

Oracle® Hyperion Data Relationship Management, Fusion Edition

Oracle® Hyperion Data Relationship Steward

Oracle® Hyperion Data Relationship Management for Oracle Hyperion Enterprise Planning Suite

Oracle® Hyperion Data Relationship Management for Oracle Hyperion Financial Close Suite

Oracle® Hyperion Data Relationship Management for Customer Hub

Oracle® Hyperion Data Relationship Management Read Only Access

Administrator's Guide

RELEASE 11.1.2.1

Data Relationship Management Administrator's Guide, 11.1.2.1

Copyright © 1999, 2011, Oracle and/or its affiliates. All rights reserved.

Authors: EPM Information Development Team

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited. The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

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

U.S. GOVERNMENT RIGHTS:

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

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

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

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

Contents

Documentation Accessibility	7
Chapter 1. Release Overview	9
New Feature Terms	10
Changes to Existing Feature Terms	10
Obsolete Terms	11
Chapter 2. Managing Users	13
User Permissions	13
Version Permissions	14
Request Permissions	14
Query Permissions	15
Compare Permissions	15
Import Permissions	16
Blender Permissions	16
Export Permissions	17
Script Permissions	17
Audit Permissions	17
Application Permissions	18
Access Permissions	18
User Roles	18
Creating Users	22
User Authentication	23
Modifying Users	23
Changing Passwords	24
Locking Out Users	24
Unlocking Users	24
Changing User Roles and Assignments	25
Deleting Users	25
Viewing User Login Status	25
Chapter 3. Managing Node Access Groups	27
Creating Node Access Groups	28

Editing Node Access Groups	29
Deleting Node Access Groups	29
Assigning Node Access Group Security	29
Chapter 4. Managing Property Categories	31
Property Categories	31
Creating Property Categories	32
Editing Property Categories	32
Deleting Property Categories	33
Chapter 5. Managing Property Definitions	35
Creating Properties	36
Data Types	38
Working with Formulas	40
Creating Derived Properties with Formulas	40
Functions	42
Special Characters	43
Literals	43
Format String Parameter	43
Date-Time Format Strings	45
Function Definitions	46
Editing Property Definitions	77
Deleting Properties	78
Chapter 6. Managing Validations	79
Validation Classes	79
Validation Levels	81
Creating Validations	82
Assigning Validations	83
Editing Validations	83
Deleting Validations	83
Chapter 7. Managing Node Types	85
Defining Node Types	85
Editing Node Types	86
Deleting Node Types	86
Working with Node Glyphs	86
Chapter 8. Working with System Preferences	89
System Preferences	89
Setting Up Change Approval	94

Local Security for Global Properties	95
Configuring System Preferences	95
Chapter 9. Working with External Connections	97
Defining External Connections	97
Editing External Connections	98
Deleting External Connections	98
Chapter 10. Migrating Data Relationship Management Metadata	99
Opening the Migration Utility	100
Extracting Metadata	100
Loading Metadata	101
Comparing Metadata	102
Viewing Metadata	103
Generating Reports	104
Index	105

Documentation Accessibility

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

Access to Oracle Support for Hearing-Impaired Customers

Oracle customers have access to electronic support through My Oracle Support or by calling Oracle Support at 1.800.223.1711. Hearing-impaired customers in the U.S. who wish to speak to an Oracle Support representative may use a telecommunications relay service (TRS). Information about the TRS is available at <http://www.fcc.gov/cgb/consumerfacts/trs.html/>, and a list of telephone numbers is available at <http://www.fcc.gov/cgb/dro/trsphonebk.html>. International hearing-impaired customers should use the TRS at +1.605.224.1837. An Oracle Support engineer will respond to technical issues according to the standard service request process.



Release Overview

In This Chapter

New Feature Terms	10
Changes to Existing Feature Terms	10
Obsolete Terms	11

In the 11.1.2 release of Oracle Hyperion Data Relationship Management, Fusion Edition, a new user interface was introduced. This user interface replaces the Data Relationship Management Win32 client and the ActiveX Web Publishing client.

The new user interface:

- Provides one user experience for all Data Relationship Management user types
- Offers Web accessibility to all Data Relationship Management Win32 features except the Console
- Allows user-authenticated and anonymous access (previously provided by Web Publishing)
- Simplifies the presentation of data and access to features for less-technical users
- Provides a task-driven approach to user interaction vs. a tool-driven approach
- Minimizes any degradation in richness of features due to Web accessibility
- Supports internationalization of the product
- Eliminates the need for additional client installation to access the Data Relationship Management user interface
- Minimizes the need for training new users or retraining existing users

For more information on user interface differences, see:

- [New Feature Terms](#)
- [Changes to Existing Feature Terms](#)
- [Obsolete Terms](#)
- “User Experience Differences” in Oracle Hyperion Data Relationship Management User's Guide

New Feature Terms

Term	Definition
Application	An instance of Data Relationship Management. The user interface can access multiple applications.
As Sibling	An Add/Insert/Move/Put action allows placement of a node as a sibling to another node.
Clipboard	Copy nodes here to work with them across multiple hierarchy windows.
Connection	External locations relative to the server that can be read from and written to.
Object Access	User metadata objects can now be: <ul style="list-style-type: none"> ● User – Personal objects that are only available to an individual user to view and run. All user roles have the ability to create and manage objects of this access level. ● Standard – Public objects that are available to all users to view and run. Only Data Manager role users have the ability to create and manage objects of this access level. ● System – Restricted objects that are only available to Application Administrator role users to view and run. Only those users have the ability to create and manage objects of this access level.
Server File	A type of connection used to access a network file system or an FTP directory.
Shortcuts	Context-sensitive links that navigate the user to another task group retaining the current selection.
Use Fast Deletes	During an export, enables a bulk deletion on a target database table.
User Roles	Permission-based roles that control user access to product features. Roles: <ul style="list-style-type: none"> ● Anonymous User ● Workflow User ● Interactive User ● Data Creator ● Data Manager ● Access Manager ● Application Administrator
Version and Hierarchy Owner	The Owner has full management privileges to a version/hierarchy. A user with the permission to create a version or hierarchy is assigned as the Version Owner or Hierarchy Owner.

Changes to Existing Feature Terms

Term	Description of Change
Action Script	Formerly the Automator interface or an Automator script.
Controlled Property	For a hierarchy, formerly assigning a controlling hierarchy to a property
Download	Formerly Save to File or Print
Hierarchy Group	Formerly a System Category, which is used for grouping hierarchies

Term	Description of Change
Locate	Formerly the Synchronize By Name feature
Load Status	Formerly represented as bold-faced print for an open/closed version
Match	Formerly the Synchronize By Property feature
Paste Properties	Properties can be pasted for any node that is taken and copied to the clipboard.
Put	Extended to support reordering children in addition to Insert and Move
Search	Formerly the Find Node feature
Take/Copy Properties	Includes the former Take and Copy Properties features
Validation (Batch)	Formerly Verification (and Verify)

Obsolete Terms

Term	Comment
Abbrev	Replaced by Name. Except for the Abbrev() formula function.
Automator	Replaced by Action Script
Export Preview	Replaced by Download
System Category	Replaced by Hierarchy Group
User Types <ul style="list-style-type: none"> ● User ● Functional Administrator ● Security Administrator ● System Administrator 	Replaced by User Roles and Permissions
Verification (and Verify)	Replaced by Batch Validation (and Validate)
Web Publishing	Replaced by Anonymous Access

2

Managing Users

In This Chapter

User Permissions.....	13
User Roles	18
Creating Users.....	22
User Authentication	23
Modifying Users	23
Deleting Users.....	25
Viewing User Login Status.....	25

User Permissions

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

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. 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

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

Query Permissions

Permission Level 1	Permission Level 2	Permission Level 3
<p>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.</p>	<p>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.</p>	<p>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.</p>
	<p>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.</p>	NA

Compare Permissions

Permission Level 1	Permission Level 2	Permission Level 3
<p>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.</p>	<p>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.</p>	<p>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.</p>
	<p>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.</p>	NA

Import Permissions

Permission Level 1	Permission Level 2	Permission Level 3
<p>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.</p>	<p>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.</p>	<p>Run Import - Users can view and run any Standard import. Users have restricted access to Property selector based on Property Category security.</p>
	<p>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.</p>	<p>NA</p>

Blender Permissions

Permission Level 1	Permission Level 2	Permission Level 3
<p>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.</p>	<p>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.</p>	<p>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.</p>
	<p>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.</p>	<p>NA</p>

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

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

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

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

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

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

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

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:

Permissions			User Roles						
Level 1	Level 2	Level 3	Access Manager	Anonymous User	Application Administrator	Data Creator	Data Manager	Interactive User	Workflow User
Manage Versions							X		
	Browse Versions			X	X	X		X	X
	Create Versions					X			
	Manage Hierarchies						X		
		Browse Hierarchies		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		

Permissions			User Roles						
Level 1	Level 2	Level 3	Access Manager	Anonymous User	Application Administrator	Data Creator	Data Manager	Interactive User	Workflow User
Manage System Imports					X				
	Manage User Imports					X			
		Run Import							
	Manage Standard Imports						X		
Manage System Blenders					X				
	Manage User Blenders					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	

Permissions			User Roles						
Level 1	Level 2	Level 3	Access Manager	Anonymous User	Application Administrator	Data Creator	Data Manager	Interactive User	Workflow 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								
	Manage Preferences								
Manage Access									
	Manage Users		X						
	Manage Roles		X						
	Manage Access Groups		X						
	Manage Property Access		X						

Creating Users

When you create users, you define a unique name and assign one or more 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.

► 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 and Phone are optional.

4 **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.
- 5 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.
 - 6 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.
 - 7 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.
 - 8 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.
 - 9 Click .
- The Change Password dialog box is displayed.
- 10 Enter a password for the user.
 - 11 Re-enter the password.
 - 12 **Optional:** Select **User must change password at next login** to require the user to change their password the next time they log in.
 - 13 Click **OK**.

User Authentication

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 Hyperion 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's 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 Re-enter the password.
- 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 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.

3

Managing Node Access Groups

In This Chapter

Creating Node Access Groups	28
Editing Node Access Groups	29
Deleting Node Access Groups	29
Assigning Node Access Group Security	29

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.

Table 1 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 *Oracle Hyperion Data Relationship Management User Guide* for more information.
- If a node access group has access to any node in a hierarchy, the entire hierarchy is visible to all users of the node access group. Conversely, if a node access group does not have access to at least one node in a hierarchy, members of the group cannot open the hierarchy.

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 **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.

- 5 Select users from the **Available** list to assign to the group. Use the arrows to move users to the **Selected** list.
- 6 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.
- 5 Click **Save**.

4

Managing Property Categories

In This Chapter

Property Categories	31
Creating Property Categories.....	32
Editing Property Categories	32
Deleting Property Categories.....	33

Property Categories

Property categories enable the grouping of 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 2 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 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 de-select it.

- 5 Use the arrows to reorder the selected properties or click  to alphabetize the selected properties.
- 6 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.
- 7 Select the row for a user in the selected list and click  in the **Action** column.
- 8 From the **Access** column, select Read or Edit to assign the user a level of access to the property category.
- 9 Click  in the **Action** column to save the change or  to discard the change.
- 10 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 de-select 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.



Managing Property Definitions

In This Chapter

Creating Properties.....	36
Working with Formulas	40
Functions	42
Editing Property Definitions	77
Deleting Properties.....	78

Property definitions are used to manage the attributes of versions, hierarchies, and nodes in 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 3 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.

Creating Properties

Caution! The following procedure is for creating Defined or Lookup properties. If you are creating a Derived property, go to [“Creating Derived Properties with Formulas” on page 40](#).

➤ 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 for convenience purposes. The property Description is an optional, long descriptor that is displayed at the bottom of the Property Editor. Multiple properties can have the same Label as long as they are not in the same namespace.

- 4 Define parameters for the property:
 - **Data Type** – See [Property Data Types](#)
 - **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: To create a derived property, go to [“Creating Derived Properties with Formulas” on page 40.](#)

- **Default Value** – Default value for the property
- **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:

- **List** – Allows property values to be selected only from a pre-defined 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.

- **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 these cases, although the property is not literally inheriting values, the Inherited option should be enabled 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.

- **Hidden** – Hides the property in the property grid

6 Do any of the following:

- To assign a property to a category:
 - a. Select the **Categories** tab.
 - b. From the **Available** list, select categories and move them to the **Selected** list.
- To add a list of values to the property:
 - a. Select the **List** option.
 - b. Select the **List Values** tab.
 - c. Click **Add** and enter a value to the list.
 - d. 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.

- To add a lookup table to the property:
 - a. Select **Lookup** as the property type.
 - b. Select the **Lookup Table** tab.
 - c. Click **Add** to enter a new key-value pair to the list.
 - d. 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.

7 Click .

Data Types

Property data types are described in the following table.

Table 4 Property Data Types

Property Data Type	Description
AscGroup	Associated node group. Points to multiple nodes. The nodes point back to the AscGroup 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.

Property Data Type	Description
AscNode	<p>Associated node. Points to a single other node. The node pointed to points back to the AscNode node. Analogy: Marriage.</p> <p>Note: This data type should only be used with global node level properties.</p> <p>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.</p>
AscNodes	<p>Associated node list. Points to multiple nodes. The nodes pointed to point back to the AscNodes but not each other. Analogy: Friends.</p> <p>Note: This data type should only be used with global node level properties.</p> <p>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.</p>
Boolean	True or False
Date	<p>Date</p> <p>Note: Formatted based on the regional settings associated with the user's session.</p>
DateTime	Date and time.
Float	<p>Floating point value</p> <p>Note: Formatted based on the regional settings associated with the user's session.</p>
FormatMemo	Formatted memo – retains all formatting (spaces, tabs, new lines, and so on) to the text
GlobalNode	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
Hier	Points to a hierarchy
Integer	Integer value
LeafNode	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.
LimbNode	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.
ListGroup	Check list of items. One or more items can be selected from the list.
Memo	Memo field – formatting is not saved and data is merged into a single line of text
MultiNode	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
NodeProps	Points to the properties of a node
Property	Points to a property
RangeList	Defines a range of values – accepts only integer values

Property Data Type	Description
Sort	Integer value that is used for sorting
SortProp	Points to a Sort property
String	String value
Time	Time Note: Formatted based on the regional settings associated with the user's session.
Version	Points to a version

Working with Formulas


Properties based on a formula enable you to define custom logic using a native formula language in Data Relationship Management. Formulas are composed of functions and string literals and must follow specific syntax rules.

Creating Derived Properties with Formulas

► To create a formula property:

- 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 for convenience purposes. The property Description is an optional, long descriptor that is displayed at the bottom of the Property Editor. Multiple properties can have the same Label as long as they are not in the same namespace.

- 4 Select **Derived** as the **Property Type** and **Formula** as the **Deriver Class**.
- 5 Click .
- 6 Select the **Parameters** tab to define additional parameters for the formula.
- 7 You can enter a text formula or insert functions and properties in the following ways:
 - 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**.
- **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 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.

8 To test the formula, click Evaluate.

Formula Evaluation

You can test formulas when you create or modify a property definition. 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. 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 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.

Functions

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

For more information see:

- [“Special Characters” on page 43](#)
- [“Literals” on page 43](#)
- [“Function Definitions” on page 46](#)
- [“Format String Parameter” on page 43](#)
- [“Date-Time Format Strings” on page 45](#)

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 square brackets ([]). For example, `FlipList(NodeList, [comma])` performs the `FlipList` function on the comma-delimited list `NodeList`.

The following functions can take `comma`, `space`, or `tab`, in square brackets ([]), for the `Delimiter` parameter: `ArrayCount`, `ArrayIndex`, `ArrayItem`, `FlipList`, `Intersection`, `ListContains`, `PadList`, `RangeListContains`, and `IsRangeListSubset`.

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

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 a character. Therefore, extra spaces should not be used in formulas unless they are necessary to derive the appropriate result. You can use the `Remove Spaces` option to strip spaces from the formula prior to 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. There can be only one specifier in the format string.

Format specifiers use the following form:

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

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 in front of 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)

Character	Description
type	<p>Conversion type character, type</p> <p>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 <code>DecimalSeparator</code> and <code>ThousandSeparator</code> global variables or their <code>TFormatSettings</code> equivalent. Valid values for type are listed in the following table.</p>

Type Value	Description
d	<p>Decimal</p> <p>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 less digits, the resulting string is left-padded with zeros.</p>
u	<p>Unsigned decimal</p> <p>Similar to d but no sign is output.</p>
e	<p>Scientific</p> <p>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.</p>
f	<p>Fixed</p> <p>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.</p>
g	<p>General</p> <p>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.</p>
n	<p>Number</p> <p>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.</p>
m	<p>Money</p> <p>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 <code>CurrencyString</code>, <code>CurrencyFormat</code>, <code>NegCurrFormat</code>, <code>ThousandSeparator</code>, <code>DecimalSeparator</code>, and <code>CurrencyDecimals</code> global variables or their equivalent in a <code>TFormatSettings</code> data structure. If the format string contains a precision specifier, it overrides the value given by the <code>CurrencyDecimals</code> global variable or its <code>TFormatSettings</code> equivalent.</p>
s	<p>String</p> <p>The property value must be a character, a string, or a <code>PChar</code> 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.</p>

Type Value	Description
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)
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)

Specifier	Display
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.
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 quotes are displayed as-is and do not affect formatting

Function Definitions

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

Abbrev

Description

Returns the Abbrev (name) 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 username from the AddedBy property on the Stats category.

Syntax

```
AddedBy(): String
```

AddedOn

Description

Returns the Added On Date/Time in the server's regional format converted from the internal format of yyyy/mm/dd hh:mm:ss am/pm and is used with the Creation Date node property.

Syntax

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

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 of the first ancestor where Prop = x

Syntax

```
AncestorProp(Operator, Property, Value, FromTop, ReturnProp
```

And

Description

Returns True if both Boolean expressions specified 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 the list.

Syntax

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

Example

```
ArrayCount(Diet Cola, Orange Cola, Root Beer, Cola)
```

Return value is 4

ArrayIndex

Description

Returns the position of the first occurrence of the specified item within the list.

Returns 0 if the item is not found.

Syntax

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

Example

```
ArrayIndex(Cola,Diet Cola,Orange Cola,Root Beer,Cola)
```

Return value is 4

ArrayItem

Description

Returns the item in the list at the specified index position. Using a negative index value returns the last item in the list.

Syntax

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

Example

```
ArrayItem(Diet Cola,Orange Cola,Root Beer,Cola,4)
```

Return value is Cola

AscNodeProp

Description

Goes to the node pointed to by the specified node property and returns the specified property.

Syntax

```
AscNodeProp(LookupProp, ReturnProp)
```

BoolToStr

Description

Converts a specified Boolean value to the string “True” or “False” and returns the result.

Syntax

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

Example

```
BoolToStr(1)
```

Return value is True

Changed

Description

Returns True if the node's Changed flag has been set and is used with the Node Changed node property.

Syntax

```
Changed()
```

ChangedBy

Description

Returns the Changed By username and is used with the Last Updated By node property.

Syntax

```
ChangedBy(): String
```

ChangedOn

Description

Returns the Changed On Date/Time in the server's regional format converted from the internal format of yyyy/mm/dd hh:mm:ss am/pm and is used with the Last Update Date node property.

Syntax

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

Children

Description

Returns a comma-delimited list of children.

Syntax

```
Children()
```

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 delimited strings into one and returns the result.

Syntax

```
ConcatWithDelimiter(, True, Item1, Item2, Item3, Item4)
```

Example

```
ConcatWithDelimiter(, 1, Item1, Item2, Item3, Item4)
```

Return value is Item1; Item2; Item3; Item4

DefaultProp

Description

Returns the default value for the property.

Syntax

```
DefaultProp(Property)
```

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 the two specified floats and returns the result.

Syntax

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

Example

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

The return value is 1.69.

DualAncestorProp

Description

Traverses up the local or controlling hierarchy until it finds a node where the specified properties equal the specified values and then returns the specified return property.

Syntax

```
DualAncestorProp(Equiv1, Prop1, CompareVal1, Equiv2, Prop2, CompareVal2, FromTop, ReturnProp)
```

Equals

Description

Returns True if two specified values are equal. The type of comparison must be specified; valid types are string, integer, floating-point, Boolean, and date.

Syntax

```
Equals(ParamType, String1, String2: String): Boolean
```

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
```

Example

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

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

FloatToStr

Description

Returns a string representing the specified floating-point value.

Syntax

```
FloatToStr(AFloat: Float): String
```

Example

```
FloatToStr(1.001)
```

Return value is 1.001.

Format

Description

Invokes Format function using a specified format string, parameter type-identifier and parameter value of the specified type. This function is limited to one value parameter. The format string used is described in the format string section.

Syntax

```
Format(AFormat, ParamType, FormatParam: String): String
```

Example

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

Return value is 123.46

FormattedDate

Description

Formats the value of a date property based on a format string.

Syntax

```
FormattedDate(Property, Format string)
```

GreaterThan

Description

Returns True if the first of two specified integer values is greater than the second. Parameter type is optional and specifies the parameter types for the values to be compared. Valid parameter types are string, integer, float, and date. The default parameter types is integer.

Syntax

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

Example

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

The return value is False.

GreaterThanOrEqual

Description

Returns True if the first of two specified integer values is greater than or equal to the second. Parameter type is optional and specifies the parameter types for the values to be compared. Valid parameter types are string, integer, float, and date. The default parameter type is integer.

Syntax

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

Example

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

The return value is True.

HasChildWith

Description

Returns True if the specified expression is True for any child.

Syntax

```
HasChildWith(Expression1: 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. Returns False for the top node of a hierarchy.

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. Formula using this function must be in a local property.

Syntax

```
HasSiblingWith(Expression1: 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, NodeAbbrev, PropAbbrev: String): String
```

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 integer 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

If the specified expression evaluates to True, this function returns the value of the ResultIfTrue parameter, otherwise it returns the value of the ResultIfFalse parameter.

Syntax

```
If(Expression: Boolean; ResultIfTrue, ResultIfFalse: String): String
```

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.

InternalPrefix

Description

Returns the non-numeric prefix of a node name.

Syntax

```
InternalPrefix()
```

Intersection

Description

Returns the set of items common to both List1 and List2. The ordering of the results is based on how the items appear in the first list.

Syntax

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

Example

```
Intersection(A, B, C, D, E, C, E, F, A, )
```

The return value is A,C,E.

IntToStr

Description

Converts the specified integer to a string value and returns the result.

Syntax

```
IntToStr(Int: Integer): String
```

Example

```
IntToStr(12345)
```

The return value is 12345.

InvertedLevel

Description

Returns the maximum depth of descendants below the node.

Syntax

```
InvertedLevel()
```

IsAlpha

Description

Returns True if the specified string contains only letters A to Z (case-insensitive).

Syntax

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

Example

```
IsAlpha(A23D)
```

The return value is False.

IsNumeric

Description

Returns True if the specified string contains only numbers 0 to 9. The optional parameter `AllowBlanksAsNumeric` allows the function to return True for a blank string. The default for the parameter is False.

Syntax

```
IsNumeric(AString: String, [AllowBlanksAsNumeric]): String
```

Example

```
IsNumeric(12345)
```

The return value is True.

IsRangeListSubset

Description

Returns True if the subset exists in the Range List.

Syntax

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

Example

```
IsRangeListSubset(ARangeList, SubsetRangeList: RangeList, Delimiter: String): Boolean
```

The return value is True.

Length

Description

Returns the number of characters in the specified string.

Syntax

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

Example

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

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

LessThan

Description

Returns True if the first of two specified integer values is less than the second. Parameter type is optional and specifies the parameter types for the values to be compared. Valid parameter types are string, integer, float, and date. The default parameter type is integer.

Syntax

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

Example

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

The return value is True.

LessThanOrEqual

Description

Returns True if the first of two specified integer values is less than or equal to the second. Parameter type is optional and specifies the parameter types for the values to be compared. Valid parameter types are string, integer, float and date. The default parameter type is integer.

Syntax

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

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. The current node must be a local node for this function to work. To ensure that the node is local, any formula containing this function must be part of a local derived formula property.

Syntax

```
ListAncestors(): String
```

Example

```
ListAncestors()
```

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 the names of the current node's immediate children.

Syntax

```
ListChildren(): String
```

Example

```
ListChildren()
```

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 a Boolean value indicating whether a value is contained in a delimited list.

Syntax

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

Example

```
ListContains(PropValue(NodeList), Colas, ;)
```

The return value is true.

ListDescendants

Description

Returns a comma-delimited list of the names of the current node's descendants.

Syntax

```
ListDescendants(): String
```

Example

```
ListDescendants()
```

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.

ListPeers

Description

Returns a comma-delimited list of the names of the current node's peers (siblings). The current node must be a local node for this function to work. To ensure that the node is local, any formula containing this function must be part of a local derived formula property.

Syntax

```
ListPeers(): String
```

Example

```
ListPeers()
```

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

ListSiblings

Description

Returns a comma-delimited list of the names of the current node's siblings (peers). The current node must be a local node for this function to work. To ensure that the node is local, any formula containing this function must be part of a local derived formula property.

Syntax

```
ListSiblings(): String
```

Example

```
ListSiblings()
```

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

Converts the specified string to lower case and returns the result.

Syntax

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

Example

```
LowerCase(HOBBS)
```

The return value is hobbess.

LTrim

Description

Returns the specified string with all spaces trimmed from the left end.

Syntax

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

Example

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

The return value is 101203.

Modulus

Description

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

Syntax

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

Example

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

The return value is 1.

Multiply

Description

Multiplies two specified integers and returns the result.

Syntax

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

Example

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

The return value is 10.

MultiplyFloat

Description

Multiplies two specified floats and returns the result.

Syntax

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

Example

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

The return value is 10.948.

NextSibling

Description

Returns the next sibling for the specified node.

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 which the currently logged in user has rights.

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
```

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
```

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 (that is, it can not contain children.)

Syntax

```
NodeIsLeaf(): Boolean
```

Example

```
NodeIsLeaf()
```

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

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.

Syntax

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

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 date and/ or time. The optional DateTimeType parameter can be [date, time, or datetime].

Syntax

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

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 where the specified expression is 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 where the specified expression is 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 are 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 of the function. When using a function such as `HasChildWith`, the originating node can be referenced inside that function by using the `ParentPropValue` function. However, when using a function such as `HasSiblingWith` or `NumDescendantsWith`, the `OrigPropValue` function must be used.

Syntax

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

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 result is the original string.

Syntax

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

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 result is the original list.

Syntax

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

Example

```
PadList(1;2;3;4,;, True, 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. The current node must be a local node for this function to work. To ensure that the node is local, any formula containing this function must be part of a local derived formula property.

Syntax

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

Example

```
ParentPropValue(Abbrev)
```

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

Pos

Description

Searches for a specified substring within the specified string and returns an integer value that is the position of the first character of the substring within the string. Pos is case-sensitive. If the sub string is not found, Pos returns zero.

Syntax

```
Pos(ASubString, AString: String): Integer
```

Example

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

The return value is 4.

PreviousSibling

Description

Returns the previous sibling for the current node.

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
```

Example

```
PropControllingHier(TimeBalance)
```

The return value is Accounts.

PropDefaultValue

Description

Returns the default value of the specified property. This value is retrieved from the property definition.

Syntax

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

Example

```
PropDefaultValue(Currency)
```

The return value is USD.

PropertyCategories

Description

Returns a comma-delimited list of property categories for which the currently logged in user has rights. AccessType parameter is used to return property categories to which the user has ReadOnly access, ReadWrite access or Both.

Syntax

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

Example

```
PropertyCategories(Both)
```

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

PropMaxValue

Description

Returns the maximum value of the specified property. This value is retrieved from the property definition.

Syntax

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

Example

```
PropMaxValue(Volume)
```

The return value is 10.

PropMinValue

Description

Returns the minimum value of the specified property. This value is retrieved from the property definition.

Syntax

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

Example

```
PropMinValue(Volume)
```

The return value is 1.

PropValue

Description

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

Syntax

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

Example

```
PropValue(Volume)
```

The return value is 2.

RangeListContains

Description

Returns True if the Range List contains the specified value.

Syntax

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

The RangeList parameter looks like this: 1-10,101-10000,9999999-10000000000

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 checks the validity of the RangeList and combines ranges with contiguous boundaries.

ReplacementAbbrev

Description

If the current node is inactive, this function returns the Abbrev (name) of the current node's replacement node. The replacement node is the merge node when using the optional Merge functionality.

Syntax

```
ReplacementAbbrev(): String
```

Example

```
ReplacementAbbrev()
```

ReplacePropValue

Description

If the current node is inactive, this function returns the value of the specified property of the current node's replacement node. The replacement node is the merge node when using the optional Merge functionality.

Syntax

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

Example

```
ReplacePropValue(Description)
```

ReplaceStr

Description

Returns the string with instance(s) of the old pattern replaced by the new pattern.

Syntax

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


Example

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

The return value is B1;B2;B3.

RTrim

Description

Returns the specified string with all spaces trimmed from the right end.

Syntax

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

Example

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

The return value is 100.

StripPadChar

Description

Returns the result of stripping a specified pad character from the beginning of a specified string. The function can strip all instances of the specified character by specifying 0 (zero) as the strip count, or a specific number of characters to strip using a non-zero integer. If the original string contains fewer pad characters than are specified for stripping, this function does not strip non-pad characters.

Syntax

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

Example

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

The return value is 3333.

StrToBool

Description

Returns a Boolean value based on the specified string.

- If the specified string starts with a Y, T, or 1 (one), regardless of case or following characters, the function returns True.

- If the specified string starts with N, F, or 0 (zero), regardless of case or following characters, the function returns False.
- If the specified string does not represent a valid boolean value, an error is returned.

Syntax

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

Example

```
StrToBool(0)
```

The return value is False.

StrToFloat

Description

Returns the floating-point value of the specified string.

Note: A space or empty string is treated as a 0 (zero).

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

Syntax

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

Example

```
StrToFloat(11.101)
```

The return value is 11.101.

StrToInt

Description

Returns the integer value of the specified string.

Note: A space or empty string is treated as a 0 (zero).

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

Syntax

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

Example

```
StrToInt (101)
```

The return value is 101.

Stuff

Description

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

Syntax

```
Stuff(APropertyName, ACharsToReplace, AReplacement: String): 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(AString: String, Index, Count: Integer): String
```

Example

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

The return value is Co.

Subtract

Description

Returns the result of subtracting the second specified integer from the first.

Syntax

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

Example

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

The return value is 8.

SubtractFloat

Description

Returns the result of subtracting the second specified Float from the first.

Syntax

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

Example

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

The return value is 4.33.

Trim

Description

Returns the specified string with all spaces trimmed from both ends.

Syntax

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

Example

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

The return value is 101.

UpperCase

Description

Returns the specified string converted to all upper case.

Syntax

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

Example

```
UpperCase(smaller)
```

The return value is SMALLER.

UserName

Description

Returns the user name for the currently logged in user.

Syntax

```
UserName(): String
```

Example

```
UserName()
```

Return value is the user name.

XOr

Description

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

Syntax

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

Example

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

If the node is a leaf, or is at level 3 in the hierarchy but not both, then the return value is True.

Editing Property Definitions

► 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 Make changes to any of these parameters:
 - Label
 - Description
 - Hidden
 - Data Type
 - Default Value


- Lookup Property
- Column Width
- Minimum and Maximum Value/Length

See “[Creating Properties](#)” on page 36 for more information.

6 Click .

Deleting Properties

► 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.

6

Managing Validations

In This Chapter

Validation Classes	79
Validation Levels	81
Creating Validations	82
Assigning Validations	83
Editing Validations	83
Deleting Validations	83

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 5 Validation Classes

Validation Class	Level	Description	Parameters
Automatic hierarchy fail	Hierarchy	Automatically fails at hierarchy level for testing purposes	none
Automatic node fail	Global Node	Automatically fails nodes at the version level for testing purposes	none
Automatic version fail	Version	Automatically fails at the version level for testing purposes	none
Date range check	Node	Verifies that the From Date is earlier than or equal to the To Date	From Date Property, To Date Property

Validation Class	Level	Description	Parameters
Fails If Prop = TRUE in hierarchy	Node	Verifies that the specified boolean property has no True values in the specified hierarchy	Property, Hierarchy
Finds branches where property is not set	Node	Verifies that the specified property is set at least once on a specified branch	Property
Global property query validation	Global Node	Verifies using predefined query and expected result	Property query name, Failure value
Hierarchy contains a reference of a node when bool property is true	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
Hierarchy contains all where Prop = TRUE	Global Node	Verifies that the specified hierarchy contains all nodes where the specified property is True	Hierarchy, Property
Hierarchy contains all where Prop = value	Global Node	Verifies that the specified hierarchy contains all nodes for which the specified property has the specified value	Hierarchy, Property, Value
Invalid name length	Node	Verifies that the node name is not equal to a specified length.	Length
Limbs without any children	Node	Verifies that all limb nodes have children	none
Local property query validation	Node	Verifies using predefined query and expected result Only a local property query can be used.	Property query name, Failure value
Maximum # of nodes in a hierarchy	Hierarchy	Verifies that the number of nodes in the hierarchy does not exceed specified limit	Maximum number of nodes
Maximum # of nodes in a version	Version	Verifies that the number of nodes in the version does not exceed specified limit	Maximum number of nodes
Maximum child nodes	Version	Verifies that the number of children per node do not exceed specified limit	Maximum number of children
Merge node equivalency validation	Merge	Verifies that the affected node and merge node have the same value for the specified property	Global node property
Merge node property overridden validation	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
No default values allowed	Node	Verifies that no default values are used for the specified property	Property
No TRUE value on a branch	Node	Verifies that the specified boolean property is set to True at least once on a specified branch	Property
Node fail random	Node	Automatically fails the specified percentage of nodes for testing purposes	Failure percentage
Property equals value	Node	Fails for all nodes for which the specified property equals the specified value	Property, Value

Validation Class	Level	Description	Parameters
Property equivalency when a third boolean prop is true	Node	Property equivalency when a third boolean property is True.	Boolean property to evaluate, First Property, Second Property
Property length check	Node	Verifies that the specified property is at least minimum length and no more than maximum length	Property, Minimum Length, Maximum Length
Property set only once per branch	Node	Verifies that the specified property is set only once per branch	Property
Remove property validation	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
Required fields	Node	Verifies that, for all nodes for which the specified property has a specified value, each property in the required list has a value: <ul style="list-style-type: none"> ● 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
Single TRUE value on a branch	Node	Verifies that the specified boolean property is set to True only once per branch	Property
Unique 2 properties within a version	Global Node	Verifies that specified properties have no duplicate values within a version (If the Include Defaults flag is False, nodes with the default value are not included)	First property, Second property, Include Defaults
Unique property value within a branch	Node	Verifies that the specified property has unique value within a branch	Property
Unique property value within a hierarchy	Node	Verifies that the specified property has no duplicate values within a hierarchy (If the Include Defaults flag is False, nodes with the default value are not included)	Property, Include Defaults
Unique property value within a version	Global Node	Verifies that the specified property has no duplicate values within a version (If the Include Defaults flag is False, nodes with the default value are not included)	Property, Include Defaults

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.

Table 6 Validation Levels

Level	Definition	Example
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.

Level	Definition	Example
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
Version	Reviews the properties of a version.	Use to ensure that a version contains no more than 100,000 nodes
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.
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.
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.
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. Can be assigned and run at the hierarchy or version level.	Use to prevent the deletion of cost center nodes from a hierarchy

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.
- 9 Click .


Assigning Validations

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

Note: 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, see the *Oracle Hyperion 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.

7

Managing Node Types

In This Chapter

Defining Node Types	85
Editing Node Types	86
Deleting Node Types	86
Working with Node Glyphs	86

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 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** and 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.



Working with System Preferences

In This Chapter

System Preferences.....	89
Configuring System Preferences.....	95

System Preferences enable administrative users to edit settings that control the behavior of Data Relationship Management.

System Preferences

The following table describes Data Relationship Management system preferences.

Table 7 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.

System Preference	Type	Description
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	<p>User authentication method:</p> <ul style="list-style-type: none"> ● 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. <p>Default value is Internal.</p> <p>Note: A change to this preference requires a restart of the Data Relationship Management application.</p>
CopyLcl	Boolean	<p>True copies local values when a node is copied.</p> <p>Default value is True.</p>
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	<p>Default property copy mode.</p> <p>Valid values are Overridden, Selected, and ForceAll.</p> <p>Default value is Overridden.</p>
EnablePropCopyOptions	Role	<p>List of roles allowed access to the property copy options.</p> <p>Default values are Interactive User, Data Creator, Data Manager.</p>
EnforceListProps	Boolean	<p>True allows updates to a List Property with values from the pre-defined list only.</p> <p>Default value is True.</p>
FiltrChr	String	Set of characters for the Replace function on the Output Option screen of exports.
FindByProperties	Property	<p>List of properties available to search with when browsing a hierarchy.</p> <p>Note: The properties displayed are those to which a user has access. Also, the properties displayed may not be applicable to all hierarchies.</p>
FindWildcardAppend	Boolean	<p>True appends an asterisk (*) to the Find criteria when Exact Match is not selected.</p> <p>Default value is False.</p>
FindWildcardPrepend	Boolean	<p>True prepends an asterisk (*) to the Find criteria when Exact Match is not selected.</p> <p>Default value is False.</p>

System Preference	Type	Description
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 (~).
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.
LeafEdit	Role	List of roles allowed to change the Leaf property. Default values are Data Manager, Data Creator, Application Administrator, Access Manager.
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.

System Preference	Type	Description
LossLevel	String	<p>Loss level to capture.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> ● Defined ● All <p>Default value is Defined. Selecting All can significantly impact system performance for removed or deleted nodes with many property values.</p> <p>Note: A change to this preference requires a restart of the Data Relationship Management application.</p>
MaxDescr	Integer	<p>Maximum number of characters for node description. Valid values are 12 to 255.</p> <p>Default value is 80.</p>
MaxLeaf	Integer	<p>Maximum number of characters for the leaf name. Valid values are 3 to 20.</p> <p>Default value is 255.</p>
MaxLimb	Integer	<p>Maximum number of characters for the limb name. Valid values are 3 to 20.</p> <p>Default value is 255.</p>
PasswordDuration	Integer	<p>Number of days that a user password is valid. Valid values are 1 to 9999.</p> <p>Default value is 30.</p>
PasswordMaxLength	Integer	<p>Maximum number of characters for user password. Valid values are 0 to 255. Zero indicates no minimum.</p> <p>Default value is zero.</p>
PasswordMinLength	Integer	<p>Minimum number of characters for user password. Valid values are 0 to 9999. Zero indicates no minimum.</p> <p>Default value is 6.</p>
PasswordPolicyEnabled	Boolean	<p>True requires the password to contain three of the following elements:</p> <ul style="list-style-type: none"> ● Uppercase letters ● Lowercase letters ● Numbers ● Special characters <p>Default value is True.</p>
PasswordWarningPeriod	Integer	<p>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.</p> <p>Default value is 1.</p>
RenameLeaf	Role	<p>List of roles allowed to rename leaf nodes.</p> <p>Default values are Data Manager, Application Administrator, Access Manager.</p>

System Preference	Type	Description
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.
SharedNodeDelimiter	String	Specifies the delimiter between the node name and the shared node suffix. Default value is 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.
SharedNodeSequenceSeparator	String	Specifies the separator character to be placed after the shared node identifier. Default value is dash (-). 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.

System Preference	Type	Description
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: <ul style="list-style-type: none"> ● Logged Action ● Core Action ● Result Action ● Loss Action 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
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.

Setting Up Change Approval

The change approval system in 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.

Local Security for Global Properties

You use two system preferences — GlobalPropLocalSecurity and GlobalPropLocalOverride — to control local security on global properties.

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**.

9

Working with External Connections

In This Chapter

Defining External Connections	97
Editing External Connections	98
Deleting External Connections	98

Application administrators can define and configure common connections to external file systems and databases. Imports, exports and books can share connections to minimize maintenance of connectivity information. External connections enable the application server to directly access, read, or write data to these network resources.

Note: You must set up external resources before defining external connections.


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** or **System**.
- 5 Select a connection type: **Server File**, **FTP**, or **Database Table**.
- 6 Do the following:
 - If you selected **Server File**, enter a UNC path to the server.

Note: The Windows user account used by the 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
- If you selected **Database Table**, do the following:
 - Select the Data Access Provider: Oracle, SqlServer, or OleDb
 - Enter Connection String
 - Enter User ID
 - Enter User Password
 - Click  and then select tables from the Available list. Use the arrows to move tables to the Selected list.

7 Click **Test Connection**.


Note: For Database connection, click Refresh to get the list of tables.

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 **Save**.

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.

In This Chapter

Opening the Migration Utility.....	100
Extracting Metadata	100
Loading Metadata.....	101
Comparing Metadata	102
Viewing Metadata	103
Generating Reports	104

The 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 and System)
- Compares (Standard and System)
- Exports (Standard and System)
- Export Books (Standard and System)

- Imports (Standard and System)
- Blenders (Standard and System)
- System Preferences
- External Connections

Opening the Migration Utility

By default, the Migration Utility is installed to:

```
MIDDLEWARE_HOME\EPMSys11R1\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 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” on page 101](#).
- Click **Generate Reports for the Metadata File** to generate a report from the XML file. For more information, see [“Generating Reports” on page 104](#).

Loading Metadata

Only files with the 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.

➤ 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**.**
- 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 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 Oracle Hyperion Data Relationship Management, Fusion Edition application. For more information, see [“Loading Metadata” on page 101](#).
- Click **Generate Reports for the Metadata File** to generate a report from the XML file. For more information, see [“Generating Reports” on page 104](#).

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.**

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.

Index

A

Abbrev function, 46
Add function, 47
AddedBy function, 47
AddedOn function, 47
AddFloat function, 47
adding node glyphs, 86
AncestorProp function, 48
And function, 48
ArrayCount function, 48
ArrayIndex function, 49
ArrayItem function, 49
AscNodeProp function, 49
assigning validations, 83

B

BoolToStr function, 49

C

Changed function, 50
ChangedBy function, 50
ChangedOn function, 50
Children function, 50
Concat function, 51
ConcatWithDelimiter function, 51
configuring system preferences, 95
creating

- node access groups, 28
- properties, 36
- property categories, 32
- users, 22
- validations, 82

D

DefaultProp function, 51
defining

external connections, 97
node types, 85

deleting

external connections, 98
node access groups, 29
node glyphs, 86
node types, 86
properties, 78
property categories, 33
users, 25
validations, 83

derived properties with formulas

creating, 40

Descr function, 51

Divide function, 52

DivideFloat function, 52

DualAncestorProp function, 52

E

editing

external connections, 98
node access groups, 29
node types, 86
property categories, 32
property definitions, 77
validations, 83

Equals function, 53

external connections

defining, 97
deleting, 98
editing, 98

F

FlipList function, 53

FloatToStr function, 53

Format function, 54

FormattedDate function, 54

- formula evaluation, [41](#)
- formula functions, [42](#)
 - Abbrev, [46](#)
 - Add, [47](#)
 - AddedBy, [47](#)
 - AddedOn, [47](#)
 - AddFloat, [47](#)
 - AncestorProp, [48](#)
 - And, [48](#)
 - ArrayCount, [48](#)
 - ArrayIndex, [49](#)
 - ArrayItem, [49](#)
 - AscNodeProp, [49](#)
 - BoolToStr, [49](#)
 - Changed, [50](#)
 - ChangedBy, [50](#)
 - ChangedOn, [50](#)
 - Children, [50](#)
 - Concat, [51](#)
 - ConcatWithDelimiter, [51](#)
 - DefaultProp, [51](#)
 - Descr, [51](#)
 - Divide, [52](#)
 - DivideFloat, [52](#)
 - DualAncestorProp, [52](#)
 - Equals, [53](#)
 - FlipList, [53](#)
 - FloatToStr, [53](#)
 - Format, [54](#)
 - FormattedDate, [54](#)
 - GreaterThan, [54](#)
 - GreaterThanOrEqual, [55](#)
 - HasChildWith, [55](#)
 - HasParentNode, [55](#)
 - HasSiblingWith, [56](#)
 - HierNodePropValue, [56](#)
 - ID, [56](#)
 - If, [57](#)
 - InternalPrefix, [57](#)
 - Intersection, [57](#)
 - IntToStr, [58](#)
 - InvertedLevel, [58](#)
 - IsAlpha, [58](#)
 - IsNumeric, [58](#)
 - IsRangeListSubset, [59](#)
 - Length, [59](#)
 - LessThan, [59](#)
 - LessThanOrEqual, [60](#)
 - ListAncestors, [60](#)
 - ListChildren, [60](#)
 - ListContains, [61](#)
 - ListDescendants, [61](#)
 - ListPeers, [61](#)
 - ListSiblings, [62](#)
 - LowerCase, [62](#)
 - LTrim, [62](#)
 - Modulus, [63](#)
 - Multiply, [63](#)
 - MultiplyFloat, [63](#)
 - NextSibling, [64](#)
 - NodeAccessGroups, [64](#)
 - NodeExists, [64](#)
 - NodeInHier, [65](#)
 - NodeIsLeaf, [65](#)
 - NodePropValue, [65](#)
 - Not, [66](#)
 - Now, [66](#)
 - NumChildWith, [66](#)
 - NumDescendantsWith, [67](#)
 - Or, [67](#)
 - OrigPropValue, [67](#)
 - PadChar, [68](#)
 - PadList, [68](#)
 - ParentPropValue, [68](#)
 - Pos, [69](#)
 - PreviousSibling, [69](#)
 - PropControllingHier, [69](#)
 - PropDefaultValue, [70](#)
 - PropertyCategories, [70](#)
 - PropMaxValue, [70](#)
 - PropMinValue, [71](#)
 - PropValue, [71](#)
 - RangeListContains, [71](#)
 - ReplacementAbbrev, [72](#)
 - ReplacePropValue, [72](#)
 - ReplaceStr, [72](#)
 - RTrim, [73](#)
 - StripPadChar, [73](#)
 - StrToBool, [73](#)
 - StrToFloat, [74](#)
 - StrToInt, [74](#)
 - Stuff, [75](#)
 - SubString, [75](#)
 - Subtract, [75](#)

SubtractFloat, [76](#)
 Trim, [76](#)
 UpperCase, [76](#)
 UserName, [77](#)
 XOr, [77](#)
 formula syntax checks, [41](#)

G

GreaterThan function, [54](#)
 GreaterThanOrEqual function, [55](#)

H

HasChildWith function, [55](#)
 HasParentNode function, [55](#)
 HasSiblingWith function, [56](#)
 HierNodePropValue function, [56](#)

I

ID function, [56](#)
 If function, [57](#)
 InternalPrefix function, [57](#)
 Intersection function, [57](#)
 IntToStr function, [58](#)
 InvertedLevel function, [58](#)
 IsAlpha function, [58](#)
 IsNumeric function, [58](#)
 IsRangeListSubset function, [59](#)

L

Length function, [59](#)
 LessThan function, [59](#)
 LessThanOrEqual function, [60](#)
 ListAncestors function, [60](#)
 ListChildren function, [60](#)
 ListContains function, [61](#)
 ListDescendants function, [61](#)
 ListPeers function, [61](#)
 ListSiblings function, [62](#)
 locking out users, [24](#)
 LowerCase function, [62](#)
 LTrim function, [62](#)

M

Modulus function, [63](#)
 Multiply function, [63](#)

MultiplyFloat function, [63](#)

N

NextSibling function, [64](#)
 node access groups
 creating, [28](#)
 deleting, [29](#)
 editing, [29](#)
 security, [29](#)
 node glyphs
 adding and deleting, [86](#)
 node types
 defining, [85](#)
 deleting, [86](#)
 editing, [86](#)
 NodeAccessGroups function, [64](#)
 NodeExists function, [64](#)
 NodeInHier function, [65](#)
 NodeIsLeaf function, [65](#)
 NodePropValue function, [65](#)
 Not function, [66](#)
 Now function, [66](#)
 NumChildWith function, [66](#)
 NumDescendantsWith function, [67](#)

O

Or function, [67](#)
 OrigPropValue function, [67](#)

P

PadChar function, [68](#)
 PadList function, [68](#)
 ParentPropValue function, [68](#)
 passwords
 changing user, [24](#)
 Pos function, [69](#)
 PreviousSibling function, [69](#)
 PropControllingHier function, [69](#)
 PropDefaultValue function, [70](#)
 properties
 categories, [31](#)
 creating, [36](#)
 data types, [38](#)
 deleting, [78](#)
 property categories, [31](#)
 creating, [32](#)

- deleting, [33](#)
- editing, [32](#)
- property data types, [38](#)
- property definitions
 - editing, [77](#)

- PropertyCategories function, [70](#)
- PropMaxValue function, [70](#)
- PropMinValue function, [71](#)
- PropValue function, [71](#)

R

- RangeListContains function, [71](#)
- ReplacementAbbrev function, [72](#)
- ReplacePropValue function, [72](#)
- ReplaceStr function, [72](#)
- RTrim function, [73](#)

S

- StripPadChar function, [73](#)
- StrToBool function, [73](#)
- StrToFloat function, [74](#)
- StrToInt function, [74](#)
- Stuff function, [75](#)
- SubString function, [75](#)
- Subtract function, [75](#)
- SubtractFloat function, [76](#)
- system preferences
 - configuring, [95](#)
 - defined, [89](#)

T

- Trim function, [76](#)

U

- unlocking users, [24](#)
- UpperCase function, [76](#)
- user authentication, [23](#)
- user roles
 - defined, [18](#)
- UserName function, [77](#)
- users
 - authenticating, [23](#)
 - changing passwords, [24](#)
 - changing roles, [25](#)
 - creating, [22](#)

- deleting, [25](#)
- locking out, [24](#)
- roles, [18](#)
- unlocking, [24](#)
- viewing login status, [25](#)

V

- validation classes, [79](#)
- validation levels, [81](#)
- validations
 - assigning, [83](#)
 - creating, [82](#)
 - deleting, [83](#)
 - editing, [83](#)

X

- XOr function, [77](#)