, Parent

- **Reactivate** -- Updates a node's global and local properties and re-activates an inactive node.

5. **Optional**: Do any of these tasks:
   - Enter text for users in the **Instructions** field.

     URLs can be inserted directly into the instructions field or the URL can use the syntax `[url=http_URL]URL_Title[/url]` where `http_URL` specifies the hyperlink text and `URL_Title` specifies the text displayed to the user. For example, this example: [url=http://support.oracle.com]Oracle Support[/url] would render in the property grid as Oracle Support.
   - Select a Hierarchy Group on which to filter.

     > **Note**
     >
     > The hierarchy group selected for the workflow task is used with the hierarchy group property configured for the workflow model to filter hierarchies available for selection for the task.
   - Select a **Domain** for the node for Add Limb or Add Leaf tasks.

     > **Note**
     >
     > The domain configured for a workflow task must match a domain used by the target version for request items using the task. If the domain for the task is not used by the version, the request item node cannot be added to the version.

     > **Note**
     >
     > If a domain is assigned, then the Description property for Add Limb and Add Leaf tasks is required.
6. On the **Properties** tab, select properties from the **Available** list to assign to the task. Use the arrows to move properties to the **Selected** list. Use the up and down arrows to order the properties.
7. Click for a property to update these options:
   - **Editable**—Select to allow the property to be edited.

- **Required**—Select to make the property required.
- **Calculate**—For Add Limb or Add Leaf tasks, select to calculate the Name value from a dynamic script. If selected, then the Editable option for the Name property is False and disabled. When you select this option, the **Calculate Name** tab becomes available and then you can enter the script for calculating the Name value.

  To calculate the Parent value from a dynamic script for Add Limb, Add Leaf, Move, and Insert tasks, click next to a parent node and then select **Calculate**. If selected, then the Editable option for the Parent property is False and disabled. When you select this option, the **Calculate Parent** tab becomes available and then you can enter the script for calculating the Parent value. For information on writing dynamic scripts, see Managing Dynamic Scripts.
- **Custom Label**—**Optional:** Enter an alternate label for the property. This label displays in the property label column on item details.
- **Property Instructions**—**Optional:** Enter specific instructions for the property. The property does not have to be editable to add instructions. Instructions display above the property value in the item details.

  URLs can be inserted directly into the instructions field or the URL can use the syntax `[url=http_URL]URL_Title[/url]` where `http_URL` specifies the hyperlink text and `URL_Title` specifies the text displayed to the user. For example, this example: [url=http://support.oracle.com]Oracle Support[/url] would render in the property grid as Oracle Support.

Click to save changes or to cancel changes.

8. On the **Validations** tab, select validations from the **Available** list to assign to the task. Use the arrows to move validations to the **Selected** list.
9. Click associate validations with specific task properties. If the selected validation fails, the validation message will be displayed for the specified properties.

   Click to save changes or to cancel changes.
10. If you selected to calculate the name or parent, select the **Calculate Name** or **Calculate Parent** tab and then do the following:
    - Enter a dynamic script to calculate the name or parent. For information on writing dynamic scripts, see Creating Dynamic Scripts.
    - Enter the following information:
      – **Request ID**—Specifies the request ID to use when evaluating the script.
      – **Request Item Number**—Specifies the request item number to use when evaluating the script.
      – **Script Timeout**—The number of seconds until the script times out.
    - **Optional:** Select **Hidden** to specify the hidden property for the name or parent that you are calculating. If selected, then the calculated name or parent is not displayed in the request item details.
    - Click **Evaluate**. The results are displayed at the bottom of the script designer.
11. **Optional:** Select the **External Commit** tab, click **Add**, and then configure the following settings:
    - **External connection**—Select the external connection
    - **Operation**—Select the external operation to perform

> **Note**
>
> The operation must have been defined as a Commit type operation in the connection.

- For each external operation parameter configure:
  - **Parameter source type**—Select Literal or Property
  - **Source**—If **Literal** was selected for source type, then enter a literal value in the Param Source column. When the external operation is called, the literal value is passed in for the current parameters. If **Property** was selected for source type, then select a property to provide the parameter value for the external operation. When the External Commit is executed, the parameter value comes from the selected property on the current node or request item.
- **Commit Status property**—Select a Boolean property to indicate if the node had any external commit errors. This property is set for the node in the target version for the request. In the event of external commit failure, this property can be used to identify changes in the version which were not committed successfully to the external system.

12. Click to save the workflow task.

Editing Workflow Tasks

The list of properties and validations for a workflow task can be edited after the task has been created. The action type for a workflow task cannot be modified after the task is saved.

Request item properties for existing requests will be affected when task properties are added, removed, changed from editable to read-only, or reordered for a workflow task. Task properties which are removed will no longer be displayed for request items using the task. Property values defined for request items using task properties changed from editable to read-only will be discarded.

To edit a workflow task:

1. On the Home page, select **Administer**.
2. Under **Workflow**, expand **Workflow Tasks**.
3. Select a task, and then click .
4. On the **Properties** and **Validations** tabs, make changes to property and validations selections.
5. Click .

Copying Workflow Tasks

You can create a workflow task by copying an existing task. The action type, properties, and validations are copied and can be edited before saving.

To copy a workflow task:

1. On the Home page, select **Administer**.
2. Under **Workflow**, expand **Workflow Tasks**.
3. Right-click the task that you want to copy and select **Copy**.

4. Enter a new name for the task.
5. Make any other changes to the task and then click to save the workflow task.

Deleting Workflow Tasks

A workflow task may be deleted if it is not assigned to any model which is assigned to a change request. If a task is assigned to a model that cannot be deleted, then the task cannot be deleted.

To delete a workflow task:

1. On the Home page, select **Administer**.
2. Under **Workflow**, expand **Workflow Tasks**.
3. Select a workflow task, and then click .
4. Click **Delete this Workflow Task** to confirm the deletion.

Managing Workflow Models

A workflow model defines a set of change management tasks of defined types that can be included together in a single request, based on that model. The model defines the set of approvals and enrichment steps required before the changes can be committed to a version.

Workflow Stages

Workflow stages are defined for each workflow model and cannot be shared across workflow models.

Stage Types

When a stage is assigned to a workflow model, the stage type attribute defines the type of participation for users in that stage of the workflow.

Table 16-1 Workflow Stage Types

Workflow Stage Type	Description	Action Types
Submit	The Submit stage is used to define the initial request items to be included in a request. Multiple workflow tasks may be associated with this stage type. At least one request item must be added to a request during the Submit stage. AddLeaf or Add Limb tasks can be optionally configured with dependent workflow tasks. The system adds a request item for the original workflow task as well as an additional request item for each dependent task. A primary task cannot also be a dependent task. The primary and dependent tasks are considered a related group when you are calculating the name of the add item on the primary task. If you delete a primary task while the name calculation is still pending, non-Add dependent tasks are also deleted. **Note:** Each request has only one Submit stage. You cannot define workflow stage criteria for this stage.	• Add Limb • Add Leaf • Update • Inactivate • Insert • Move • Remove • Delete
Enrich	The Enrich stage is used to update request items that were added in the Submit stage or add request items. You can define workflow stage criteria for this stage. An Enrich stage has a single workflow task associated with it. A typical Enrich stage uses a workflow task with an Update action for the existing request items. However, some Enrich stages may require that additional line items be created, for example: • The insertion of a single node into multiple hierarchies • Update the local properties of a single node in several hierarchies This stage occurs between the Submit and Commit stages. **Note:** Any number of Enrich stages may be defined for a workflow model.	• Update (existing request items) • Insert (add new items) • Move (add new items) • All action types available for the Submit stage

Table 16-1 (Cont.) Workflow Stage Types

Workflow Stage Type	Description	Action Types
Approve	The Approve stage is used to view and approve all request items that were added in the Submit stage or added or updated during an Enrich stage. Users cannot add or edit request items during an Approve stage. You can define workflow stage criteria for this stage. An Approve stage uses a single workflow task to view properties and run validations for request items while the request is in the stage. Update tasks are available for use in Approve stages in a read-only mode. To update properties of request items in an intermediate stage, use an Enrich stage type instead. This stage occurs between the Submit and Commit stages. **Note:** Any number of approval stages may be defined for a workflow model.	Update (existing request items)
Commit	The Commit stage is used to provide a final approval of the request to trigger the commit of the request items in a request to a target versions. A committing user must approve all request items in a request. You can define workflow stage criteria for this stage but a request cannot be split at this stage. A Commit stage does not have a workflow task associated with it. Instead, the commit stage displays the superset of properties and runs the superset of validations available for the request items for previous Submit and Enrich stages. Users in the Commit stage can make updates to any editable properties displayed for request items to allow for final adjustments. This is the final workflow stage. **Note:** Each request has only one Commit stage.	N/A

Stage Conditions

Stage conditions can be used to alter the workflow path of a particular request based on specified criteria evaluated for the items in the request. You set up a condition for the stage and select what action should be taken if the condition is met, for example whether a request can enter the stage or if some request items are split off into a separate request. A workflow stage condition can be evaluated based on these criteria:

- **Property Criteria**—Use property query operators and literal values to evaluate as stage criteria for the stage.

- **Selected Validations**—Select one or more validations to run as stage criteria for the stage. You can select this option for an Approve, Enrich, or Commit stage.
- **Task Validations**—Failures of validations assigned to the workflow tasks. When selected, validations assigned to the task are also run as stage criteria for the stage. You can select this option for an Approve or Enrich stage. This option is not available if the task assigned to the stage does not have any validations assigned to it.

If any of the request items meet the stage condition for a workflow stage, then one of these actions can be taken:

- **Enter Stage**—For Approve, Enrich, or Commit stages, the request is assigned to users in the stage. The request enters the stage and workflow processing continues for that stage.
- **Split Request Items**—For Approve or Enrich stages, request items that meet the stage condition are moved into a separate, submitted request using the same workflow model. The new request enters the workflow stage and is assigned to users in the stage. Items not meeting the stage condition remain in the original request and the stage is skipped for the original request. If all request items meet stage criteria, the request is not split and the Split stage is entered.

If the request items do not meet the stage condition for a workflow stage, the stage is skipped and the request moves to the next stage in the workflow model.

Approval Methods

You select which users must approve a stage in a request:

- **Any Group**—Any user from an assigned node access group may approve the request in order to advance it to the next workflow stage. The node access group must be assigned to the hierarchy with access to the current stage type or greater. If none of the assigned access groups to the stage have proper data access to the request items in the request, the stage may be skipped as long as required values are provided and validations pass for all request items.
- **All Groups**—At least one user from all assigned node access groups must approve the request before it advances to the next stage. If none of the assigned access groups to the stage have proper data access to the request items in the request, the request is escalated to Data Managers for resolution.

Reapproval

If a request is pushed back to a previous stage and the request items are modified while pushed back, the changes to the request may require reapproval by users who have already provided their initial approval for the original request. This option determines whether changes made in each stage while in pushback mode are required to be reapproved by other users. Select one of the following options:

- **Current**—Changes to the request in this stage must be reapproved for the current stage only. After approval, the request is assigned to the user who previously pushed back the request.
- **All**—Changes to the request in this stage must be reapproved for subsequent stages.

Separation of Duties

Workflow stages can be optionally configured to require a separate approving user who has not submitted or approved for any other stage in the request. When the Separation of Duties option is enabled, a user who has submitted or approved for another workflow stage may not claim the request in the stage where the option is enabled. Note the following exceptions:

- The submitter may claim a request pushed back to the Submit stage.
- Prior approvers for the stage may claim a request pushed back to an Approve or Enrich stage.
- Data Manager role users may claim any request assigned to them regardless of prior approval.

Notifications

Notifications include both Web client alerts and e-mail notifications. You can set up if and when alerts and notifications are sent to workflow users for a workflow stage. Notifications are filtered to specific users based on the Notify setting for the stage and the type of workflow event that triggered the notification.

Note

Users do not receive notifications for actions they perform.

Select from these Notify options for each stage:

- **None**—No users are notified of actions performed for this workflow stage.
- **Assignees**—Users who belong to any workflow node access group currently assigned to the request are notified when Assign, Approve, Commit, or Reject actions occur.

  Assignees are only notified if they are members of a workflow access group assigned to the stage with a Notify setting of either Assignees or Assignees and Participants.
- **Participants**
  - When Commit or Reject actions occur, users who have submitted or claimed the request are notified.
  - When Approve or Promote actions occur, users who have submitted the request are notified.

  Participants are only notified if they are members of a workflow node access group assigned to the stage with a Notify setting of either Participants or Assignees and Participants.
- **Assignees and Participants**—Assignees and participants are notified.

The following table lists actions that trigger notifications and the recipients of the notifications based on the Notify setting of each stage.

Table 16-2 Workflow Alerts

Workflow Action	Notifications Sent To			
	Assignees	Submitter	Participants	Notify Users
Assign	X			
Approve	X	X		X
Promote		X		X
Escalate	X			X
Reject	X		X	X
Commit	X		X	X

Note

Notify Users are users who are members of a workflow node access group assigned to a stage with only Notify access to request items. They are only notified if the Notify setting is either Assignees or Assignees and Participants. If the Notify option is None or Participants, then these users are not notified

Dependent Workflow Tasks

Dependent workflow tasks can be used to automatically perform a workflow task in a governance request when another task is being performed. For example, when a node is being added, the node can also be inserted into other hierarchies in order to ensure synchronization across all hierarchies when the request is committed. Dependent tasks can be configured for primary workflow tasks using an Add Leaf and Add Limb action type.

When a request item is added to a request, the selected task for the item is the primary task. If the primary task is configured with dependent tasks, additional request items will be automatically added to the request for each dependent task.

Model Filters

You can restrict the versions, hierarchies, and node types that users can view and select for a particular type of request.

- **Version Variable**—Limits the selection of a version for request items in a request of a particular workflow model.
- **Hierarchy Group Property**—Limits the hierarchies from which nodes can be selected for request items in a request for a particular workflow model.
- **Hierarchy Group**—Required if a Hierarchy Group Property is specified.
- **Node Types**—Limits the nodes that can be added as request items to a request of a particular workflow model.

Request and Claim Duration

The workflow model for a request may be configured with a request or claim duration interval to control automatic handling of the request by a governance workflow based on an estimated amount of time expected for a particular type of request.

- **Request Duration**—Indicates the expected number of days that a request should take to be approved and committed. After the age of a request exceeds the request durations, the request is marked as Overdue.
- **Claim Duration**—Indicates the expected number of days that a request should be claimed for a workflow stage by a governance user. After the age of a request exceeds the claim duration, the request is automatically unclaimed to make it available for other assigned users to claim.

Note

A value of zero for either option indicates that the Overdue and automatically Unclaimed functionality is disabled for the workflow model.

Creating Workflow Models

To create a workflow model:

1. On the Home page, select **Administer**.
2. From **New**, select **Workflow Model**.
3. Enter a name, label, and description for the workflow model.

   The name is the unique name for the workflow model. The label is a user-friendly label for the workflow model and can be the same as the name. The description is optional.

   URLs can be inserted directly into the description field or the URL can use the syntax `[url=http_URL]URL_Title[/url]` where `http_URL` specifies the hyperlink text and `URL_Title` specifies the text displayed to the user. For example, this example: [url=http://support.oracle.com]Oracle Support[/url] would render in the property grid as Oracle Support.
4. **Optional:** Enter the number of days for **Request Duration** and **Claim Duration**
5. On the **Workflow Stages** tab, double click a stage (Submit or Commit) or click **Add Stage**.
6. On the **Stage** tab, configure the following options. See Workflow Stages for additional information on these options.
   - **Label**—Enter a label for the stage. The stage label can be edited at any time even after requests exist for the model.
   - **Type**—Select the stage type. The stage type can be edited until requests exist for the model; then it cannot be changed.
   - **Workflow Method**—Specify which node access groups must approve a stage in a request.
   - **Re-Approval**—Specify whether changes made only in the current stage or in all stages require reapproval.
   - **Notify**—Specify to whom notification and alerts are sent.
   - **Separation of Duties**—Select to require a separate approving user who has not submitted or approved for any other stage in the request.
   - **Recalculate Task Properties**—Select for use with external lookup properties or to allow a calculated name or parent value to be recalculated. This option is required when data is input in a later workflow stage which is used to calculate the final Name or Parent for a request item.

   Note

   If a workflow model has been set up to allow Recalculated Task Properties and the calculated Name or Parent is manually overridden, then the Name or Parent will not be calculated again during that stage or any subsequent stage.
7. For Submit stage tasks only, on the **Tasks** tab, configure tasks for the stage:
   - Select tasks to assign to the stage using the left and right arrow buttons
   - Position the tasks in the desired order using the up and down arrow buttons.

- If a task is a dependent task, you need to set the primary task that it is dependent on. For the dependent task, click and from the Primary Task drop-down list, select the primary task.

  > **Note**
  >
  > Only Add Limb or Add Leaf tasks can be set as primary tasks. Primary tasks cannot be hidden and cannot also be dependent tasks.

- **Hidden**—If selected for a dependent task, then the task does not display in the Add Items dialog within requests.

  > **Note**
  >
  > Selected tasks are editable until requests exist for the model, then they cannot be changed.

8. On the **Node Access Groups** tab, select workflow node access groups to be associated with the workflow stage.

   Only node access groups of the Workflow type can be assigned to a stage.
9. **Optional:** To add criteria for a workflow stage, on the **Condition** tab, select the type of condition, select the action to perform, and then click :
   - **Type**
     – **Property Criteria**—Select one or more properties to evaluate as stage criteria for the stage. Click **Add** to insert a criteria row. Select a **Property** and **Operator** for the row, and enter a **Value**.
     – **Selected Validations**—Select one or more validations to run as stage criteria for the stage. Click the arrow to move validations to the **Selected** list.
     – **Task Validations**—Select to run validations assigned to the task as stage criteria.
   - **Action**—Select an action to perform (Enter Stage or Split Request Items) for the workflow stage when stage criteria are met. See Stage Conditions for more information.
10. Click to save the workflow stage.
11. **Optional**: On the **Filters** tab, make selections to restrict the versions, hierarchies, and node types that users can view and select for a particular type of request.
12. **Optional**: Click **Add Stage** to add Enrich or Approve stages to the workflow model, and then follow steps 6-8 for each stage added.
13. Click to save the workflow model.

Editing Workflow Models

Workflow models for which requests have been created are restricted from certain edits in order to ensure existing requests are not negatively affected during workflow processing and their content is not altered after the requests have completed. For models that have change requests these editing restrictions apply:

- Workflow stages for the model may not be added, removed or reordered.
- The stage type for a stage may not be changed.
- The task for a workflow stage on the model may not be changed.

To edit a workflow model:

1. On the Home page, select **Administer**.
2. Under **Workflow**, expand **Workflow Models**.
3. Select a model and then click .
4. Make changes to the workflow model and click .

Copying Workflow Models

You can create a workflow model by copying an existing model. All workflow stages, model filters, and duration settings are copied and can be edited before saving. In situations where an existing workflow model being used for current requests needs to be edited to handle future requests differently, the model can be copied and the changes made to the new model. The edited copy of the model can then be used for newly created requests.

To copy a workflow model:

1. On the Home page, select **Administer**.
2. Under **Workflow**, expand **Workflow Models**.
3. Select the model that you want to copy and then click .
4. Enter a new name for the model.
5. Make any other changes to the model and then click to save the workflow model.

Renaming Workflow Models

To support different workflow requirements over time, workflow models may be copied to apply edits to their configuration. In these cases, the model copy can be renamed to match the name of the original workflow model with which governance users are already familiar.

To rename a workflow model:

1. On the Home page, select **Administer**.
2. Under **Workflow**, expand **Workflow Models**.
3. Select the model that you want to rename and then click .
4. Enter a new name for the model and then click .

Hiding Workflow Models

Workflow models can be hidden to prevent users from creating new requests using those models. Existing requests created prior to a workflow model being hidden will continue through the model to completion. When a workflow model is copied and modified in order to replace the original model, the original model can be hidden so that only one instance of the model is available for new requests.

Note

Requests using the workflow model that you choose to hide will continue their process flow to completion.

To hide a workflow model:

1. On the Home page, select **Administer**.
2. Under **Workflow**, expand **Workflow Models**.
3. Select the model that you want to hide and then click .
4. Select **Hidden** and then click .

Deleting Workflow Models

A workflow model may be deleted only if there are no requests associated with it (including in-flight or historical requests). Completed requests are retained until the version for the request is deleted, requiring that the workflow model also be available in order to view the requests.

Tip

Consider the information in Hiding Workflow Models to determine if this may be a more appropriate option.

To delete a workflow model:

1. On the Home page, select **Administer**.
2. Under **Workflow**, expand **Workflow Models**.
3. Select a model, and then click .
4. Click **Delete this Workflow Model** to confirm the deletion.

17

Managing Data Relationship Management Analytics

Oracle Data Relationship Management Analytics provides dashboards for change tracking, growth analysis, request monitoring, workflow model performance, and participant and user group performance. The Data Relationship Management Analytics Dashboards are:

- **Change**—Provides aggregated views of changes that have occurred in the Oracle Data Relationship Management system over time. Metrics in this dashboard are based on committed requests and all interactive changes. This dashboard includes change actions such as adds, updates, moves, and deletes across node and property changes to lend change perspectives by hierarchy, node type, property category, and so on. Users can understand change trends by change method, interactive, or workflow to ratify governance uptake. Users can drill into each change contextually to inspect transaction details and export these details to a flat file for further analysis offline.
- **Growth**—Provides analysis of how versions and hierarchies have changed over time by displaying the number of orphan and shared nodes, the total number of nodes, and the total increase or decrease in nodes from a previous version (for lineaged versions) and the total increase or decrease in the last 30 days for non-lineaged versions.
- **Requests**—Displays key performance indicators as they relate to open Oracle Data Relationship Governance requests allowing you to identify bottlenecks and requests that are overdue or near due, and provides the ability drill-back into Data Relationship Governance requests to make changes to a request.
- **Model**—Provides analysis of Data Relationship Governance workflow model design by displaying historical performance of requests that are completed (committed or rejected), including participant behavior trends, resource workload, and the ability to drill-back into Data Relationship Governance requests. Workflow model analysis reports on performance of completed requests processed by each workflow model to understand model performance based on service level agreements, level of automation achieved, cycle time, resources committed, request workload, throughput, and participant engagement.
- **Reports**—Used to view user and group membership, security, and activity. Information provided includes user role assignments, access group assignment reports, and user login activity.
  - **User Role Assignment Report**—Provides a list of users by role or roles by user with counts by licensed user types.
  - **Access Group Membership Report**—Provides a list of users by interactive and workflow user groups.
  - **Object Access Group Authorization Report**—Provides mapping of users and user groups to specific Data Relationship Management objects.
  - **Hierarchy Access Group Assignment Report**—Provides data grants of users and groups to nodes in a hierarchy.
  - **Workflow Access Group Assignment Report**—Provides data grants of users and groups to workflow model stages.
  - **User Login Activity Report**—Provides trend reports for user login activity over time.

  - **Metadata Object Usage Report**—Provides frequency distribution and aging information for Data Relationship Management objects: queries, compares, imports, exports, blenders, and books.

Accessing Data Relationship Analytics

Before configuring Oracle Data Relationship Management Analytics ensure the following tasks have been completed:

- Set up Analytics URL—Provides the link to Data Relationship Management Analytics from Oracle Data Relationship Management. See "Configuring Analytics URL" in *Oracle Data Relationship Management Installation Guide*.
- Set up Web farm—Enables drillback from Data Relationship Management Analytics to Data Relationship Management. See "Configuring Web Servers" in *Oracle Data Relationship Management Installation Guide*.
- Version lineage has been set up—Version lineage allows Data Relationship Management Analytics to aggregate changes across lineages and across multiple versions. See "Editing Version Properties" in *Oracle Data Relationship Management User's Guide*.
- Set up in Data Relationship Management when hierarchy and version node counts are updated. Node counts are updated when a version is opened, saved, or closed and as specified in a system preference. See AnalyticsNodeCountUpdateTime in System Preferences.
- Set hierarchy group property to the default Core property type. Only the default Core property type is supported in Data Relationship Management Analytics. See Step 6 of Creating Properties.

In Data Relationship Management, click the Analytics link.

Note

The Analytics link is available only if the user is assigned to any of these roles: Analytics User, Governance Manager, Access Manager, Data Manager, Application Administrator.

Working with Preferences

Before creating execution plans, preferences need to be configured.

To set preferences:

1. Click 📋.
2. **Optional:** Do the following:
   - **Batch Size**—Enter a batch size value. Used for model analysis. The default value is 250 MB and should not be changed unless absolutely necessary. The larger the batch size the larger the memory and database requirements.
   - **Initial Extract Date**—Set the date from which data will be extracted for all Oracle Data Relationship Management Analytics tasks.

3. Click **Save**.

Working with Execution Plans

Predefined tasks extract the information from Oracle Data Relationship Management and return it to the specific Oracle Data Relationship Management Analytics dashboard where it can be filtered and reviewed. Jobs consist of dashboard-specific tasks. Multiple jobs can be included in an execution plan.

Execution plans consist of a schedule and one or more jobs and their tasks. Execution plans can be configured to run daily, weekly, or monthly and can be scheduled to run as Simple (run now or run at a future date/time) or as Cron (using a Cron expression to indicate scheduling information). Execution plans can be edited, inactivated when not in use, and deleted when no longer needed.

Table 17-1 Job Tasks

Jobs	Tasks
Change Analysis	Transaction Fact Table Transaction Aggregate Transaction Property Aggregate Version Lineage
User Activity Reports	Transaction Fact Table
Growth Analysis	Version Lineage Hierarchy Counts Version Counts
Model Analysis	Model Analysis

Creating Execution Plans

To create an execution plan:

1. In the Oracle Data Relationship Management Analytics dashboard, select **Settings**.
2. Click **Create** and enter the following information:
   - **Name**—Enter a name for the execution plan
   - **Schedule Type**—Select from the following options:
     – **Simple**—Use to specify a start and end date
     – **Cron**—Use to specify a Cron expression
   - **Scheduler Timeframe**—Select Run Now or Future.
3. Click **Next**.
4. Do the following:
   - If you selected **Simple** as the Schedule Type and **Run Now** for the Scheduler Timeframe, do the following:
     a. **Optional:** Select **Truncate and Load** to truncate any tables associated with this job and reload based on the initial extract date in the system. If not selected, then an incremental load runs.
     b. Click **OK** if you are sure you want to truncate and load.

- If you selected **Simple** as the Schedule Type and **Future** for the Scheduler Timeframe, do the following:
  a. Select the Frequency to run the execution plan: Daily, Weekly, or Monthly.
  b. Click to enter the start date and time.
  c. **Optional:** Click to enter the end date and time.
- If you selected **Cron** as the Schedule Type, enter a Cron expression for when the scheduler will run.

5. Click **Next**.
6. Select jobs to add to the execution plan. User the Move, Move All, Remove, and Remove All buttons to move jobs from the Available list to the Selected list.
7. Click **Next**.
8. Review the execution plan settings and then click **Schedule Plan**.

   Note

   For execution plans to run, the Scheduler must be started. To start the Scheduler, click and select **Start**.

9. Click **OK** to confirm scheduling plan.

Editing Execution Plans

When you edit an execution plan, all fields are editable except for the plan name.

To edit an execution plan:

1. Select the plan to edit.
2. Click and make changes to the plan by following steps 2-9 in Creating Execution Plans.

   Note

   You cannot change the plan name. If you need to change the plan name, delete the plan and create a new plan.

Inactivating and Reactivating Execution Plans

When an execution plan is inactivated, any future scheduled plans are removed from the scheduler and the plan is moved to the Inactive Plan tab. To reactivate the plan, on the Inactive Plans tab, edit the plan and then schedule it.

To inactivate an execution plan:

1. Select and then select the plan to inactivate.
2. Click next to the plan name.

To reactivate an execution plan:

1. Select and then select the plan to reactivate.
2. Click and make changes to the plan by following steps 2-9 in Creating Execution Plans.

> **Note**
>
> You cannot change the plan name. If you need to change the plan name, delete the plan and create a new plan.

Deleting Execution Plans

To delete an execution plan:

1. Select the execution plan that you want to delete.
2. Click next to the plan name.
3. Click **OK** to confirm the deletion.

Viewing Activity

In the Recent Activity section, you can view the results of execution plans that have been run. You can view the start and end time of the execution plan, the duration of the run, the number of records processed, and the status of the run. Note that if you schedule multiple jobs in the same plan and more than one job includes a task that another job has already run, the execution will skip the task in subsequent jobs and will show as Skipped Duplicate in the execution plan results.

To view results of execution plans that have been run:

1. Click or click .
2. Expand the execution plan that you want to view by clicking the arrow to the left of the plan name. You can expand jobs within the plan to review the associated tasks.
3. **Optional:** Click the filter bar and set filter options:
   - **Timeframe**—Enter the number days for which to show plan activity. For example, if you enter 2, then plan activity from the last 2 days is displayed.
   - **Name**—Select **All** or select execution plan names to include in the results.
   - **Status**—Select **All** or select execution plan statuses to include in the results. Plan statuses are Complete, Partial Failure, Failed, and Processing.

Note

The Status filter criteria only applies to execution plan status, not to job or task status.

18 Integrating External Workflow Applications

External workflow applications can be used to process proposed changes to Oracle Data Relationship Management from an external source. The Web Service API provides an external request interface which allows multiple changes to be grouped together for validation and commitment of the changes as a single unit of work during an external workflow process. API users must have the Workflow User role in order to participate with external requests. This request interface is generic and does not support the use of workflow models, workflow tasks, or the Worklist page in the Web Client. These generic, external requests are recorded in and only accessible from the Request History.

For more information on API support for external requests, see the "Oracle Data Relationship Management API Reference".

External Requests

You can create external requests to:

- Add hierarchies
- Add nodes
- Insert and move nodes
- Activate, inactivate and remove nodes
- Update properties
- Remove property values

External requests can be stored in a draft state for approval and validated against a Oracle Data Relationship Management version without committing the changes to the version immediately. External requests in this pending approval state can be updated by multiple users at different times and re-validated as needed. The transactions in a request are committed to a Data Relationship Management version when the request is approved.

Note

After an external request has been approved, the request cannot be modified and the request cannot be deleted until the associated version is deleted.

An external request consists of the following elements:

- Target Data Relationship Management version.
- Owner of the request — A valid Data Relationship Management user ID.
- Custom workflow ID — Identifier for the request in a workflow application.
- Custom workflow label — Short description for the request in a workflow application.
- Custom workflow status — Manages the status of the request in a workflow application.
- Custom workflow info — Stores extra information needed by a workflow application.

- Request comments — Annotation for the request.
- Created by — User who created the initial request.
- Created date — Date when the request was created.
- Updated by — User who last updated the request.
- Updated date — Date when the request was last updated.
- Approved By — User who approved the request.
- Approved Date — Date when the request was approved.
- Validated Flag — Indicates whether the request has been validated since it was last updated.
- Approved Flag — Indicates whether the request has been approved.
- Additional batch validations that should be applied to only the actions in the request during a validate or approve operation
- List of action items that affect hierarchies and nodes for the current request

19 Migrating Data Relationship Management Metadata

The Oracle Data Relationship Management Migration Utility provides application administrators the ability to move metadata object types between Data Relationship Management applications.

In the Migration Utility, you can:

- Extract metadata object types from a Data Relationship Management application to an XML file and generate an HTML report from the results
- Load metadata from an XML file into a Data Relationship Management application
- Compare metadata differences between two sources, create an XML file with the differences, and generate an HTML report from the results
- View metadata in an XML file and generate an HTML report from the file

You can extract, load, compare, and view the following types of metadata:

- Property Definitions
- Property Categories
- Validations
- Node Types
- Glyphs
- Node Access Groups
- Hierarchy Groups
- Queries (Standard, System, and Custom)
- Compares (Standard, System, and Custom)
- Domains
- Version Variables (Standard, System, and Custom)
- Exports (Standard, System, and Custom)
- Export Books (Standard, System, and Custom)
- Imports (Standard, System, and Custom)
- Blenders (Standard, System, and Custom)
- System Preferences
- External Connections (Standard, System, and Custom)

  External Connections display the connection name only; object access group name prefixes are not added.

Note

Connection string, user ID, and password do not migrate with migration loads and extracts.

- Object Access Groups
- Workflow Tasks
- Workflow Models

Migrating Core Property Configurations and Settings

The following core property configurations and settings can be migrated between instances of Data Relationship Management (on same release) using the Metadata Migration Utility:

- Core.DefaultDisplayBy [Default Display Properties]
- Core.DefaultPasteProps [Default Paste Properties]
- Core.DefaultSynchBy [Default Match By]
- Core.EnableSharedNodes [Enable Shared Nodes]
- Core.HierarchyNodeType [Hierarchy Node Type]
- Core.IDLengthLeafProp [ID Length Leaf Property]
- Core.IDLengthLimbProp [ID Length Limb Property]
- Core.PrefillLeafProp [Prefill Leaf Property]
- Core.PrefillLimbProp [Prefill Limb Property]
- Core.SortOrder [Sort Order]
- Core.StandardHierSort [Standard Hierarchy Sort]

Opening the Migration Utility

By default, the Migration Utility is installed to:

```
MIDDLEWARE_HOME\EPMSystem11R1\products\DataRelationshipManagement\client
```

To open the Migration Utility, double click **Data Relationship Management Migration Utility**.

Extracting Metadata

You can select the types of metadata to extract from a Oracle Data Relationship Management application. You extract the information into an XML file which you can then view, load into another Data Relationship Management application, compare to another XML file, or compare to another Data Relationship Management application. You can also use this file for backup, storage, and auditing purposes.

You can generate a report from the information in the XML file that is created.

To extract metadata from a Data Relationship Management application:

1. On the Main Menu, click **Extract**.
2. Enter Data Relationship Management connection information and click **Log In**.
3. Select the object types or objects to extract and click **Next**.

> **Note**
>
> Click the plus sign in the hierarchy tree to see objects. Select the checkbox for an object type to select the object type and all of its objects, or select the checkbox for the objects that you want to extract. Click on an object name to display the object type definition in a new window.

4. **Optional:** Click **Find** to search for a metadata object type or object.

   > **Note**
   >
   > Any object type containing the text entered is returned. To navigate to a particular object in the results, click the Jump To link.

5. Review the summary information.

   > **Note**
   >
   > The Migration Utility performs additional checks for object types that have dependencies. For example, an export may depend on property definitions or a property definition may reference another property definition. If there are dependencies missing in the summary, you may select specific dependencies to include. You can include all excluded dependencies or exclude all dependencies.

   > **Note**
   >
   > Increasing the page size allows you to define the number of object types to view on a page.

6. **Optional:** Enter metadata details for this extract.

   You can enter the following information:

   - **Title**—Maximum of 255 characters
   - **Purpose**—Formatted memo
   - **Usage**—Formatted memo
   - **Application Version**—Maximum of 20 characters
   - **File Version**—Maximum of 20 characters

7. Click **Run Extract**.
8. Do any of the following:
   - Click **Download the Metadata File** to open or save the XML file.
   - Click **View the Metadata File** to view the XML file details.
   - Click **Load the Metadata File** to load the XML file into a Data Relationship Management application. For more information, see Loading Metadata.
   - Click **Generate Reports for the Metadata File** to generate a report from the XML file. For more information, see Generating Reports.

Loading Metadata

Only files with the Oracle Data Relationship Management XML format can be loaded into a Data Relationship Management application. A log file is created after a load is performed and displays the following severities of data: audit, information, warning, and error message.

> **Note**
>
> Before loading a metadata file, it is recommended that you perform an extract of existing metadata in case you want to revert back to the previous configuration. It is also a good idea to perform a database backup before loading metadata, particularly if you are loading a migration file into a production environment.

To load metadata from an XML file into a Data Relationship Management application:

1. On the Main Menu, click **Load**.
2. Click **Browse**, select the XML file that you want to load, and click **Upload**.

   > **Note**
   >
   > Migration files must be UTF-8 encoded.

3. Review the uploaded file information and click **Next**.
4. Enter Data Relationship Management connection information and click **Log In**.
5. Select the object types or objects to load and click **Next**.

   > **Note**
   >
   > Click the plus sign in the hierarchy tree to see objects. Select the checkbox for an object type to select the object type and all of its objects, or select the checkbox for the objects that you want to load. Click on an object name to display the object type definition in a new window.

6. Review the summary information and click **Next**.

   > **Note**
   >
   > Page size allows you to define the number of object types to view on a page.

7. **Optional:** Select **Continue Load After Error** for the load to continue even if errors are encountered.
8. Click **Run Load**.
9. Review the load results.

   You can change the view of the log file by selecting the severity of detail to display: audit, information, warning, and error. To save the log file, click **Download**.

Note

The log items can be sorted by any column using the column header links.

Comparing Metadata

You can compare two metadata sources. You can compare metadata differences between two Oracle Data Relationship Management applications, between two XML files, or between a Data Relationship Management application and an XML file. You can generate an XML file containing the differences between the two metadata sources. The results can be used to restore data, undo unauthorized changes, or find wrong object type configurations.

You can generate a report from the information in the XML file that is created.

To compare metadata:

1. On the Main Menu, click **Difference**.
2. From the **Source #1** drop-down list, select the type of source: Server Connection or XML File.
3. Do one of the following:
   - If you selected **Server Connection**, enter Data Relationship Management connection information and click **Log In**.
   - If you selected **XML File**, click **Browse** and select the XML file that you want to use in the comparison and click **Upload**.
4. If you uploaded a file, review the uploaded file information and click **Next**. Otherwise, skip to the next step.
5. Repeat steps 2–4 for Source #2.
6. Click **Next**.
7. Select the object types to include in a difference file by using the following actions:
   - Select a filter
   - Click **>** to select a object type from Source #1.
   - Click **<** to select a object type from Source #2.
   - Click **X** to deselect a object type.
   - Click the left column header to select all objects from Source #1 based on the selected filter.
   - Click the right column header to select all objects from Source #2 based on the selected filter.
   - Click the center column header to deselect all objects based on selected filter.
   - Click the page links at the top of the compare results to switch to a different page.

   Note

   Page size allows you to define the number of object types to view on a page.

8. Click **Create Difference File**.

9. Do any of the following:
   - Click **Download the Metadata Difference File** to open or save the XML file.
   - Click **View the Metadata Difference File** to view the XML file details.
   - Click **Load the Metadata Difference File** to load the file into an Data Relationship Management application. For more information, see Loading Metadata.
   - Click **Generate Reports for the Metadata File** to generate a report from the XML file. For more information, see Generating Reports.

Viewing Metadata

You can view a metadata file and generate a report from the information in it.

To view metadata in an XML file:

1. On the Main Menu, click **View File**.
2. Click **Browse** and select the XML file that you want to view and click **Upload**.
3. Review the uploaded file information and click **Next**.
4. Click the plus signs in the hierarchy tree to view metadata objects.
5. **Optional:** Click **Find** to search for an item in the file.

   Note

   Any object type containing the text is returned. To navigate to a particular object in the results, click the Jump To link.

6. **Optional:** Click the **Reports** tab to generate an HTML report from the file.

Metadata File Restrictions

The default limit for uploaded files in the Migration Utility is 4 MB. When loading or viewing a large metadata file using the Migration Utility, the following error may occur if the size of the file exceeds the configured limit.

"Unexpected Error There was an unexpected error trying to process your request: Maximum request length exceeded."

For information on configuring a larger file size, see "Configuring Migration Utility" in the *Oracle Data Relationship Management Installation Guide*.

Generating Reports

You can generate an HTML report from an XML file generated after an extract, from a difference report, and from a metadata file that you are viewing.

To generate an HTML report:

1. Do one of the following:
   - After extracting metadata or creating a difference report, click **Generate Reports for the Metadata File**.

- After viewing a metadata file, click **Reports**.

2. Do one of the following:
   - Click **View Report** to display the report.
   - Click **Download Report** to save the report.