Oracle® Hyperion Data Relationship Management

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 11.1.2.2
# Contents

**Documentation Accessibility** ...........................................................  9

**Chapter 1. Release Overview** ..........................................................  11
  New Feature Terms .................................................................  12
  Changes to Existing Feature Terms ...............................................  12
  Obsolete Terms .................................................................  13

**Chapter 2. Managing Users** ...........................................................  15
  User Permissions .................................................................  15
    Version Permissions ..........................................................  15
    Request Permissions ..........................................................  16
    Query Permissions ...........................................................  16
    Compare Permissions ...........................................................  17
    Import Permissions .............................................................  17
    Blender Permissions .............................................................  17
    Export Permissions ............................................................  18
    Script Permissions ..............................................................  18
    Audit Permissions ..............................................................  18
    Application Permissions .......................................................  19
    Access Permissions ............................................................  19
  User Roles .................................................................  20
  Creating Users ...............................................................  23
  User Authentication ............................................................  24
  Modifying Users ...............................................................  25
    Changing Passwords ............................................................  25
    Locking Out Users ............................................................  25
    Unlocking Users ..............................................................  25
    Changing User Roles and Assignments .......................................  26
  Deleting Users ...............................................................  26
  Viewing User Login Status .......................................................  27

**Chapter 3. Managing Node Access Groups** ...........................................  29
  Creating Node Access Groups .......................................................  30
HasChildWith ....................................................  73
HasParentNode ...................................................  73
HasSiblingWith ...................................................  74
HierNodePropValue ................................................  74
ID .............................................................  74
If ..............................................................  75
InheritedPropOrigin ................................................  75
InRange .........................................................  75
InternalPrefix .....................................................  76
Intersection ......................................................  76
IntToStr .........................................................  77
InvertedLevel .....................................................  77
IsAlpha .........................................................  77
IsAlphaNumeric ...................................................  77
IsBlank ..........................................................  78
IsBottomNode .....................................................  78
IsDataType .......................................................  78
IsDefinedPropVal ..................................................  79
IsNodeAbove .....................................................  79
IsNodeBelow .....................................................  80
IsNumeric .......................................................  80
IsRangeListSubset ..................................................  80
Length ..........................................................  80
LessThan ........................................................  81
LessThanOrEqual ..................................................  81
ListAncestors .....................................................  81
ListChildren ......................................................  82
ListContains ......................................................  82
ListDescendants ...................................................  83
ListDistinct .......................................................  83
ListNodePropValues ................................................  84
ListNodesWith ....................................................  84
ListRelatedNodesWith ..............................................  84
ListSiblings .......................................................  85
LowerCase .......................................................  85
LTrim ..........................................................  86
MaxList .........................................................  86
MinList ..........................................................  86
Modulus ........................................................  87
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiply</td>
<td>87</td>
</tr>
<tr>
<td>MultiplyFloat</td>
<td>87</td>
</tr>
<tr>
<td>NextSibling</td>
<td>88</td>
</tr>
<tr>
<td>NodeAccessGroups</td>
<td>88</td>
</tr>
<tr>
<td>NodeExists</td>
<td>88</td>
</tr>
<tr>
<td>NodeInHier</td>
<td>89</td>
</tr>
<tr>
<td>NodeIsLeaf</td>
<td>89</td>
</tr>
<tr>
<td>NodePropValue</td>
<td>89</td>
</tr>
<tr>
<td>Not</td>
<td>90</td>
</tr>
<tr>
<td>Now</td>
<td>90</td>
</tr>
<tr>
<td>NumChildWith</td>
<td>90</td>
</tr>
<tr>
<td>NumDescendantsWith</td>
<td>91</td>
</tr>
<tr>
<td>Or</td>
<td>91</td>
</tr>
<tr>
<td>OrigPropValue</td>
<td>91</td>
</tr>
<tr>
<td>PadChar</td>
<td>92</td>
</tr>
<tr>
<td>PadList</td>
<td>92</td>
</tr>
<tr>
<td>ParentPropValue</td>
<td>93</td>
</tr>
<tr>
<td>Pos</td>
<td>93</td>
</tr>
<tr>
<td>PreviousSibling</td>
<td>93</td>
</tr>
<tr>
<td>PropControllingHier</td>
<td>94</td>
</tr>
<tr>
<td>PropDefaultValue</td>
<td>94</td>
</tr>
<tr>
<td>PropertyCategories</td>
<td>94</td>
</tr>
<tr>
<td>PropMaxValue</td>
<td>95</td>
</tr>
<tr>
<td>PropMinValue</td>
<td>95</td>
</tr>
<tr>
<td>PropValue</td>
<td>95</td>
</tr>
<tr>
<td>RangeListContains</td>
<td>96</td>
</tr>
<tr>
<td>ReplacementAbbrev</td>
<td>96</td>
</tr>
<tr>
<td>ReplacePropValue</td>
<td>96</td>
</tr>
<tr>
<td>ReplaceStr</td>
<td>97</td>
</tr>
<tr>
<td>RTrim</td>
<td>97</td>
</tr>
<tr>
<td>SortList</td>
<td>97</td>
</tr>
<tr>
<td>StripPadChar</td>
<td>98</td>
</tr>
<tr>
<td>StrToBool</td>
<td>98</td>
</tr>
<tr>
<td>StrToFloat</td>
<td>98</td>
</tr>
<tr>
<td>StrToInt</td>
<td>99</td>
</tr>
<tr>
<td>Stuff</td>
<td>99</td>
</tr>
<tr>
<td>SubString</td>
<td>99</td>
</tr>
<tr>
<td>Subtract</td>
<td>100</td>
</tr>
<tr>
<td>SubtractFloat</td>
<td>100</td>
</tr>
</tbody>
</table>
Documentation Accessibility

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.
In the 11.1.2 release of Oracle Hyperion Data Relationship Management, 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

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>An instance of Data Relationship Management. The user interface can access multiple applications.</td>
</tr>
<tr>
<td>As Sibling</td>
<td>An Add/Insert/Move/Put action allows placement of a node as a sibling to another node.</td>
</tr>
<tr>
<td>Clipboard</td>
<td>Copy nodes here to work with them across multiple hierarchy windows.</td>
</tr>
<tr>
<td>Connection</td>
<td>External locations relative to the server that can be read from and written to.</td>
</tr>
<tr>
<td>Object Access</td>
<td>User metadata objects can now be:</td>
</tr>
<tr>
<td></td>
<td>- 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.</td>
</tr>
<tr>
<td></td>
<td>- 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.</td>
</tr>
<tr>
<td></td>
<td>- 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.</td>
</tr>
<tr>
<td>Server File</td>
<td>A type of connection used to access a network file system or an FTP directory.</td>
</tr>
<tr>
<td>Shortcuts</td>
<td>Context-sensitive links that navigate the user to another task group retaining the current selection.</td>
</tr>
<tr>
<td>Use Fast Deletes</td>
<td>During an export, enables a bulk deletion on a target database table.</td>
</tr>
<tr>
<td>User Roles</td>
<td>Permission-based roles that control user access to product features. Roles:</td>
</tr>
<tr>
<td></td>
<td>- Anonymous User</td>
</tr>
<tr>
<td></td>
<td>- Workflow User</td>
</tr>
<tr>
<td></td>
<td>- Interactive User</td>
</tr>
<tr>
<td></td>
<td>- Data Creator</td>
</tr>
<tr>
<td></td>
<td>- Data Manager</td>
</tr>
<tr>
<td></td>
<td>- Access Manager</td>
</tr>
<tr>
<td></td>
<td>- Application Administrator</td>
</tr>
<tr>
<td>Version and Hierarchy Owner</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Changes to Existing Feature Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Script</td>
<td>Formerly the Automator interface or an Automator script.</td>
</tr>
<tr>
<td>Controlled Property</td>
<td>For a hierarchy, formerly assigning a controlling hierarchy to a property</td>
</tr>
<tr>
<td>Download</td>
<td>Formerly Save to File or Print</td>
</tr>
<tr>
<td>Hierarchy Group</td>
<td>Formerly a System Category, which is used for grouping hierarchies</td>
</tr>
<tr>
<td>Term</td>
<td>Description of Change</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>Locate</td>
<td>Formerly the Synchronize By Name feature</td>
</tr>
<tr>
<td>Load Status</td>
<td>Formerly represented as bold-faced print for an open/closed version</td>
</tr>
<tr>
<td>Match</td>
<td>Formerly the Synchronize By Property feature</td>
</tr>
<tr>
<td>Paste Properties</td>
<td>Properties can be pasted for any node that is taken and copied to the clipboard.</td>
</tr>
<tr>
<td>Put</td>
<td>Extended to support reordering children in addition to Insert and Move</td>
</tr>
<tr>
<td>Search</td>
<td>Formerly the Find Node feature</td>
</tr>
<tr>
<td>Take/Copy Properties</td>
<td>Includes the former Take and Copy Properties features</td>
</tr>
<tr>
<td>Validation (Batch)</td>
<td>Formerly Verification (and Verify)</td>
</tr>
</tbody>
</table>

### Obsolete Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbrev</td>
<td>Replaced by Name. Except for the Abbrev() formula function.</td>
</tr>
<tr>
<td>Automator</td>
<td>Replaced by Action Script</td>
</tr>
<tr>
<td>Export Preview</td>
<td>Replaced by Download</td>
</tr>
<tr>
<td>ListPeers() formula function</td>
<td>Replaced by ListSiblings()</td>
</tr>
<tr>
<td>System Category</td>
<td>Replaced by Hierarchy Group</td>
</tr>
<tr>
<td>User Types</td>
<td>Replaced by User Roles and Permissions</td>
</tr>
<tr>
<td>• User</td>
<td></td>
</tr>
<tr>
<td>• Functional Administrator</td>
<td></td>
</tr>
<tr>
<td>• Security Administrator</td>
<td></td>
</tr>
<tr>
<td>• System Administrator</td>
<td></td>
</tr>
<tr>
<td>Verification (and Verify)</td>
<td>Replaced by Batch Validation (and Validate)</td>
</tr>
<tr>
<td>Web Publishing</td>
<td>Replaced by Anonymous Access</td>
</tr>
</tbody>
</table>
In This Chapter

User Permissions........................................................................................................15
User Roles ..................................................................................................................20
Creating Users............................................................................................................23
User Authentication....................................................................................................24
Modifying Users..........................................................................................................25
Deleting Users............................................................................................................26
Viewing User Login Status.........................................................................................27

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

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Versions - User has access to Version and Hierarchy menu options</td>
<td>Browse Versions - Users have access to any version that they are granted rights to in Node Access Groups</td>
<td>NA</td>
</tr>
<tr>
<td>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.</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>Manage Hierarchies - Users have access to Hierarchy menu options.</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
### Request Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manage Requests</strong> - Users can delete any request in the system that has not already been committed.</td>
<td><strong>Create Requests</strong> - Users can query any request in the system and can manage (update/delete) any request of which they are the owner.</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Query Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manage System Queries</strong> - 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.</td>
<td><strong>Manage User Queries</strong> - 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.</td>
<td><strong>Run Query</strong> - 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.</td>
</tr>
<tr>
<td><strong>Manage Standard Queries</strong> - 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.</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
## Compare Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manage System Compares</strong> - 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.</td>
<td><strong>Manage User Compares</strong> - 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.</td>
<td><strong>Run Compare</strong> - 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.</td>
</tr>
<tr>
<td><strong>Manage Standard Compares</strong> - 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.</td>
<td><strong>Manage Standard Compares</strong> - 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.</td>
<td>NA</td>
</tr>
</tbody>
</table>

## Import Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manage System Imports</strong> - Users have access to system imports and Import menu options. Users have restricted access to Property selector based on Property Category security.</td>
<td><strong>Manage User Imports</strong> - 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.</td>
<td><strong>Run Import</strong> - Users can view and run any Standard import. Users have restricted access to Property selector based on Property Category security.</td>
</tr>
<tr>
<td><strong>Manage Standard Imports</strong> - Users have access to Import menu options for Standard imports. Users have restricted access to Property selector based on Property Category security.</td>
<td><strong>Manage Standard Imports</strong> - Users have access to Import menu options for Standard imports. Users have restricted access to Property selector based on Property Category security.</td>
<td>NA</td>
</tr>
</tbody>
</table>

## Blender Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manage System Blenders</strong> - 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.</td>
<td><strong>Manage User Blenders</strong> - Users have access to view and run User and Standard blenders. Users do not have access to Blender menu options for Standard Blenders.</td>
<td><strong>Run Blender</strong> - 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.</td>
</tr>
</tbody>
</table>

User Permissions 17
### Permission Level 1
### Permission Level 2
### Permission Level 3

**Manage Standard Blenders** - Users have access to Blender menu options for Standard blenders. Users have restricted access to Version, Hierarchy, Node, and Property selectors based on Node Access Group assignments and Property Category security.

| Manage Standard Blenders | NA |

---

### Export Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manage System Exports</strong> - 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.</td>
<td><strong>Manage User Exports</strong> - 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.</td>
<td><strong>Run Export</strong> - 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.</td>
</tr>
<tr>
<td>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.</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>

---

### Script Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Run Action Script</strong> - 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.</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

---

### Audit Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Audit User Transactions</strong> - 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.</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
### 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.

### 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.

### Application Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Application</td>
<td>Manage Categories</td>
<td>Browse Categories</td>
</tr>
<tr>
<td>Manage Properties</td>
<td>Browse Properties</td>
<td>Manage Property Lists</td>
</tr>
<tr>
<td>Manage Validations</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Manage Node Types</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Manage Preferences</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Access Permissions

<table>
<thead>
<tr>
<th>Permission Level 1</th>
<th>Permission Level 2</th>
<th>Permission Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Access</td>
<td>Manage Users</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Manage Roles</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Manage Access Groups</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Manage Property Access</td>
<td>NA</td>
</tr>
</tbody>
</table>
## 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:

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Access Manager</th>
<th>Anonymous User</th>
<th>Application Administrator</th>
<th>Data Creator</th>
<th>Data Manager</th>
<th>Interactive User</th>
<th>Workflow User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Versions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browse Versions</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create Versions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Hierarchies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browse Hierarchies</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create Hierarchies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create Requests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage System Queries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage User Queries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Query</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Standard Queries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage System Compares</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permissions</td>
<td>User Roles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td>Level 2</td>
<td>Level 3</td>
<td>Access Manager</td>
<td>Anonymous User</td>
<td>Application Administrator</td>
<td>Data Creator</td>
<td>Data Manager</td>
<td>Interactive User</td>
<td>Workflow User</td>
</tr>
<tr>
<td>Manage User</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compares</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Compare</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Manage Standard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compares</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imports</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Manage User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imports</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Import</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Standard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imports</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Manage System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blenders</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Blenders</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run Blender</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Standard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Blenders</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exports</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Manage User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exports</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Run Export</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Manage Standard</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Exports</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permissions</td>
<td>User Roles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Level 1</strong></td>
<td><strong>Level 2</strong></td>
<td><strong>Level 3</strong></td>
<td>Access</td>
<td>Anonymous</td>
<td>Application Administrator</td>
<td>Data Creator</td>
<td>Data Manager</td>
<td>Interactive User</td>
<td>Workflow User</td>
</tr>
<tr>
<td>Run Action Script</td>
<td></td>
<td></td>
<td>Manager</td>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit User Transactions</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit Data Transactions</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audit System Transactions</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Manage Categories</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browse Categories</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Manage Properties</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browse Properties</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Manage Property Lists</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Manage Validations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Node Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Preferences</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Access</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Users</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Roles</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manage Access Groups</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

22 Managing Users
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.
Enter a password for the user.

Enter the password again.

Optional: Select User must change password at next login to require the user to change their password the next time they log in.

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 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.
3. Select a user and click .
4. Click .
5. Enter a new password for the user.
6. Enter the password again.
7. Optional: Select User must change password at next login to require the user to change their password the next time they log in.
8. Click OK.

Locking Out Users

You can lockout a user to prevent their access to a 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.
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.
Managing Node Access Groups

In This Chapter

Creating Node Access Groups ................................................................. 30
Editing Node Access Groups ................................................................. 31
Deleting Node Access Groups ................................................................. 31
Assigning Node Access Group Security .................................................. 31

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

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

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’s 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.
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.
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.
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.
Domains are used to manage referential integrity for multiple sets of nodes from different sources within the same Data Relationship Management application. A domain is a related set of nodes of a common type to consistently manage their usage across different versions within the same application. A domain provides a simple method for:

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

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

## Creating Domains

To create a domain:

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

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

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

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

• **Allow Node Delete** – Select if you want to give users the ability to delete nodes from the version.

• **Allow Leaf Edit** – Select if you want to give users the ability to change the leaf system property value for nodes in the domain.

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

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

5 Click ✉.

### Editing Domains

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

• The name cannot be changed

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

To edit a domain:

1 **From the Home page, select Administer.**

2 Select a domain and click ✉.

3 **Make changes to the domain and click ✉.**

### Deleting Domains

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

**Note:** If a domain with nodes assigned is deleted, all nodes that are assigned to the node are reverted back to non-domain nodes.
To delete a domain:

1. From the Home page, select Administer.
2. Select a domain and click ✗.
3. Select Delete this Domain.
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

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>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.</td>
</tr>
<tr>
<td>Shared Info</td>
<td>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.</td>
</tr>
<tr>
<td>Stats</td>
<td>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.</td>
</tr>
<tr>
<td>Validation</td>
<td>Validations assigned for the node—one property for each validation</td>
</tr>
<tr>
<td>Leaf Access</td>
<td>Node security groups and their leaf access levels for the node—one property for each group</td>
</tr>
<tr>
<td>Limb Access</td>
<td>Node security groups and their limb access levels for the node—one property for each group</td>
</tr>
</tbody>
</table>
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 \( \uparrow \) 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 \( \uparrow \) 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 \( \uparrow \) in the Action column to save the change or \( \downarrow \) to discard the change.
10. Click \( \uparrow \).

Editing Property Categories

To edit a property category:
1. From the Home page, select Administer.
2. Select a property category and click \( \uparrow \).
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 \( \uparrow \) to alphabetize the selected properties.
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.

Select the row for a user in the selected list and click in the Action column.

From the Access column, select Read or Edit to assign the user a level of access to the property category.

Click in the Action column to save the change or to discard the change.

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Qualified Name</td>
<td>Custom.AccountType</td>
</tr>
<tr>
<td>Field</td>
<td>Example</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>Namespace</td>
<td>Custom</td>
</tr>
<tr>
<td>Name</td>
<td>AccountType</td>
</tr>
<tr>
<td>Label</td>
<td>AccountType</td>
</tr>
</tbody>
</table>

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” on page 47.

➢ 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:**

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

- **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 this 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” on page 47.

Default Value – Default value for the property

Domain – For any property where the data type is Node, Limb Node, LeafNode, MultiNode, Associated Node, Associated Nodes, or Associated Group (all of which represent a node or nodes stored as the value), a Domain drop-down is available. The drop-down contains all the domains defined in the system and you can optionally select one of the existing domains.

Column Width – Width for fixed-width columns if the property type is Defined.

Minimum Value/Length – Value or length for the property based on data type.

Maximum Value/Length – Value or length for the property based on data type.

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 such cases, although the property is not literally inheriting values, enable the Inherited option to allow the specification of a controlling hierarchy.

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

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

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

6 Do any of the following:

- To assign a property to categories, select categories from the **Available** list and move them to the **Selected** list.

- If you selected the **Defined** property type along with the **List** option, on the **List Values** tab do the following:
  a. Click **Add** and enter a value to the list.
  b. Click **Save** in the Action column for the row.

  Note: Use Move or Delete for each row to reorder or delete list values. Use Edit or double-click a row to edit it and Cancel to cancel edits.

- If you selected the **Lookup** property type, select the **Lookup Table** tab and do the following:
  a. Click **Add** to enter a new key-value pair to the list.
  b. Click **Save** in the Action column for the row.

  Note: Use Move or Delete for each row to reorder or delete list values. Use Edit or double-click a row to edit it and Cancel to cancel edits.

7 Click ![button](button.png).

**Data Types**

Property data types are described in the following table.

**Table 4  Property Data Types**

<table>
<thead>
<tr>
<th>Property Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Associated Group   | Associated node group. Points to multiple nodes. The nodes point back to the Associated Group node and to each other. Analogy: Fraternity.  
  **Note:** This data type should only be used with global node level properties.  
  **Caution!** Associated node properties that are loaded by an import may not correctly point to all other nodes as a result of their not yet existing in the version based on the order in which nodes are imported. |
<table>
<thead>
<tr>
<th>Property Data Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Associated Node    | Associated node. Points to a single other node. The node pointed to points back to the Associated Node node. Analogy: Marriage.  
**Note:** This data type should only be used with global node level properties.  
**Caution!** Associated node properties that are loaded by an import may not correctly point to all other nodes as a result of their not yet existing in the version based on the order in which nodes are imported. |
| Associated Nodes   | Associated node list. Points to multiple nodes. The nodes pointed to point back to the Associated Nodes but not each other. Analogy: Friends.  
**Note:** This data type should only be used with global node level properties.  
**Caution!** Associated node properties that are loaded by an import may not correctly point to all other nodes as a result of their not yet existing in the version based on the order in which nodes are imported. |
| Boolean            | True or False |
| Date               | Date  
**Note:** Formatted based on the regional settings associated with the user's session. |
| DateTime           | Date and time. |
| Float              | Floating point value  
**Note:** Formatted based on the regional settings associated with the user's session. |
| Formatted Memo     | Formatted memo — retains all formatting (spaces, tabs, new lines, and so on) to the text. Also allows for hyperlink text to be included in the formatted memo. See the Hyperlink data type for details on formatting URLs for hyperlinks.  
**Note:** Non URL text is not suppressed when both text and hyperlink is used in property value. |
| Global Node        | Points to a node in a version; when value is assigned it shows node name only in the value field of the property grid |
| Group              | List of comma-delimited items |
| Hierarchy          | Points to a hierarchy |
| Hierarchy Group    | Points to a hierarchy group.  
Hierarchy group properties allow hierarchies to be grouped in multiple ways based on the context in which you want to view them. You can group hierarchies within the same version in different ways based on usage. |
| Hyperlink          | Allows for hyperlink capability for URL text. Multiple URL input is separated by Carriage Return-Linefeed (CRLF, or 0x0D0A) with no spaces. Entered URLs are displayed as navigable hyperlinks. Non-URL text in this field is suppressed. Only the parsed, delimited URLs or formatted URLs are displayed. URLs should follow this format:  
[url=http://URL]URL_Title[/url]  
where http_URL specifies the hyperlink text and URL_Title specifies the text displayed to the user.  
For example, this markup example: [url=http://support.oracle.com]Oracle Support[/url] would render in the property grid as Oracle Support. |
<p>| Integer            | Integer value |
| Leaf Node          | Points to a leaf node in a hierarchy. When value is assigned it shows hierarchy name and node name in the value field of the Property Grid. |</p>
<table>
<thead>
<tr>
<th>Property Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limb Node</td>
<td>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.</td>
</tr>
<tr>
<td>List Group</td>
<td>Check list of items. One or more items can be selected from the list.</td>
</tr>
</tbody>
</table>
| Memo               | Memo field — formatting is not saved and data is merged into a single line of text. Also hyperlink in the memo. See the Hyperlink data type for details on formatting URLs for hyperlinks.  
  **Note:** Non URL text is not suppressed when both text and hyperlink is used in property value. |
| Multiple Node      | Points to multiple nodes |
| Node               | Points to a node in a hierarchy; when value is assigned, it shows hierarchy name and node name in the value field of the property grid. |
| Node Properties    | Points to the properties of a node |
| Property           | Points to a property |
| Range List         | Defines a range of values; accepts only integer values |
| Sort               | Integer value that is used for sorting |
| Sort Property      | Points to a Sort property |
| Standard Query     | Points to a standard query |
| String             | String value |
| Time               | Time  
  **Note:** Formatted based on the regional settings associated with the user's session. |
| Version            | Points to a version |

### 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. Modify any editable parameters.

**Caution!** If you change the Property Type from a defined value (RWDerived or Defined) to a value that does not allow storage (Derived or Lookup), defined property values are deleted and this data will be lost. Before making this type of change, you must confirm that the potential for data loss is acceptable.
See “Creating Properties” on page 42 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.

Creating Derived Properties

➢ To create a derived 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.
5 Click ✅.
6 Select the Parameters tab to define the formula for the derived property.
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**.
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>
<thead>
<tr>
<th>Validation Class</th>
<th>Level</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic hierarchy fail</td>
<td>Hierarchy</td>
<td>Automatically fails at hierarchy level for testing purposes</td>
<td>none</td>
</tr>
<tr>
<td>Automatic node fail</td>
<td>Global Node</td>
<td>Automatically fails nodes at the version level for testing purposes</td>
<td>none</td>
</tr>
<tr>
<td>Automatic version fail</td>
<td>Version</td>
<td>Automatically fails at the version level for testing purposes</td>
<td>none</td>
</tr>
<tr>
<td>Date range check</td>
<td>Node</td>
<td>Verifies that the From Date is earlier than or equal to the To Date</td>
<td>From Date Property, To Date Property</td>
</tr>
</tbody>
</table>

Table 5  Validation Classes
<table>
<thead>
<tr>
<th>Validation Class</th>
<th>Level</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fails If Prop = TRUE in hierarchy</td>
<td>Node</td>
<td>Verifies that the specified boolean property has no True values in the specified hierarchy</td>
<td>Property, Hierarchy</td>
</tr>
<tr>
<td>Finds branches where property is not set</td>
<td>Node</td>
<td>Verifies that the specified property is set at least once on a specified branch</td>
<td>Property</td>
</tr>
<tr>
<td>Formula</td>
<td>Node</td>
<td>Verifies a node using business logic expressed in a formula. A formula result of False results in a validation failure.</td>
<td>Formula</td>
</tr>
<tr>
<td>Global property query validation</td>
<td>Global Node</td>
<td>Verifies using predefined query and expected result</td>
<td>Property query name, Failure value</td>
</tr>
<tr>
<td>Hierarchy contains a reference of a node when bool property is true</td>
<td>Node</td>
<td>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.</td>
<td>Hierarchy name, Boolean property for all nodes, Boolean property for leaf nodes</td>
</tr>
<tr>
<td>Hierarchy contains all where Prop = TRUE</td>
<td>Global Node</td>
<td>Verifies that the specified hierarchy contains all nodes where the specified property is True</td>
<td>Hierarchy, Property</td>
</tr>
<tr>
<td>Hierarchy contains all where Prop = value</td>
<td>Global Node</td>
<td>Verifies that the specified hierarchy contains all nodes for which the specified property has the specified value</td>
<td>Hierarchy, Property, Value</td>
</tr>
<tr>
<td>Invalid name length</td>
<td>Node</td>
<td>Verifies that the node name is not equal to a specified length.</td>
<td>Length</td>
</tr>
<tr>
<td>Limbs without any children</td>
<td>Node</td>
<td>Verifies that all limb nodes have children</td>
<td>none</td>
</tr>
<tr>
<td>Local property query validation</td>
<td>Node</td>
<td>Verifies using predefined query and expected result Only a local property query can be used.</td>
<td>Property query name, Failure value</td>
</tr>
<tr>
<td>Maximum # of nodes in a hierarchy</td>
<td>Hierarchy</td>
<td>Verifies that the number of nodes in the hierarchy does not exceed specified limit</td>
<td>Maximum number of nodes</td>
</tr>
<tr>
<td>Maximum # of nodes in a version</td>
<td>Version</td>
<td>Verifies that the number of nodes in the version does not exceed specified limit</td>
<td>Maximum number of nodes</td>
</tr>
<tr>
<td>Maximum child nodes</td>
<td>Version</td>
<td>Verifies that the number of children per node do not exceed specified limit</td>
<td>Maximum number of children</td>
</tr>
<tr>
<td>Merge node equivalency validation</td>
<td>Merge</td>
<td>Verifies that the affected node and merge node have the same value for the specified property</td>
<td>Global node property</td>
</tr>
<tr>
<td>Merge node property overridden validation</td>
<td>Merge</td>
<td>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)</td>
<td>Property</td>
</tr>
<tr>
<td>No default values allowed</td>
<td>Node</td>
<td>Verifies that no default values are used for the specified property</td>
<td>Property</td>
</tr>
<tr>
<td>No TRUE value on a branch</td>
<td>Node</td>
<td>Verifies that the specified boolean property is set to True at least once on a specified branch</td>
<td>Property</td>
</tr>
<tr>
<td>Node fail random</td>
<td>Node</td>
<td>Automatically fails the specified percentage of nodes for testing purposes</td>
<td>Failure percentage</td>
</tr>
<tr>
<td>Validation Class</td>
<td>Level</td>
<td>Description</td>
<td>Parameters</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>Property equals value</td>
<td>Node</td>
<td>Fails for all nodes for which the specified property equals the specified value</td>
<td>Property, Value</td>
</tr>
<tr>
<td>Property equivalency when a third boolean prop is true</td>
<td>Node</td>
<td>Property equivalency when a third boolean property is True.</td>
<td>Boolean property to evaluate, First Property, Second Property</td>
</tr>
<tr>
<td>Property length check</td>
<td>Node</td>
<td>Verifies that the specified property is at least minimum length and no more than maximum length</td>
<td>Property, Minimum Length, Maximum Length</td>
</tr>
<tr>
<td>Property set only once per branch</td>
<td>Node</td>
<td>Verifies that the specified property is set only once per branch</td>
<td>Property</td>
</tr>
<tr>
<td>Remove property validation</td>
<td>Remove</td>
<td>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).</td>
<td>Property1, Property2, Property3, Value1, Value2, Value3</td>
</tr>
<tr>
<td>Required fields</td>
<td>Node</td>
<td>Verifies that, for all nodes for which the specified property has a specified value, each property in the required list has a value:</td>
<td>Property, Value, Reject Default Records, Required Properties</td>
</tr>
<tr>
<td>Single TRUE value on a branch</td>
<td>Node</td>
<td>Verifies that the specified boolean property is set to True only once per branch</td>
<td>Property</td>
</tr>
<tr>
<td>Unique 2 properties within a version</td>
<td>Global Node</td>
<td>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)</td>
<td>First property, Second property, Include Defaults</td>
</tr>
<tr>
<td>Unique property value within a branch</td>
<td>Node</td>
<td>Verifies that the specified property has unique value within a branch</td>
<td>Property</td>
</tr>
<tr>
<td>Unique property value within a hierarchy</td>
<td>Node</td>
<td>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)</td>
<td>Property, Include Defaults</td>
</tr>
<tr>
<td>Unique property value within a version</td>
<td>Global Node</td>
<td>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)</td>
<td>Property, Include Defaults</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Level</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Reviews node relationships and properties to ensure criteria are met.</td>
<td>Use to determine whether a node level string property value has a valid length.</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Reviews properties in a hierarchy to ensure criteria are met. Can be assigned and run at the hierarchy or version levels.</td>
<td>Use to ensure that a hierarchy has no more than 10,000 nodes</td>
</tr>
<tr>
<td>Version</td>
<td>Reviews the properties of a version.</td>
<td>Use to ensure that a version contains no more than 100,000 nodes</td>
</tr>
<tr>
<td>Global node</td>
<td>Assigned at a version level. Validates every node in the version regardless of hierarchy, including orphans. Only properties defined as global are reviewed.</td>
<td>Use to ensure that all nodes within a version have a unique property value.</td>
</tr>
<tr>
<td>Merge</td>
<td>Runs when an operation requiring a merge (for example, a delete or an inactivate) is performed. Assigned at the version level.</td>
<td>Use to ensure that a leaf node is merged only into another leaf node.</td>
</tr>
<tr>
<td>Move</td>
<td>A validation triggered when an attempt is made to move a node. Assigned at the hierarchy level.</td>
<td>Use to prevent moving of cost centers within a hierarchy.</td>
</tr>
<tr>
<td>Remove</td>
<td>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.</td>
<td>Use to prevent the deletion of cost center nodes from a hierarchy</td>
</tr>
</tbody>
</table>

### 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.

For more information on creating formulas, see Managing Formulas.

9 Click ✍️.

---

**Assigning Validations**

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

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

For information on assigning validations to domains, see Chapter 4, “Managing Domains.” For information on assigning validations to versions, hierarchies, and nodes, 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 validation, 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.
Managing Formulas

In This Chapter

- Working with Functions ................................................................. 55
- Formula Evaluation ........................................................................ 59
- Considerations for Using Formulas .................................................. 61
- Function Definitions ........................................................................ 63
- Function Groups ............................................................................ 102

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

For more information, see:

- Creating Derived Properties
- Managing Validations

Working with 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 56
- “Literals” on page 56
- “Function Definitions” on page 63
- “Format String Parameter” on page 56
- “Date-Time Format Strings” on page 58

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

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

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

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

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

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

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

**Literals**

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

**Format String Parameter**

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

Format specifiers use the following form:

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

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

<table>
<thead>
<tr>
<th>Type Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| d         | Decimal  
The property value must be an integer. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros. |
| u         | Unsigned decimal  
Similar to d but no sign is output. |
| e         | Scientific  
The property value must be a floating-point value. The value is converted to a string of the form "-d.ddd...E+ddd". The resulting string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The total number of digits in the resulting string (including the one before the decimal point) is given by the precision specifier in the format string; a default precision of 15 is assumed if no precision specifier is present. The "E" exponent character in the resulting string is always followed by a plus or minus sign and at least three digits. |
| f         | Fixed  
The property value must be a floating-point value. The value is converted to a string of the form "-ddd.ddd...". The resulting string starts with a minus sign if the number is negative. The number of digits after the decimal point is given by the precision specifier in the format string; a default of two decimal digits is assumed if no precision specifier is present. |
| g         | General  
The property value must be a floating-point value. The value is converted to the shortest possible decimal string using fixed or scientific format. The number of significant digits in the resulting string is given by the precision specifier in the format string; a default precision of 15 is assumed if no precision specifier is present. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision, and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format. |
| n         | Number  
The property value must be a floating-point value. The value is converted to a string of the form "-d.ddd,ddd,ddd...". The "n" format corresponds to the "f" format, except that the resulting string contains thousand separators. |
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>The property value must be a floating-point value. The value is converted to a string that represents a currency amount. The conversion is controlled by the CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparator, DecimalSeparator, and CurrencyDecimals global variables or their equivalent in a TFormatSettings data structure. If the format string contains a precision specifier, it overrides the value given by the CurrencyDecimals global variable or its TFormatSettings equivalent.</td>
</tr>
<tr>
<td>s</td>
<td>The property value must be a character, a string, or a PChar value. The string or character is inserted in place of the format specifier. The precision specifier, if present in the format string, specifies the maximum length of the resulting string. If the property value is a string that is longer than this maximum, the string is truncated.</td>
</tr>
<tr>
<td>x</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Date followed by time</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The time is not displayed if the date-time value indicates midnight precisely.</td>
</tr>
<tr>
<td>d</td>
<td>Day as a number without a leading zero (1–31)</td>
</tr>
<tr>
<td>dd</td>
<td>Day as a number with a leading zero (01–31)</td>
</tr>
<tr>
<td>ddd</td>
<td>Day as an abbreviation (Sun-Sat)</td>
</tr>
<tr>
<td>dddd</td>
<td>Day as a full name (Sunday-Saturday)</td>
</tr>
<tr>
<td>dddddd</td>
<td>Short format of date</td>
</tr>
<tr>
<td>ddddd</td>
<td>Long format of date</td>
</tr>
<tr>
<td>e</td>
<td>Year in the current period/era as a number without a leading zero (Japanese, Korean, and Taiwanese locales only)</td>
</tr>
<tr>
<td>ee</td>
<td>Year in the current period/era as a number with a leading zero (Japanese, Korean, and Taiwanese locales only)</td>
</tr>
<tr>
<td>g</td>
<td>Period/era as an abbreviation (Japanese and Taiwanese locales only)</td>
</tr>
<tr>
<td>gg</td>
<td>Period/era as a full name (Japanese and Taiwanese locales only)</td>
</tr>
<tr>
<td>Specifier</td>
<td>Display</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>m</strong></td>
<td>Month as a number without a leading zero (1–12)</td>
</tr>
<tr>
<td><strong>mm</strong></td>
<td>Month as a number with a leading zero (01–12)</td>
</tr>
<tr>
<td><strong>mmm</strong></td>
<td>Month as an abbreviation (Jan-Dec)</td>
</tr>
<tr>
<td><strong>mmmm</strong></td>
<td>Month as a full name (January-December)</td>
</tr>
<tr>
<td><strong>yy</strong></td>
<td>Year as a two-digit number (00–99)</td>
</tr>
<tr>
<td><strong>yyyy</strong></td>
<td>Year as a four-digit number (0000–9999)</td>
</tr>
<tr>
<td><strong>h</strong></td>
<td>Hour without a leading zero (0–23)</td>
</tr>
<tr>
<td><strong>hh</strong></td>
<td>Hour with a leading zero (00–23)</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>Minute without a leading zero (0–59)</td>
</tr>
<tr>
<td><strong>nn</strong></td>
<td>Minute with a leading zero (00–59)</td>
</tr>
<tr>
<td><strong>s</strong></td>
<td>Second without a leading zero (0–59)</td>
</tr>
<tr>
<td><strong>ss</strong></td>
<td>Second with a leading zero (00–59)</td>
</tr>
<tr>
<td><strong>z</strong></td>
<td>Millisecond without a leading zero (0–999)</td>
</tr>
<tr>
<td><strong>zzz</strong></td>
<td>Millisecond with a leading zero (000–999)</td>
</tr>
<tr>
<td><strong>t</strong></td>
<td>Time using the format given by the ShortTimeFormat global variable</td>
</tr>
<tr>
<td><strong>tt</strong></td>
<td>Time using the format given by the LongTimeFormat global variable</td>
</tr>
<tr>
<td><strong>am/pm</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>a/p</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>ampm</strong></td>
<td>Uses the 12-hour clock for the preceding “h” or “hh” specifier</td>
</tr>
<tr>
<td>/</td>
<td>Date separator character given by the regional settings</td>
</tr>
<tr>
<td>:</td>
<td>Time separator character given by the regional settings</td>
</tr>
<tr>
<td>‘xx’/”xx”</td>
<td>Characters enclosed in single or double quotation marks are displayed as-is and do not affect formatting.</td>
</tr>
</tbody>
</table>

**Formula Evaluation**

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

Formulas are evaluated left to right, with evaluation of functions and string literals performed as they are encountered. By this method, nested functions are evaluated before additional parameters that are displayed to the right of the nested function. Functions can be nested explicitly in the formula or they can be implicitly nested by retrieving the value of another formula property. Circular references (property formulas that refer to the property itself, either explicitly or implicitly) should be avoided in most cases. 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.

Considerations for Using Formulas

The following section provide additional considerations for using formulas.

Data Type Conversion

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

Property Level Restrictions

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

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

Referencing Properties from Other Nodes

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

- NodePropValue
- ParentPropValue
- HierNodePropValue
- AncestorProp
- DualAncestorProp
- AscNodeProp
Referencing Local Node Properties from Global Node Properties

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

Nesting Functions

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

Using Properties as Variables for Other Properties

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

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

Using Recursion to Traverse Hierarchy Relationships

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

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

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

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

**Function Definitions**

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

**Abbrev**

**Description**

Returns the name (Abbrev) of the current node.

**Syntax**

Abbrev(): String

**Example**

Abbrev()

Return value is the name of the node.

**Add**

**Description**

Adds two specified integer values and returns the result.

**Syntax**

Add(Int1,Int2:Integer):Integer
Example
Add(1,4)

Return value is 5.

**AddedBy**

**Description**
Returns the value of the Added By change tracking property.

**Syntax**
AddedBy():String

**Example**
AddedBy()

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

**AddedOn**

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

**Syntax**
AddedOn():Date/Time

**Example**
AddedOn()

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

**AddFloat**

**Description**
Adds two specified float values and returns the result.

**Syntax**
AddFloat(Float1,Float2:Float):Float

**Example**
AddFloat(2.14,3.75)
The return value is 5.89.

**AncestorProp**

**Description**

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

**Syntax**

AncestorProp(Operator:String, Property:String, Value:String, FromTop:Boolean, ReturnProp:String)

**And**

**Description**

Returns True if all specified Boolean expressions evaluate to True.

**Syntax**

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

**Example**

And(1,T,True)

Return value is True.

**ArrayCount**

**Description**

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

**Syntax**

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

**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 (array). Returns zero (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 (array) 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

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

Syntax

AscNodeProp(LookUpProperty,ReturnProperty)

**AvgList**

Description

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

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

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

Example

AvgList(1;2;3,;,Integer)

Return value is 2.

**BooToStr**

Description

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

Syntax

BooToStr(Expression: Boolean): String

Example

BooToStr(1)

Return value is True.

**Changed**

Description

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

Syntax

Changed()

**ChangedBy**

Description

Returns the value of the Changed By change tracking property as a date/time.

Syntax

ChangedBy(): String
Example

ChangedBy()

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

**ChangedOn**

**Description**

Returns the value of the Changed On change tracking property.

**Syntax**

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

**Example**

```
ChangedOn()
```

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

**Concat**

**Description**

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

**Syntax**

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

**Example**

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

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

**ConcatWithDelimiter**

**Description**

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

**Syntax**

```
ConcatWithDelimiter(Delimiter, SkipBlanks, Items)
```

SkipBlanks is a boolean parameter that indicates whether to skip blank values in the list of strings. Valid values are 1, 0, T, F, t, f. Items is the list of strings to concatenate.
Example
ConcatWithDelimiter(,,1, Item1, Item2, Item3, Item4)

Return value is Item1; Item2; Item3; Item4.

**Decode**

**Description**


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

**Syntax**

`Decode(CodedString: String): String`

**DefaultProp**

**Description**

Returns the default value for the property.

**Syntax**

`DefaultProp(Property: String)`

**Descr**

**Description**

Returns the description of the current node.

**Syntax**

`Descr(): String`

**Example**

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

Description
Divides two specified integer values and returns the result.

Syntax
Divide(Int1, Int2: Integer): Integer

Example
Divide(200, 10)
Return value is 20.

**DivideFloat**

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

Syntax
Divide(Float1, Float2: Float): Float

Example
DivideFloat(2.535, 1.5)
The return value is 1.69.

**DualAncestorProp**

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

Syntax
DualAncestorProp(Operator1, Property1, Value1, Operator2, Property2, Value2, FromTop, ReturnProp)

**Equals**

Description
Returns True if two specified values are equal.
Syntax
Equals(ParamType,Param1,Param2: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 float value converted to a string. Returns zero (0) if the input value does not represent a float.

Syntax
FloatToStr(Float1:Float):String

Example
FloatToStr(1.001)
Return value is 1.001.

Format

Description
Formats the value using a specified format string parameter type identifier and parameter value of the specified type. This function is limited to one value parameter.
Syntax
Format(Format,ParamType, ValueToFormat:String):String

Example
Format('%8.2f',Float,123.456)
Return value is 123.46.

FormattedDate

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

Syntax
FormattedDate(PropertyName,FormatString)

GreaterThan

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

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

Example
GreaterThan(1,2)
The return value is False.

GreaterThanOrEqual

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

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

Example
GreaterThanOrEqual(2,2)
The return value is True.

**HasCharacters**

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

**Syntax**
HasCharacters(Input:String,CharList:String):Boolean

**HasChildWith**

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

**Syntax**
HasChildWith(Expression:Boolean):Boolean

**Example**
HasChildWith(GreaterThan(ID(),200))

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

**HasParentNode**

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

**Syntax**
HasParentNode():Boolean

**Example**
HasParentNode()

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

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

Syntax
HasSiblingWith(Expression:Boolean):Boolean

Example
HasSiblingWith(PropValue(Leaf))
If any of the children are leaves, then the return value is True.

HierNodePropValue

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

Syntax
HierNodePropValue(HierAbbrev,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 ID of the current node.

Syntax
ID():Integer

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

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

Syntax
If(Expression:Boolean, TrueResult,FalseResult: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.

InheritedPropOrigin

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

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

Example
InheritedPropOrigin(Custom.AccountType,Abbrev())

InRange

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

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

• Length check - string
Range check - integer, float, datetime, date, time

DataType can be one of these types:

- String
- Integer
- Float
- Datetime
- Date
- Time

**Syntax**

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

**Example**

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

Return True.

**InternalPrefix**

**Description**

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

**Syntax**

```
InternalPrefix()
```

**Intersection**

**Description**

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

**Syntax**

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

**Example**

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

The return value is A,C,E.
**IntToStr**

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

**Syntax**
IntToStr(Int1:Integer):String

**Example**
IntToStr(12345)
The return value is 12345.

**InvertedLevel**

**Description**
Returns the maximum depth of descendants below the current node.

**Syntax**
InvertedLevel()

**IsAlpha**

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

**Syntax**
IsAlpha(String:String):Boolean

**Example**
IsAlpha(A23D)
The return value is False.

**IsAlphaNumeric**

**Description**
Returns True if the specified string contains only alphabetical or numeric characters (case-insensitive). If AllowBlanks is True, then blank spaces are considered alphanumerical.
Syntax
IsAlphaNumeric(String:String,AllowBlanks:Boolean):Boolean

Example
IsAlphaNumeric(ABC123,True)
Returns True.

IsBlank

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

Syntax
IsBlank(Input:String):Boolean

Example
IsBlank(Descr())
Returns True if the node description is blank.

IsBottomNode

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

Syntax
IsBottomNode(Node:String):Boolean

Example
IsBottomNode(Abbrev())
Returns True if the node does not have children.

IsDataType

Description
Returns True if the input value matches the specified data type. Valid data types are:

- String
- Integer
- Float
- Datetime
- Date
- Time

**Syntax**

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

**Example**

IsDataType(123, Integer)

Returns True.

**IsDefinedPropVal**

**Description**

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

**Syntax**

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

**Example**

IsDefinedPropVal(Custom.AccountType, Abbrev())

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

**IsNodeAbove**

**Description**

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

**Syntax**

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

**Example**

IsNodeAbove(Parent, Child)

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

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

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

Example
IsNodeBelow(Child,Parent)
Returns True if node child is descendant of the parent node.

IsNumeric

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

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

Example
IsNumeric(12345)
The return value is True.

IsRangeListSubset

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

Syntax

Length

Description
Returns the number of characters in the specified string value.
**Syntax**

Length(String:String):Integer

**Example**

Length(Desc())

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

---

**LessThan**

**Description**

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

**Syntax**

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

**Example**

LessThan(1,2)

The return value is True.

---

**LessThanOrEqual**

**Description**

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

**Syntax**

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

**Example**

LessThanOrEqual(3,3)

The return value is True.

---

**ListAncestors**

**Description**

Returns a comma-delimited list of the names of the current node’s ancestors starting from the top node. Returns a blank string if the current node is not a local node. Supported sort order values:
- [hier] – Default value for local context. The list of nodes is returned in the standard hierarchy sort order for the current hierarchy.
- [alpha] – The list of nodes is returned sorted by node name.
- [nodeid] – Limited use for legacy compatibility. The list of nodes is returned sorted numerically on the node ID of each node in the return list.

**Syntax**

`ListAncestors(SortOrder:String):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 children for the current node. Supported sort order values:

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

**Syntax**

`ListChildren(SortOrder:String):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 True if the specified list contains the specified value.

**Syntax**

`ListContains(List:String, Item:String, Delimiter: String):Boolean`
Example
ListContains(PropValue(NodeList),Colas,;)
The return value is True.

ListDescendants

Description
Returns a comma-delimited list of descendants for the current node. Supported sort order values:

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

Syntax
ListDescendants(SortOrder:String):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.

ListDistinct

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

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

Example
ListDistinct(A;B:C;A;D,;)
The return value is A;B;C;D.
**ListNodePropValues**

**Description**

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

**Syntax**

```
```

**Example**

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

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

**ListNodesWith**

**Description**

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

**Syntax**

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

**Example**

```
ListNodesWith(100;200;300,;,NodeIsLeaf())
```

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

**ListRelatedNodesWith**

**Description**

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

- **Ancestors** – local properties can be referenced in the specified expression
- **Siblings** – local properties can be referenced in the specified expression
- **Children** – local and global properties can be referenced in the specified expression
- **Descendants** – local and global properties can be referenced in the specified expression

Supported sort order values are:

- **[hier]** – Default value for local context. The list of nodes is returned in the standard hierarchy sort order for the current hierarchy. Available in local node context only.
- \([\alpha]\) – The list of nodes is returned sorted by node name. Available in local and global node contexts.
- \([\text{nodeid}]\) – Limited use for legacy compatibility. The list of nodes is returned sorted numerically on the node ID of each node in the return list. Available in local and global node contexts.

Max is an integer value indicating the maximum number of nodes to be returned. If no value is specified, all nodes are returned.

**Syntax**

\[
\text{ListRelatedNodesWith}(\text{Relation}: \text{String}, \text{Expression}: \text{String}, \text{SortOrder}: \text{String}, \text{Max}: \text{Integer}): \text{String}
\]

**Example**

\[
\text{ListRelatedNodesWith}(\text{children}, \text{NodeIsLeaf}(), [\alpha], 1000)
\]

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

**ListSiblings**

**Description**

Returns a comma-delimited list of siblings (peers) of the current node. Supported sort order values:

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

**Syntax**

\[
\text{ListSiblings}(\text{SortOrder}: \text{String}): \text{String}
\]

**Example**

\[
\text{ListSiblings}()
\]

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**

Returns the specified string value converted to lower case.
Syntax
LowerCase(String: String): String

Example
LowerCase(HOBBES)

The return value is hobbes.

LTrim

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

Syntax
LTrim(AString: String): String

Example
LTrim("   101203")

The return value is 101203.

MaxList

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

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

Example
MaxList(1;2;3,,Integer)

Return value is 3.

MinList

Description
Returns the minimum item from the specified list, ignoring blank items. Returns a blank string if the list contains an item not of the specified type.
Syntax
MinList(InputList,Delimiter,ItemType)

Example
MinList(1;2;3,,Integer)
Return value is 1.

**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 floating-point numbers (float) 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 current node based on the sort order used for the current hierarchy.

**Syntax**

`NextSibling()`: String

**Example**

`NextSibling()`

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

**NodeAccessGroups**

**Description**

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

**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 node.

**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 system date and/or time.

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 for the current node where the specified expression evaluates to True.
Syntax
NumChildWith(Expression: Boolean): Integer

Example
NumChildWith(NodeIsLeaf())

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

**NumDescendantsWith**

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

Syntax
NumDescendantsWith(Expression: Boolean): Integer

Example
NumDescendantsWith(NodeIsLeaf())

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

**Or**

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

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

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

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

**OrigPropValue**

Description
Returns the value of the specified property for the originating node when using the HasSiblingWith or NumDescendantsWith functions.
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 original list is returned.

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 original list is returned.

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. Returns a blank string if the node has no parent, or if the current node is not a local node.

Syntax
ParentPropValue(PropAbbrev: String): String

Example
ParentPropValue(Abbrev)

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

**Pos**

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

Syntax
Pos(ASubString, AString: String): Integer

Example
Pos(D, ABCDEFG)

The return value is 4.

**PreviousSibling**

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

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 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 the current user.

**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 definition.

**Syntax**

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

**Example**

```
PropMaxValue(Volume)
```

The return value is 10.

**PropMinValue**

**Description**

Returns the minimum value of the specified property definition.

**Syntax**

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

**Example**

```
PropMinValue(Volume)
```

The return value is 1.

**PropValue**

**Description**

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

**Syntax**

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

**Example**

```
PropValue(Volume)
```

The return value is 2.
RangeListContains

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

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

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 verifies RangeList and combines ranges with contiguous boundaries.

ReplacementAbbrev

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

Syntax
ReplacementAbbrev(): String

Example
ReplacementAbbrev()

ReplacePropValue

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

Syntax
ReplacePropValue(PropAbbrev: String): String
Example
ReplacePropValue(Description)

**ReplaceStr**

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

**Syntax**
ReplaceStr(String: String,OldPattern: String,NewPattern: String,ReplaceAll: Boolean): String

**Example**
ReplaceStr(A1;A2;A3,A,B,True)
The return value is B1;B2;B3.

**RTrim**

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

**Syntax**
RTrim(String: String): String

**Example**
RTrim("100   ")
The return value is 100.

**SortList**

**Description**
Returns the specified list in a sorted order.

**Syntax**
SortList(InputList: String,Delimiter: String,IgnoreCase: Boolean,ItemType: String)
**StripPadChar**

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

**Syntax**

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

**Example**

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

The return value is 3333.

**StrToBool**

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

**Syntax**

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

**Example**

```
StrToBool(0)
```

The return value is False.

**StrToFloat**

**Description**
Returns the float value of the specified string. Returns zero (0) for a space or blank string. If the specified string does not represent a floating point number, an error is returned.

**Syntax**

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

**Example**

```
StrToFloat(11.101)
```
The return value is 11.101.

**StrToInt**

**Description**

Returns the integer value of the specified string. Returns zero (0) for a space or blank string. If the specified string does not represent an integer number, an error is returned.

**Syntax**

\[
\text{StrToInt}(\text{String}: \text{String}): \text{Integer}
\]

**Example**

\[
\text{StrToInt}(101)
\]

The return value is 101.

**Stuff**

**Description**

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

**Syntax**

\[
\text{Stuff}(	ext{PropAbbrev}: \text{String}, \text{CharsToReplace}: \text{String}, \text{ReplacementChars}: \text{String}): \text{String}
\]

**Example**

\[
\text{Stuff}(\text{Abbrev}(), \text{GEO}, \text{RIO})
\]

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**

\[
\text{SubString}(\text{String}: \text{String}, \text{Index}: \text{Integer}, \text{Count}: \text{Integer}): \text{String}
\]

**Example**

\[
\text{SubString}(\text{Colas}, 1, 2)
\]
The return value is Co.

**Subtract**

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

**Syntax**
Subtract(Minuend,Subtrahend: Integer): Integer

**Example**
Subtract(10,2)
The return value is 8.

**SubtractFloat**

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

**Syntax**
SubtractFloat(Minuend,Subtrahend: Float): Float

**Example**
SubtractFloat(8.09,3.76)
The return value is 4.33.

**SumList**

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

**Syntax**
SumList(InputList: String,Delimiter: String,ItemType: String):Integer

**Example**
SumList(1;2;3,;,Integer)
Return value is 6.
Trim

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

Syntax
Trim(String: String): String

Example
Trim("   101   ")
The return value is 101.

UpperCase

Description
Returns a string value converted to uppercase.

Syntax
UpperCase(String: String): String

Example
UpperCase(smaller)
The return value is SMALLER.

UserName

Description
Returns the user name for the current user.

Syntax
UserName(): String

Example
UserName()
Return value is the user name.
**XOr**

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

**Syntax**

\[
\text{XOr(} \text{Expression1: Boolean, Expression2: Boolean): Boolean}
\]

**Example**

\[
\text{XOr(NodeIsLeaf(), Equals(Integer, PropValue(Level), 3))}
\]

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

**Function Groups**

The following table groups functions by use.

<table>
<thead>
<tr>
<th>Function Group</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aggregate</strong></td>
<td>● AvgList</td>
</tr>
<tr>
<td></td>
<td>● MaxList</td>
</tr>
<tr>
<td></td>
<td>● MinList</td>
</tr>
<tr>
<td></td>
<td>● SumList</td>
</tr>
<tr>
<td><strong>Change Tracking</strong></td>
<td>● AddedBy</td>
</tr>
<tr>
<td></td>
<td>● AddedOn</td>
</tr>
<tr>
<td></td>
<td>● Changed</td>
</tr>
<tr>
<td></td>
<td>● ChangedBy</td>
</tr>
<tr>
<td></td>
<td>● ChangedOn</td>
</tr>
<tr>
<td></td>
<td>● Now</td>
</tr>
<tr>
<td><strong>Comparison</strong></td>
<td>● Equals</td>
</tr>
<tr>
<td></td>
<td>● GreaterThan</td>
</tr>
<tr>
<td></td>
<td>● GreaterThanOrEqual</td>
</tr>
<tr>
<td></td>
<td>● InRange</td>
</tr>
<tr>
<td></td>
<td>● IsBlank</td>
</tr>
<tr>
<td></td>
<td>● IsRangeListSubset</td>
</tr>
<tr>
<td></td>
<td>● LessThan</td>
</tr>
<tr>
<td></td>
<td>● LessThanOrEqual</td>
</tr>
<tr>
<td></td>
<td>● RangeListContains</td>
</tr>
<tr>
<td>Function Group</td>
<td>Functions</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>Conditional</td>
<td>• And</td>
</tr>
<tr>
<td></td>
<td>• If</td>
</tr>
<tr>
<td></td>
<td>• Not</td>
</tr>
<tr>
<td></td>
<td>• Or</td>
</tr>
<tr>
<td></td>
<td>• XOr</td>
</tr>
<tr>
<td>Data Type</td>
<td>• BoolToStr</td>
</tr>
<tr>
<td></td>
<td>• FloatToStr</td>
</tr>
<tr>
<td></td>
<td>• IntToStr</td>
</tr>
<tr>
<td></td>
<td>• IsDataType</td>
</tr>
<tr>
<td></td>
<td>• IsNumeric</td>
</tr>
<tr>
<td></td>
<td>• StrToBool</td>
</tr>
<tr>
<td></td>
<td>• StrToFloat</td>
</tr>
<tr>
<td></td>
<td>• StrToInt</td>
</tr>
<tr>
<td>List</td>
<td>• ArrayCount</td>
</tr>
<tr>
<td></td>
<td>• ArrayIndex</td>
</tr>
<tr>
<td></td>
<td>• ArrayItem</td>
</tr>
<tr>
<td></td>
<td>• Intersection</td>
</tr>
<tr>
<td></td>
<td>• ListContains</td>
</tr>
<tr>
<td></td>
<td>• ListDistinct</td>
</tr>
<tr>
<td></td>
<td>• ListNodePropValues</td>
</tr>
<tr>
<td></td>
<td>• ListNodesWith</td>
</tr>
<tr>
<td></td>
<td>• SortList</td>
</tr>
<tr>
<td>Mathematical</td>
<td>• Add</td>
</tr>
<tr>
<td></td>
<td>• AddFloat</td>
</tr>
<tr>
<td></td>
<td>• Divide</td>
</tr>
<tr>
<td></td>
<td>• DivideFloat</td>
</tr>
<tr>
<td></td>
<td>• Modulus</td>
</tr>
<tr>
<td></td>
<td>• Multiply</td>
</tr>
<tr>
<td></td>
<td>• MultiplyFloat</td>
</tr>
<tr>
<td></td>
<td>• Subtract</td>
</tr>
<tr>
<td></td>
<td>• SubtractFloat</td>
</tr>
<tr>
<td>Node</td>
<td>• Abbrev</td>
</tr>
<tr>
<td></td>
<td>• ID</td>
</tr>
<tr>
<td></td>
<td>• InternalPrefix</td>
</tr>
<tr>
<td></td>
<td>• NodeExists</td>
</tr>
<tr>
<td></td>
<td>• NodeInHier</td>
</tr>
<tr>
<td></td>
<td>• NodeIsLeaf</td>
</tr>
<tr>
<td>Function Group</td>
<td>Functions</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
</tr>
</tbody>
</table>
| Property       | - AncestorProp  
|                | - AscNodeProp    
|                | - DefaultProp    
|                | - Descr          
<p>|                | - DualAncestorProp |
|                | - HierNodePropValue |
|                | - InheritedPropOrigin |
|                | - IsDefinedPropVal |
|                | - NodePropValue   |
|                | - OrigPropValue   |
|                | - ParentPropValue |
|                | - PropControllingHier |
|                | - PropDefaultValue |
|                | - PropMaxValue    |
|                | - PropMinValue    |
|                | - PropValue       |
|                | - ReplacePropValue |
| Relationship   | - Children       |
|                | - HasChildWith    |
|                | - HasParentNode   |
|                | - HasSiblingWith  |
|                | - InvertedLevel   |
|                | - IsBottomNode    |
|                | - IsNodeAbove     |
|                | - IsNodeBelow     |
|                | - ListAncestors   |
|                | - ListChildren    |
|                | - ListDescendants |
|                | - ListRelatedNodesWith |
|                | - ListSiblings    |
|                | - NextSibling     |
|                | - NumChildWith    |
|                | - NumDescendantsWith |
|                | - PreviousSibling |
|                | - ReplacementAbbrev |</p>
<table>
<thead>
<tr>
<th>Function Group</th>
<th>Functions</th>
</tr>
</thead>
</table>
| String Manipulation | • Concat  
|                  | • ConcatWithDelimiter  
|                  | • Decode  
|                  | • FlipList  
|                  | • Format  
|                  | • FormattedDate  
|                  | • HasCharacters  
|                  | • IsAlpha  
|                  | • IsAlphaNumeric  
|                  | • Length  
|                  | • LowerCase  
|                  | • LTrim  
|                  | • PadChar  
|                  | • PadList  
|                  | • Pos  
|                  | • ReplaceStr  
|                  | • RTrim  
|                  | • StripPadChar  
|                  | • Stuff  
|                  | • SubString  
|                  | • Trim  
|                  | • UpperCase |
| User | • NodeAccessGroups  
|      | • PropertyCategories  
|      | • UserName |
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 a select the different PNG file.
5 Click Upload.
6 Click Save.

➤ To delete a glyph:
1 On the Home page, select Administer.
2 Under Metadata, expand Glyphs.
3 Select a glyph and click ✗.
4 Click Delete this Item to confirm the deletion.
In This Chapter

System Preferences ................................................................. 111
Configuring System Preferences ............................................ 117

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>
<thead>
<tr>
<th>System Preference</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowAsOf</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>AllowNextIDGeneration</td>
<td>Boolean</td>
<td>True enables automatic Next ID generation. Default value is False.</td>
</tr>
<tr>
<td>AllowNextIDKeyCreation</td>
<td>Role</td>
<td>List of roles allowed to create a new key in NextID feature. Default values are Interactive User, Data Creator, Data Manager.</td>
</tr>
<tr>
<td>AllowPru</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>AllowRelaxedMove</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>AllowSpac</td>
<td>Boolean</td>
<td>True allows spaces in node names. Default is True.</td>
</tr>
<tr>
<td>System Preference</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ApprovalGroups</td>
<td>String</td>
<td>Comma-delimited list of approval groups.</td>
</tr>
<tr>
<td>ApprovalGroupTrackProperties</td>
<td>String</td>
<td>Delimited list of approval properties tracked by groups.</td>
</tr>
<tr>
<td>ApprovalPropertyByApprovalGroup</td>
<td>String</td>
<td>Global boolean approval property by approval group.</td>
</tr>
<tr>
<td>AuthMethod</td>
<td>String</td>
<td>User authentication method:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Internal – Users are only authenticated within Data Relationship Management.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- CSS (External) – Users are only authenticated externally. Requires access to Shared Services.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Mixed – Users are authenticated internally or externally based on a setting for each individual user.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is Internal.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> A change to this preference requires a restart of the Data Relationship Management application.</td>
</tr>
<tr>
<td>CopyLcl</td>
<td>Boolean</td>
<td>True copies local values when a node is copied.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is True.</td>
</tr>
<tr>
<td>DefaultCurrentVersion</td>
<td>Version</td>
<td>Default current version. This preference can be set using the Make Default option for versions.</td>
</tr>
<tr>
<td>DefaultPreviousVersion</td>
<td>Version</td>
<td>Default previous version. This preference can be set using the Make Default option for versions.</td>
</tr>
<tr>
<td>DefaultPropCopyMode</td>
<td>String</td>
<td>Default property copy mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values are Overridden, Selected, and ForceAll.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is Overridden.</td>
</tr>
<tr>
<td>EnablePropCopyOptions</td>
<td>Role</td>
<td>List of roles allowed access to the property copy options.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default values are Interactive User, Data Creator, Data Manager.</td>
</tr>
<tr>
<td>EnforceListProps</td>
<td>Boolean</td>
<td>True allows updates to a List Property with values from the pre-defined list only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is True.</td>
</tr>
<tr>
<td>FiltrChr</td>
<td>String</td>
<td>Set of characters for the Replace function on the Output Option screen of exports.</td>
</tr>
<tr>
<td>FindByProperties</td>
<td>Property</td>
<td>List of properties available to search with when browsing a hierarchy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The properties displayed are those to which a user has access. Also, the properties displayed may not be applicable to all hierarchies.</td>
</tr>
<tr>
<td>FindWildCardAppend</td>
<td>Boolean</td>
<td>True appends an asterisk (*) to the Find criteria when Exact Match is not selected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is False.</td>
</tr>
<tr>
<td>FindWildCardPrepend</td>
<td>Boolean</td>
<td>True prepends an asterisk (*) to the Find criteria when Exact Match is not selected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is False.</td>
</tr>
<tr>
<td>System Preference</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GlobalPropLocalOverride</td>
<td>Property</td>
<td>List of properties to exclude from local checks on global properties. These are used when GlobalPropLocalSecurity is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> A change to this preference requires a restart of the Data Relationship Management application.</td>
</tr>
<tr>
<td>GlobalPropLocalSecurity</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is False.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> A change to this preference requires a restart of the Data Relationship Management application.</td>
</tr>
<tr>
<td>HierSep</td>
<td>String</td>
<td>Hierarchy and node separator character.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is tilde (~).</td>
</tr>
<tr>
<td>IdleTime</td>
<td>Integer</td>
<td>Number of minutes to session time out on the application server.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is 60.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> A change to this preference requires a restart of the Data Relationship Management application.</td>
</tr>
<tr>
<td>Inactivate</td>
<td>Role</td>
<td>List of user roles allowed to inactivate nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is all roles.</td>
</tr>
<tr>
<td>InactiveChanges</td>
<td>Role</td>
<td>List of roles allowed to change inactive nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default values are Data Manager, Application Administrator, Access Manager.</td>
</tr>
<tr>
<td>InvDescr</td>
<td>String</td>
<td>List of invalid characters for node description property.</td>
</tr>
<tr>
<td>InvName</td>
<td>String</td>
<td>List of invalid characters for node name.</td>
</tr>
<tr>
<td>LeafEdit</td>
<td>Role</td>
<td>List of roles allowed to change the Leaf property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default values are Data Manager, Data Creator, Application Administrator, Access Manager.</td>
</tr>
<tr>
<td>LockoutInactivity</td>
<td>Integer</td>
<td>Maximum number of days of inactivity before a user is locked out.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is 30; zero indicates no maximum.</td>
</tr>
<tr>
<td>LockoutInvalidLogins</td>
<td>Integer</td>
<td>Maximum number of invalid logins before a user is locked out.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is 6; zero indicates no maximum.</td>
</tr>
<tr>
<td>System Preference</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LossLevel</td>
<td>String</td>
<td>Loss level to capture. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is Defined. Selecting All can significantly impact system performance for removed or deleted nodes with many property values.</td>
</tr>
<tr>
<td>MaxDescr</td>
<td>Integer</td>
<td>Maximum number of characters for node description. Valid values are 12 to 255. Default value is 80.</td>
</tr>
<tr>
<td>MaxLeaf</td>
<td>Integer</td>
<td>Maximum number of characters for the leaf name. Valid values are 3 to 20. Default value is 255.</td>
</tr>
<tr>
<td>MaxLimb</td>
<td>Integer</td>
<td>Maximum number of characters for the limb name. Valid values are 3 to 20. Default value is 255.</td>
</tr>
<tr>
<td>PasswordDuration</td>
<td>Integer</td>
<td>Number of days that a user password is valid. Valid values are 1 to 9999. Default value is 30.</td>
</tr>
<tr>
<td>PasswordMaxLength</td>
<td>Integer</td>
<td>Maximum number of characters for user password. Valid values are 0 to 255. Zero indicates no minimum. Default value is zero.</td>
</tr>
<tr>
<td>PasswordMinLength</td>
<td>Integer</td>
<td>Minimum number of characters for user password. Valid values are 0 to 9999. Zero indicates no minimum. Default value is 6.</td>
</tr>
<tr>
<td>PasswordPolicyEnabled</td>
<td>Boolean</td>
<td>True requires the password to contain three of the following elements:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Uppercase letters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Lowercase letters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Numbers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Special characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value is True.</td>
</tr>
<tr>
<td>PasswordWarningPeriod</td>
<td>Integer</td>
<td>Positive or negative number to indicate how many days before (-) or after (+) the password expiration date to warn users to change their password before no longer allowing them to log in. Valid values are -30 to 30. Default value is 1.</td>
</tr>
<tr>
<td>RenameLeaf</td>
<td>Role</td>
<td>List of roles allowed to rename leaf nodes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default values are Data Manager, Application Administrator, Access Manager.</td>
</tr>
<tr>
<td>System Preference</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RenameLimb</td>
<td>Role</td>
<td>List of roles allowed to rename limb nodes. Default value is all roles.</td>
</tr>
<tr>
<td>ReqMerge</td>
<td>Boolean</td>
<td>True requires merge for inactivates or deletes when UseMerge is enabled. Default value is False.</td>
</tr>
<tr>
<td>SharedNodeDelimiter</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>SharedNodeIdentifier</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>SharedNodeMaintenanceEnabled</td>
<td>Boolean</td>
<td>True enables shared nodes. Default value is False. Note: A change to this preference requires a restart of the Data Relationship Management application.</td>
</tr>
<tr>
<td>SharedNodeNamingType</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>SharedNodeSequenceLength</td>
<td>Integer</td>
<td>Specifies the length of the uniqueness key when using numeric sequence type. Default is 3. Note: A change to this preference requires a restart of the Data Relationship Management application.</td>
</tr>
<tr>
<td>SharedNodeSequenceSeparator</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>SharedNodeSequenceType</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>SortLimbsFirst</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>System Preference</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TopNodeParentString</td>
<td>String</td>
<td>Used in Import and Export to denote parent value for a top node. Default value is None.</td>
</tr>
<tr>
<td>TransactionLevels</td>
<td>String</td>
<td>List of transaction levels to capture. Turning on As-Of or specifying result or loss actions forces core actions to be captured. Valid values are: Logged Action, Core Action, Result Action, Loss Action. Default values are Logged Action, Core Action, Result Action, Loss Action. <strong>Note:</strong> A change to this preference requires a restart of the Data Relationship Management application.</td>
</tr>
<tr>
<td>UpName</td>
<td>Boolean</td>
<td>True uses uppercase always for the node name. Default value is False.</td>
</tr>
<tr>
<td>UseChangeApproval</td>
<td>Boolean</td>
<td>True enables change approval. Default value is False.</td>
</tr>
<tr>
<td>UseMerge</td>
<td>Boolean</td>
<td>True enables use of Merge methodology for inactivated and deleted nodes. <strong>Note:</strong> 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.</td>
</tr>
<tr>
<td>ValSec</td>
<td>Boolean</td>
<td>True checks node access group security to determine whether a user can run batch validations for a node. Default value is False.</td>
</tr>
<tr>
<td>WarnHL</td>
<td>Integer</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 turn 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 Edit.
  4. Modify the value and click Save.
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.

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.
Migrating Data Relationship Management Metadata

In This Chapter

- Opening the Migration Utility ................................................................. 122
- Extracting Metadata ............................................................................. 122
- Loading Metadata .................................................................................. 123
- Comparing Metadata ............................................................................ 124
- Viewing Metadata ................................................................................... 125
- Generating Reports ............................................................................... 126

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)
- Domains
- Version Variables
Opening the Migration Utility

By default, the Migration Utility is installed to:

`MIDDLEWARE_HOME\EPMS\System11R1\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 123.
- Click Generate Reports for the Metadata File to generate a report from the XML file. For more information, see “Generating Reports” on page 126.

### 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.

Note: Before loading a metadata file, it is recommended that you perform an extract of existing metadata in case you want to revert back to the previous configuration. It is also a good idea to perform a database backup before loading metadata, particularly if you are loading a migration file into a production environment.

To load metadata from an XML file into a Data Relationship Management application:

1 On the Main Menu, click Load.

2 Click Browse, select the XML file that you want to load, and click Upload.
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.
Click Next.

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.

Click Create Difference File.

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 application. For more information, see “Loading Metadata” on page 123.
- Click Generate Reports for the Metadata File to generate a report from the XML file. For more information, see “Generating Reports” on page 126.

**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, 63
Add function, 63
AddedBy function, 64
AddedOn function, 64
AddFloat function, 64
adding node glyphs, 108
AncestorProp function, 65
And function, 65
ArrayCount function, 65
ArrayIndex function, 66
ArrayItem function, 66
AscNodeProp function, 66
assigning validations, 53
AvgList function, 66

B
BoolToStr function, 67

C
change approval
setting up, 116
Changed function, 67
ChangedBy function, 67
ChangedOn function, 68
comparing metadata, 124
Concat function, 68
ConcatWithDelimiter function, 68
configuring system preferences, 117
creating
  domains, 33
  node access groups, 30
  properties, 42
  property categories, 38
  users, 23
  validations, 52

dualAncestorProp function, 70

dominations
creating, 33
delting, 34
digiting, 34
DualAncestorProp function, 70

E
digiting
  domains, 34
  external connections, 120
  node access groups, 31
  node types, 108
  property categories, 39
  users, 26
  validations, 54

\textit{Derivative functions}
creating, 47

Descr function, 69

Divide function, 70
DivideFloat function, 70
domains
creating, 33
delting, 34
eistinguishing, 34

DualAncestorProp function, 70


Equals function, 70

D
Decode function, 69
DefaultProp function, 69
defining
  external connections, 119
  node types, 107
delting
  domains, 34
  external connections, 120
  node access groups, 31
  node glyphs, 108
  node types, 108
  properties, 47
  property categories, 39
  users, 26
  validations, 54
derived properties
creating, 47

Descr function, 69
Divide function, 70
DivideFloat function, 70
domains
creating, 33
delting, 34
eistinguishing, 34

DualAncestorProp function, 70


Equals function, 70

D
Decode function, 69
DefaultProp function, 69
defining
  external connections, 119
  node types, 107
delting
  domains, 34
  external connections, 120
  node access groups, 31
  node glyphs, 108
  node types, 108
  properties, 47
  property categories, 39
  users, 26
  validations, 54
derived properties
creating, 47

Descr function, 69
Divide function, 70
DivideFloat function, 70
domains
creating, 33
delting, 34
eistinguishing, 34

DualAncestorProp function, 70


Equals function, 70

D
Decode function, 69
DefaultProp function, 69
defining
  external connections, 119
  node types, 107
delting
  domains, 34
  external connections, 120
  node access groups, 31
  node glyphs, 108
  node types, 108
  properties, 47
  property categories, 39
  users, 26
  validations, 54
derived properties
creating, 47

Descr function, 69
Divide function, 70
DivideFloat function, 70
domains
creating, 33
delting, 34
eistinguishing, 34

DualAncestorProp function, 70


Equals function, 70
external connections
defining, 119
deleting, 120
editing, 120
extracting metadata, 122

F
FlipList function, 71
FloatToStr function, 71
Format function, 71
FormattedDate function, 72
formula evaluation, 59
formula functions, 55
  Abbrev, 63
  Add, 63
  AddedBy, 64
  AddedOn, 64
  AddFloat, 64
  AncestorProp, 65
  And, 65
  ArrayCount, 65
  ArrayIndex, 66
  ArrayItem, 66
  AscNodeProp, 66
  AvgList, 66
  BoolToStr, 67
  Changed, 67
  ChangedBy, 67
  ChangedOn, 68
  Concat, 68
  ConcatWithDelimiter, 68
data-time format strings, 58
Decode, 69
DefaultProp, 69
Descrip, 69
Divide, 70
DivideFloat, 70
DualAncestorProp, 70
Equals, 70
FlipList, 71
FloatToStr, 71
Format, 71
FormattedDate, 72
Greater, 72
GreaterThan, 72
GreaterThanOrEqual, 72
grouped by use, 102
HasCharacters, 73
HasChildWith, 73
HasParentNode, 73
HasSiblingWith, 74
HierNodePropValue, 74
ID, 74
If, 75
InheritedPropOrigin, 75
InRange, 75
InternalPrefix, 76
Intersection, 76
IntToStr, 77
InvertedLevel, 77
IsAlpha, 77
IsAlphaNumeric, 77
IsBlank, 78
IsButtonDown, 78
IsDataType, 78
IsDefinedPropVal, 79
IsNodeAbove, 79
IsNodeBelow, 80
IsNumeric, 80
IsRangeListSubset, 80
Length, 80
LessThan, 81
LessThanOrEqual, 81
ListAncestors, 81
ListChildren, 82
ListContains, 82
ListDescendants, 83
ListDistinct, 83
ListNodePropValues, 84
ListNodesWith, 84
ListRelatedNodesWith, 84
ListSiblings, 85
LowerCase, 85
LTrim, 86
MaxList, 86
MinList, 86
Modulo, 87
Multiply, 87
MultiplyFloat, 87
NextSibling, 88
NodeAccessGroups, 88
NodeExists, 88
NodeInHier, 89
NodeIsLeaf, 89
NodePropValue, 89
Not, 90
Now, 90
NumChildWith, 90
NumDescendantsWith, 91
Or, 91
OrigPropValue, 91
PadChar, 92
PadList, 92
ParentPropValue, 93
Pos, 93
PreviousSibling, 93
PropControllingHier, 94
PropDefaultValue, 94
PropertyCategories, 94
PropMaxValue, 95
PropMinValue, 95
PropValue, 95
RangeListContains, 96
ReplacementAbbrev, 96
ReplacePropValue, 96
ReplaceStr, 97
RTrim, 97
SortList, 97
StripPadChar, 98
StrToBool, 98
StrToFloat, 98
StrToInt, 99
Stuff, 99
SubString, 99
Subtract, 100
SubtractFloat, 100
SumList, 100
Trim, 101
UpperCase, 101
UserName, 101
using properties as variables, 62
using recursion, 62

G
GreaterThanOrEqual function, 72
GreaterThanOrEqual function, 72

H
HasCharacters function, 73
HasChildWith function, 73
HasParentNode function, 73
HasSiblingWith function, 74
HierNodePropValue function, 74

I
ID function, 74
If function, 75
InheritedPropOrigin function, 75
InRange function, 75
InternalPrefix function, 76
Intersection function, 76
IntToStr function, 77
InvertedLevel function, 77
IsAlpha function, 77
IsAlphaNumeric function, 77
IsBlank function, 78
IsBottomNode Function, 78
IsDataType function, 78
IsDefinedPropVal function, 79
IsNodeAbove function, 79
IsNodeBelow function, 80
IsNumeric function, 80
IsRangeListSubset function, 80

L
Length function, 80
LessThan function, 81
LessThanOrEqual function, 81
ListAncestors function, 81
ListChildren function, 82
ListContains function, 82
ListDescendants function, 83
ListDistinct function, 83
ListNodePropValues function, 84
ListNodesWith function, 84
ListRelatedNodesWith function, 84
ListSiblings function, 85
loading metadata, 123
locking out users, 25
LowerCase function, 85
LTrim function, 86
MaxList function, 86
metadata
  migrating, 121
migrating metadata, 121
  comparing, 124
  extracting, 122
  generating reports, 126
  loading, 123
  opening the Migration Utility, 122
MinList function, 86
Modulus function, 87
Multiply function, 87
MultiplyFloat function, 87
M
namespaces
  defined, 41
NextSibling function, 88
node access groups
  creating, 30
  deleting, 31
  editing, 31
  security, 31
node glyphs
  adding and deleting, 108
node types
  defining, 107
  deleting, 108
  editing, 108
NodeAccessGroups function, 88
NodeExists function, 88
NodeInHier function, 89
NodeIsLeaf function, 89
NodePropValue function, 89
Not function, 90
Now function, 90
NumChildWith function, 90
NumDescendantsWith function, 91
O
Or function, 91
OrigPropValue function, 91
P
PadChar function, 92
PadList function, 92
ParentPropValue function, 93
passwords
  changing user, 25
Pos function, 93
PreviousSibling function, 93
PropControllingHier function, 94
PropDefaultValue function, 94
properties
  categories, 37
  creating, 42
  data types, 44
  deleting, 47
property categories
  creating, 38
  defined, 37
  deleting, 39
  editing, 38
property data types
  defined, 44
property definitions
  editing, 46
PropertyCategories function, 94
PropMaxValue function, 95
PropMinValue function, 95
PropValue function, 95
R
RangeListContains function, 96
ReplacementAbbrev function, 96
ReplacePropValue function, 96
ReplaceStr function, 97
RTrim function, 97
S
SortList function, 97
StripPadChar function, 98
StrToBool function, 98
StrToFloat function, 98
StrToInt function, 99
Stuff function, 99
SubString function, 99
Subtract function, 100
SubtractFloat function, 100
SumList function, 100
system preferences
  configuring, 117
  defined, 111

T
Trim function, 101

U
unlocking users, 25
UpperCase function, 101
user authentication, 24
user roles
  defined, 20
UserName function, 101
users
  authenticating, 24
  changing passwords, 25
  changing roles, 26
  creating, 23
  deleting, 26
  locking out, 25
  modifying, 25
  permissions, 15
  roles, 20
  unlocking, 25
  viewing login status, 27

V
validation classes
  defined, 49
validation levels
  defined, 51
validations
  assigning, 53
  creating, 52
  deleting, 54
  editing, 53
viewing metadata, 125