## Contents

**Documentation Accessibility** ..................................................... 7

**Chapter 1. About Data Relationship Management** ........................................... 9
- Features ........................................................................................................ 9
  - Key Features ............................................................................................... 9
  - Hierarchical Data Management Types ....................................................... 11
- System Architecture ....................................................................................... 11
- Data Relationship Management Components ............................................... 12
  - Applications ............................................................................................... 12
  - Versions ..................................................................................................... 12
  - Hierarchies ............................................................................................... 13
  - Nodes ......................................................................................................... 13
  - Referential Integrity ..................................................................................... 14
  - Properties ................................................................................................. 14
  - Inheritance ............................................................................................... 15
  - Validations and Verifications .................................................................... 15
  - Imports ...................................................................................................... 16
  - Automator ................................................................................................. 16
  - Blender ...................................................................................................... 16
  - Exports ...................................................................................................... 17
  - Property Queries ....................................................................................... 17
  - Compare .................................................................................................... 17
- Data Relationship Management Typical Data Life Cycle ............................. 18

**Chapter 2. Getting Started** ............................................................................. 21
- Securing Data Relationship Management .................................................... 21
  - Encryption Used in Data Relationship Management .................................... 22
  - Securing Data Relationship Management Default Accounts ....................... 23
  - Securing the Repository Database ................................................................ 23
  - Securing the Application Server .................................................................. 23
  - Securing the Web Publishing Application ................................................... 24
  - Securing Communications ......................................................................... 24
Chapter 3. Managing Users ................................................................. 41
   About User Management ............................................................... 41
      System Admin .............................................................................. 43
      Functional Admin ......................................................................... 43
      Security Admin .............................................................................. 44
   Managing Users ................................................................................. 45
      Defining Users ................................................................................. 46
      Assigning an Authentication Method ............................................. 46
      Assigning Node Access Groups ................................................... 47
      Assigning Property Categories .................................................... 47
      Creating Users ............................................................................... 47
      Locking Out Users .......................................................................... 49
   Security Configuration Best Practices ........................................... 49
      Assigning Data Relationship Management Users ............................ 49
      Grouping Properties into Categories ............................................. 49
      Assigning User Access to Categories ............................................ 49
      Assigning Access per Node Access Group .................................... 50
   Working with Node Access Groups .................................................. 50
      Using Node Access Groups ............................................................ 51
      Defining Node Access Groups ....................................................... 52
      Assigning Users to Node Access Groups ....................................... 52
      Defining Controlling Hierarchies .................................................. 52
      Using Node Access Group Security ............................................... 52

Chapter 4. Managing Properties .......................................................... 55
   Using the Properties Dialog Box ...................................................... 55
      Property Definition Tab ................................................................... 56
      Creating Properties ......................................................................... 63
      Using the Controlling Hierarchy Tab .............................................. 64
      Using the List Values Tab ............................................................... 65
      Using the Lookup Table Tab .......................................................... 66
      Using the Property Params Tab ....................................................... 67
Chapter 5. Managing Validations and Verifications ........................................... 71
   Validations and Verifications Overview ........................................... 71
   Using the Validations Dialog Box ........................................... 72
      Validation Definition Tab ........................................... 72
      Parameters Tab .................................................... 76
   Creating Validations and Verifications ........................................... 77
   Assigning Validations and Verifications ........................................... 78

Chapter 6. Managing System Categories .................................................. 81
   Versions and Hierarchies Overview ........................................... 81
   Using the System Categories Dialog Box ........................................... 81
      Hierarchies ....................................................... 83

Chapter 7. Managing Node Types ....................................................... 85
   Categorizing Nodes by Node Types ........................................... 85
   Node Types ......................................................... 85
   Using Node Types ........................................... 86
      Using Properties ..................................................... 86
      Using Validations .................................................. 87
   Configuring Node Types ........................................... 87
      Disabling Node Types ..................................................... 88
   Adding, Modifying, and Deleting Node Glyphs ................................ 88

Chapter 8. Working with Formulas ...................................................... 91
   Formula Derivers ..................................................... 91
   Creating and Editing Formulas ........................................... 91
      Formula Syntax Checks ........................................... 92
      Formula Evaluation .................................................. 93
      Property Names in the Syntax Check .................................... 93
   Functions ......................................................... 94
      Function Names ..................................................... 94
      Function Parameters .................................................. 94
      Literals .......................................................... 94
      Formula Evaluation .................................................. 95
      Function Definitions .................................................. 95
Chapter 9. Working with Transaction Requests ............................................. 125
  Viewing Request History ................................................................. 126
  Specifying Request History Range and Filters .................................. 127
  Viewing Transaction Request Details .............................................. 128
  Deleting Transaction Requests ...................................................... 128
  System Preferences for Transaction Requests ................................... 128

Chapter 10. Migrating Data Relationship Management Metadata .................. 129
  Configuration Settings ..................................................................... 130
  Extracting Metadata ....................................................................... 131
  Loading Metadata .......................................................................... 132
  Comparing Metadata ...................................................................... 133
  Viewing Metadata .......................................................................... 134
  Generating Reports ........................................................................ 135

Index .................................................................................................. 137
Documentation Accessibility

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

TTY Access to Oracle Support Services

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

Oracle Hyperion Data Relationship Management, Fusion Edition functions as a hub where reporting structures are maintained, analyzed, and validated before moving throughout the enterprise.

Key Features

Data Relationship Management does the following:

- Manages hierarchies, business rules, mappings, and validations
- Enforces referential integrity across all subscribing systems and hierarchies
- Enables users to create and manage alternate views
- Maintains historical versions for comparative reporting and analysis
- Tracks all hierarchy and attribute changes with a full-featured audit log
- Manages parallel hierarchies
- Can serve as the main point-of-entry to update subscribing systems or be used after-the-fact for reconciliation and analysis

Ease of Use

Although Data Relationship Management is designed to manage and validate the most complex dimensional structures and their related metadata requirements, the interface provides an intuitive and familiar look and feel.

For each system or data set that the tool manages (through import or direct input), there is a graphical representation of the data. To validate this dimensional data, a metadata repository contains user-defined business rules, validation instructions, and relationship information. As
the dimensional data changes in each participating system or within Data Relationship Management, data integrity issues are regulated, validated, and reported.

**Efficiency**

Data Relationship Management is specifically designed to synchronize data across multiple systems. All functionality in the tool is focused on adding efficiency to the hierarchical data management process.

The implementation of Data Relationship Management involves a configuration process, as opposed to a custom development process. Configuration mitigates the inherent risks and testing cycles associated with custom programming.

After implementation, Data Relationship Management provides several data management features that support large-scale analysis and reconciliation of data.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versioning</td>
<td>Multiple instances of data dimensions can be saved and used for historical reporting, comparisons, reconciliations, and what-if analyses</td>
</tr>
<tr>
<td>Business rule enforcement</td>
<td>User-defined validations, ranging from simple data verification to organizational policies, can be applied across dimensions in a real-time or batch mode</td>
</tr>
<tr>
<td>Hierarchy debugging tools</td>
<td>Tools that enable the user to analyze integrity issues across multiple data sets</td>
</tr>
</tbody>
</table>

**Flexibility**

To support the changing needs of a dynamic business environment, Data Relationship Management provides an intuitive, graphical configuration utility. This utility, designed for the non-technical user, enables immediate changes to Data Relationship Management functionality without the need for custom programming.

**Security**

The security model of Data Relationship Management manages user access at the following levels:

- **Versions**—User access is restricted based on the version status
  - A working version is accessible to all users based on their access
  - A submitted version is editable only by system administrators and functional administrators.
  - Finalized and expired versions are read only to all users and administrators.

- **Hierarchy**—User access can be restricted to certain areas within a hierarchy. For example, within the chart of accounts, a group of users may only need to maintain the Asset structure. If a user does not have access to a hierarchy, the hierarchy is not displayed. If a user does not have access to any hierarchies in a version, the version is not displayed.
● Property categories—User access can be restricted to certain property categories for a specified node. In most cases, property categories are based on a system basis. For example, a user may be allowed to access the Essbase property category with read/write permission and the Financial Management property category group with read-only permission. If a user does not have access to a property category, the tab does not display.

● Administrative and user types—Users can be grouped into four types:
  ○ System Administrator — has full access including to metadata
  ○ Functional Administrator — has full access to the hierarchy data but does not have ability to change metadata
  ○ Security Administrator — can only create and assign logins
  ○ Normal user — has access to only the information based on the node access groups to which they are assigned and the property categories to which they are granted access

The configuration utility for Data Relationship Management supports user-defined validations that enforce integrity and policy issues. These business rules can be applied during the data entry process or routinely in a batch mode.

Hierarchical Data Management Types

Types of hierarchical data managed by Data Relationship Management are:

● Organizational structures
● Charts of accounts
● Products
● Customer relationships

System Architecture

The Data Relationship Management N-tier application is based on an application-server architecture. Use of an application-server architecture enables the bulk of the system processing to be performed on a centralized server and simplifies the client requirements for each end user.

The following diagram gives a top-level overview of the Data Relationship Management N-tier architecture. The Data Relationship Management client is a Windows application that runs on the user’s local computer. The client connects to the Data Relationship Management application server which can support multiple, simultaneous users. The Data Relationship Management database can be hosted on the application server machine or elsewhere.
Data Relationship Management Components

The following topics describe the main components and features of Data Relationship Management.

Applications

A Data Relationship Management installation can use multiple back-end databases to support the many business needs for hierarchical data within an organization. Each database is referred to as an application, and each application is independent from any other application. If you are in doubt about which application to use, see your system administrator.

Versions

Data Relationship Management groups sets of hierarchies into versions. A version represents a single, independent set of data arranged into related hierarchies. Versions are usually related to time periods or functions. Examples of versions are March 2001, 3rd Quarter 2000, and Planning. All Data Relationship Management maintenance is performed in versions; users cannot copy or move nodes across versions. The only features that work across versions are compares and some exports.

Nodes and properties within a version are shared among the hierarchies within the version. Versions are typically used for the following purposes:

- To represent a set of hierarchies used during a particular month (or other business cycle period). Each month a version is created, maintaining an audit trail of hierarchies.
- To differentiate between real and test data during system testing.
- To compare different versions to identify changes made to the hierarchies within a time dimension.
You can create new versions by copying existing versions, but, after a version is created, it is independent of other versions. Versions can be copied, created, and deleted by a system administrator.

Each version has an associated status.

### Table 2  Status Descriptions

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working</td>
<td>Users can edit the hierarchies within the version</td>
</tr>
<tr>
<td>Submitted</td>
<td>System administrators are performing final validations on the hierarchies. No further changes by other users are permitted</td>
</tr>
<tr>
<td>Finalized</td>
<td>No changes are permitted. From the current version, all exports to other systems are performed.</td>
</tr>
<tr>
<td>Expired</td>
<td>The hierarchies are now out-of-date. Data is maintained for historical purposes and as an audit trail</td>
</tr>
</tbody>
</table>

### Hierarchies

A hierarchy is a set of nodes that are all descendants of the same node. Thus, a hierarchy is defined by its top node and represents all nodes in the hierarchical relationships below that top node.

Hierarchies are contained within a version; and a version can contain multiple hierarchies. Hierarchies provide the main interface for a user when working with Data Relationship Management.

Each hierarchy is usually associated with a certain view, external system, or management report. Examples of hierarchies are Line of Business, Geographic, SAP - Legal, and Management Summary.

### Nodes

A node is a point within a hierarchy. Every point in a hierarchy is a node. For example, within a hierarchy that represents an organizational structure, a node might represent a department or a cost center.

Within a version, a node may be a part of more than one hierarchy. A node can have many user-defined properties that store information about the node and control the use of the node within the information systems of an organization.

The following terms are used to define the position of a node and behavior of the node within a hierarchy:

### Table 3  Node Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaf</td>
<td>A node that cannot have children</td>
</tr>
<tr>
<td>Limb</td>
<td>A node that can have children</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Child</td>
<td>A node directly below another node (if B is directly below A, B is a child of A)</td>
</tr>
<tr>
<td>Parent</td>
<td>A node directly above another node (in the previous example, A is the parent of B)</td>
</tr>
<tr>
<td>Descendants</td>
<td>All nodes below a specified node (including children and all children of children)</td>
</tr>
<tr>
<td>Ancestors</td>
<td>All nodes between a node and the top of the hierarchy (including the parent, the parent of the parent, and so on)</td>
</tr>
<tr>
<td>Children</td>
<td>All nodes directly below a node</td>
</tr>
<tr>
<td>Siblings</td>
<td>All nodes that share a parent node in a particular hierarchy</td>
</tr>
<tr>
<td>Orphan</td>
<td>A node not assigned to any hierarchy</td>
</tr>
</tbody>
</table>

**Referential Integrity**

Referential integrity (RI) is a concept normally associated with relational databases. In Data Relationship Management, RI means that no relational data violates established keys or domain ranges. In this database context, RI refers to two core rules that the Data Relationship Management engine enforces while users are editing hierarchies:

- A node must have the same children in all hierarchies. Thus, a node always represents the same rollup structure regardless of context.
- A node cannot exist more than once in a hierarchy. Thus avoiding any “circular reference” problems.

By enforcing RI, Data Relationship Management inhibits errors related to structure and redundancy that might occur during the maintenance process.

**Properties**

Properties are data elements that are similar to fields in a database. Properties can be defined and stored at four levels:

- Version
- Hierarchy
- Global node - the value for the node is the same no matter what hierarchy it is in or what parent it has
- Local node - the value for the node can be different for the node in different hierarchies

Most properties in Data Relationship Management are defined at the node level and contain node descriptions. Examples include the name of a node (called Abbrev in Data Relationship Management), the description for a node, and the number of children of a node. Properties at the version and hierarchy level are less common. Examples include properties used for header-level information in Data Relationship Management exports.
The system administrator can define as many properties as needed. Data Relationship Management is delivered with two categories of standard properties:

- System properties that define a node
- Statistical properties that provide information about a node

**Inheritance**

Inheritance is a feature that enables high-level nodes to share their property values with lower points in the hierarchy, eliminating the need to store and maintain redundant information. It enables newly entered nodes to automatically obtain their property values from the appropriate ancestors. Proper use of inheritance can greatly reduce data-entry requirements.

When defining a property, the system administrator can define the property as inheriting. This definition enables the values for the property to cascade down to its descendants.

Inheritance moves through a specific chain of events to determine the value for a property:

1. Data Relationship Management looks for a value entered at the current node. If a user has directly entered a value at the node, the entered value is used.
2. If a value does not exist, Data Relationship Management searches the ancestors of the node for a value. The first entered value that Data Relationship Management finds, moving up the hierarchy is used. Thus, a change to the properties of a node can affect any descendents.
3. If no ancestor has an entered value, the default value is used. A default value is assigned by the system administrator.

Global properties that inherit follow a slightly different path. In step 2, as Data Relationship Management moves up the hierarchy in search of an entered value, it encounters the ancestors in the controlling hierarchy. When the system administrator creates a global property, a controlling hierarchy must be designated for the property. A controlling hierarchy tells the system which hierarchy to use to determine the inheriting value for a global property.

Many tools are available in Data Relationship Management for maintaining inherited values:

- Locking a value so that it cannot be overridden at a descendant
- Clearing all descendant values for a particular property
- Removing an overridden value so that the property inherits from a node above

**Validations and Verifications**

Validations and verifications are tests to ensure that hierarchy rules are observed. They help enforce business rules.

- Validations—run automatically (in real-time) as users edit the structures and properties of hierarchies. Validations are automatically performed for the node being edited and also for its parent.
- Verifications—run on-demand (as a batch) after users make a set of changes. Users can choose specific verifications or run a set of verifications defined by the system administrator.
Validation and Verification Example
An organization has a business rule that requires that a sales representative assigned more than 20 customers obtain special approval from the marketing department.

- A validation using this business rule within Data Relationship Management permits no more than 20 customers to be added to a sales representative node.
- A verification using this business rule reports only upon sales representatives with more than 20 customers.

Comparing Validations and Verifications

- Some business rules call for real-time enforcement and, therefore, should be defined by the system administrator as validations.
- Other business rules only need to be monitored and should be established as verifications. In the preceding example, a verification is probably more appropriate because, in certain circumstances, it is appropriate for a sales representative to have more than 20 customers. Validations are more appropriate for rules that can never be broken and require immediate enforcement.

Imports
Data Relationship Management can import data from external systems of various formats, including text files, data extracts, and other sources. An import is always performed on a new, empty version that is created as part of the import process. Thus, the user can verify that the import has run correctly before moving the imported data into a production environment. After the data is verified, Blender can be used to incorporate the new data. Imports can be customized and saved by any Data Relationship Management user who is authorized to run imports.

Automator
Automator is an alternative mechanism that enables Data Relationship Management users to make bulk hierarchy changes. The changes are defined in a text file. The functions supported are typical node and property manipulations. When Automator is used, the changes defined in a text file can be performed automatically and easily. Rather than using Automator, you can make corrections to the data as it is processed.

Blender
Blender enables you to combine elements of two versions into a new version or to combine elements into an existing version. The elements to be blended can include various combinations of structural elements such as hierarchies, nodes, and properties.
Exports

Exports are tools that are used to transfer hierarchy data to external systems or to a database table. Exports are sometimes referred to as reports or extracts. Exports also serve as the main reporting mechanism for Data Relationship Management. Data Relationship Management provides users a wizard-like export builder that enables them to create and save exports.

Exports can create paper reports for distribution or text files to be imported by external systems. If an external system requires a complex export that cannot be generated by the report writer, a new class of export can be created by the programming team and added into the export builder. Each export class has a set of parameters that enable it to be customized for a user’s specific requirements.

Property Queries

Using Structured Query Language (SQL) to query hierarchically structured data has always been difficult (if not impossible) due to the recursive nature of the required query. Property queries enable you to investigate the hierarchical structure and the property values of sets of nodes without the need for complex recursive SQL programming.

Property queries can be used for several purposes:

● To find nodes that meet certain criteria
● In exports and comparisons as a filtering mechanism
● As a parameter in a generic validation routine (providing queries additional functionality as validations)

Users create a list of criteria, similar to the WHERE clause in a conventional SQL statement, and run it against a set of nodes. The property query returns a list of nodes that meet the defined criteria.

Note that:

● Property queries can be saved, so Data Relationship Management users can create and share their own toolkit of queries
● The system administrator can create standard queries that are available to all users

Global property queries are similar to property queries but are run against the list of all nodes in a version instead of against a particular set of nodes within a hierarchy. Because a global property query runs against a set of nodes without reference to a hierarchy, only global properties can be included in the criteria.

Property queries can be used as the basis for filters for exports and compares as well as the basis for validations and verifications.

Compare

The Compare feature enables you to compare the structure and node properties of hierarchies and versions.
Three Compare options are available:

- The standard Compare option requires that two points, one in each of two hierarchies, be selected. Their structures are then analyzed for differences. The points may be in different versions.
- The Property Compare option enables two points, one in each of two hierarchies. The representatives of a particular property within the hierarchies are then analyzed for differences.
- The Renamed Node Compare option duplicates the standard Compare option, except it treats renamed nodes as being the same for comparison purposes. For example, assume that Hierarchy 1 in a compare includes nodes A, B, and C, and Hierarchy 2 includes nodes A, B, and C, but node C has been renamed to CC. The Renamed Node Compare option does not identify differences between the two hierarchies. But the standard Compare option regards nodes C and CC as different.

**Note:**

The Renamed Node Compare option should only be used between versions that are related (that is, one copied from the other).

A Compare process returns two lists:

- A list of all nodes that exist in the first hierarchy but not in the second
- A list of all nodes that exist in the second hierarchy but not in the first

Queries can be used to filter the lists of nodes to be compared.

---

**Data Relationship Management Typical Data Life Cycle**

Most organizations use Data Relationship Management on a cyclical basis that matches their management reporting periods (typically every four weeks or every month). Within each reporting period, the use of Data Relationship Management follows a predictable pattern:

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Data Life Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td><strong>Principal Players</strong></td>
</tr>
<tr>
<td>A Data Relationship Management working version is created as a copy of the finalized version for the previous period. The new version may contain multiple hierarchies (for example, for the chart of accounts, the organization structure, and the product structure).</td>
<td>System administrator</td>
</tr>
<tr>
<td>Changes are made to the working version by business users who have access to Data Relationship Management. Validations are automatically performed as users enter or amend hierarchy data.</td>
<td>Data Relationship Management users</td>
</tr>
<tr>
<td>If necessary, bulk changes to hierarchy data are performed using Data Relationship Management Automator.</td>
<td>Data Relationship Management users and/or the system administrator</td>
</tr>
</tbody>
</table>
As the reporting period deadline looms, the version status is changed to *submitted* (changes by users are no longer permitted). Validations and verifications are performed to ensure the integrity of the data. Compare options can be used to identify changes.

When data integrity is assured, the version status is changed to *finalized* (no further changes are permitted). Simultaneously, the version status for the previous reporting period is changed from *finalized* to *expired*, and the version is stored for possible future use in historical analyses or as an audit record.

Exports are performed from the new *finalized* version to feed hierarchy data to participating systems. After all exports are complete (and have been loaded by the destination systems), all participating systems have consistent hierarchical data as a basis for the period end reporting process.

And the cycle repeats for the next reporting period.

Existing organizational workflow constraints can be enforced by Data Relationship Management:

- Business rules might require that all new cost centers be approved by Corporate Treasury. In this case, a property can be added to indicate approval, and no nodes are exported to other systems until the property is changed to approved. Corporate Treasury can be granted access to Data Relationship Management to update only the indicator property. A property query can also be defined to identify indicator nodes.

- Business processes might require that all hierarchy updates be redirected to a dedicated group responsible for implementing such updates. Following review and approval, using Automator, changes can be entered into a flat file for bulk loading into Data Relationship Management. This automated approach can significantly reduce potential typing errors.

Other tasks that are performed on an irregular basis:

- New hierarchies can be established to support an expansion in scope of the participating systems. Such an action is performed using a process similar to the initial load of Data Relationship Management with hierarchical data.

- Hierarchies can be restructured to match a change in business needs. Data Relationship Management includes tools, such as Blender, to facilitate this process.
Securing Data Relationship Management

The following features in Data Relationship Management are used to secure the system:

- Encryption and hashing is used to secure passed and stored password information
- HTTPS can be used to further secure the communications between the client and the server
- Database access does not need to be given to the end users
- Users are given access via roles and access groups to limit them to only what they need to have access to within the system
- Data Relationship Management can be configured to use an external authentication system such as LDAP, NTLM, or MSAD through Oracle's Hyperion® Shared Services
- When using internal authentication, Data Relationship Management system preferences can be configured to require complex passwords and require password changes and lockout conditions

In addition to the features in Data Relationship Management, the system can be further secured in the following ways:

- Use specific, limited accounts for the Data Relationship Management services, COM+ and DCOM objects rather than the local system account
- Secure access to the Web Publishing page through IIS or other technologies
- Change default passwords for MDM_DB, MDM_ADMIN, MDM_SYSTEM, and Data Relationship Management Schema Owner
- Create new Data Relationship Management system and Admin accounts and disable the default accounts (MDM_ADMIN, MDM_SYSTEM and MDM_DB)
- If the Data Relationship Management Oracle Schema account is not being used to access the system, secure it by removing the CreateSession privilege. See above item about creating different accounts and disabling the default accounts.
● Use the ability to store the user and password for the Batch Client with the Windows Data Protection System instead of putting the username and password on the command line or in the configuration file in clear text
● Use SSL to secure communications to the client for Data Relationship Management and Web Publishing
● Do not give out database access to the back-end repository
● Change the authentication level for calls for the COM+ objects to packet privacy
● Ensure that the Launch and Activation, Access Permissions, and Configuration Permissions settings for the DCOM objects are appropriate

**Encryption Used in Data Relationship Management**

Data Relationship Management uses encryption in the following ways:

● Encryption keys for storing server configuration passwords are generated upon installation. Encryption keys for providing the public key for encryption of passwords being passed to the server are generated during server startup.
● Encryption of the password in the Create Session API call (using the public key generated at server startup)
● Storage of the passwords in the internal username table using a salted hash (internal authentication)
● Storage of the encrypted passwords for the application server layer configuration (using the public key generated during installation)

**Create Session Encryption**

Encryption is used in passing the password from the client to the server and this uses RSA at 256 bits. The public/private key pair is randomly generated during server startup and the client requests the public key from the server.

The public key is generated randomly at the server startup. This means that the encrypted password cannot be saved and used again after the server is restarted. The client should ensure that it has the latest public key from the server.

External programs communicating through the API can choose to use this encryption or to pass the password in clear text. This is controlled by setting the ClearTextPassword parameter to True or False in the CreateSession call. For more information on the Data Relationship Management API see the *Oracle Hyperion Data Relationship Management N-tier Application Programming Interface Guide*.

**Password Storage for Internal Authentication**

If internal authentication is being used, Data Relationship Management stores the password for the user in its database repository using Secure Hash Algorithm 1 (SHA1). The hash is salted to
prevent comparisons of different hashes within the database to detect password usage. If external authentication is used, then no passwords are stored in Data Relationship Management.

**Password Storage for Application Server Configuration**

When Data Relationship Management stores password information in the `config.xml` file for the application server layer configuration, the passwords are encrypted using Advanced Encryption Standard (AES), also known as Rijndael. The key for this encryption is generated during installation so that it is not the same between two clients.

**Securing Data Relationship Management Default Accounts**

After installation, the following steps should be taken:

- The default passwords for the Data Relationship Management accounts should be changed. This includes the passwords for MDM_DB, MDM_ADMIN, and MDM_SYSTEM as well as the MDM_DB accounts on the database server.
- To further secure the system, new accounts can be created to take the place of the default system accounts in Data Relationship Management and on the database server. After these accounts are created and the system is switched to use them, then the default accounts can be disabled.

**Securing the Repository Database**

Instead of using the database schema or owner account (MDM_DB), a more limited account can be created and assigned the MDM_ADMIN_ROLE role to grant it access to the repository. The schema or owner account can also be secured. In Oracle, you can remove CREATESESSION after an alternate account is being used. Note that this account would need to be re-enabled to perform an upgrade.

Do not give out database access to end users. The database access should be tightly controlled. Follow Oracle best practices in securing the database environment.

**Securing the Application Server**

Securing the Data Relationship Management application server can be done in the following ways:

- Change the account used for the Data Relationship Management service. The account used should have at least the following privileges:
  - Edit registry settings
  - Read and write to the local file system
  - Launch processes
  - Run as a service
Use specific, limited accounts for the DCOM and COM+ objects. The account used should have at least the following privileges:
  - Edit registry settings
  - Read and write to the local file system
  - Launch processes

Change the authentication level for calls for the COM+ objects to packet privacy

Ensure the Launch and Activation, Access Permissions, and Configuration Permissions settings for the DCOM objects are configured correctly

**Securing the Web Publishing Application**

Securing the Data Relationship Management Web Publishing application can be done in the following ways:

- Change the account used for the Data Relationship Management Web Publishing service. The account used should have at least the following privileges:
  - Edit registry settings
  - Read and write to the local file system
  - Launch processes
  - Run as a service

- Use specific, limited accounts for the DCOM objects. The account used should have at least the following privileges:
  - Edit registry settings
  - Read and write to the local file system
  - Launch processes

- Secure access to the Web Publishing page through IIS or other technologies such as SiteMinder.

- Ensure the Launch and Activation, Access Permissions, and Configuration Permissions settings for the DCOM objects are configured correctly

**Securing Communications**

Data Relationship Management also supports the use of HTTPS to secure the packet information transmitted between the client and the Web layer. HTTPS is recommended to secure information passed between the client and the Data Relationship Management Web layer.

HTTPS can be implemented for the Data Relationship Management client to server applications as well as for the Web Publishing Web page.

HTTPS can also be used to secure the communications to the Oracle's Hyperion® Shared Services system when external authentication is used.
Securing the Batch Client Utility

The batch client utility (mdm_connect_ntier.exe) is used to perform batch processes in Data Relationship Management. Many times it is used in a lights out fashion. This means that it needs to have a username and password provided to it to be able to connect to the Data Relationship Management server.

Prior to the 9.3.1 release, the username and password were provided via the command line or in the configuration file. In both cases, the information was in clear text. These files could be secured on the server to prevent releasing the username and password.

Now, the batch client utility can use the Windows Data Protection Application Interface (DPAPI) to allow a more secure method to store credentials.

DPAPI protects the information and places it in a location that is accessible only to the user who creates it.

Data Relationship Management provides a command line utility (mdm_connect_credentials.exe) that can set the credentials for use by the batch client utility.

When the batch client utility runs and no credentials are passed on the command line or in the configuration file, the utility uses the credentials stored using the Windows DPAPI.

The ability to pass the username and password in the command line and in the configuration tool has been left in for backward compatibility and to ease upgrades. However, it is recommended that the secure DPAPI methodology be used going forward.

About System Administration

The Data Relationship Management system administrator performs administrative functions, which include managing the following areas:

- Users and security
- Versions
- Properties and property categories
- Validations
- Node types

Caution!
Administrator functions are designed for use by experienced system administrators. Hence, some potentially damaging options are available and cannot be undone.

Naming Metadata Objects

When you create metadata objects such as properties, system categories, node access groups, validations, and so on, do not use special characters other than the following:
Admin Menu

When an administrative user is logged on to Data Relationship Management, the Admin menu is available, which contains the following commands:

- Node Access Groups
- Node Types
- Node Type Glyphs
- Properties
- Property Categories
- System Categories
- System Preferences
- User Management
- Validations

Each option opens a dialog box.

Using System Preferences

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

Caution!

Using commas in node names can cause difficulties when working with certain properties that are comma-delimited, such as Ancestor List, Child List and client-created node lists. In addition,
using the characters defined in the system preferences for the shared node separator and the hierarchy separator can cause unpredictable behavior and should be avoided. If the query wildcard characters asterisk ( * ) and underscore ( _ ) are used in a node name they cannot be easily searched for unless a derived formula is used to search for them using the Pos function.

## Configuring System Preferences

To display the System Preferences dialog box:

1. From the Data Relationship Management main menu, select Admin, then System Preferences.

![System Preferences](image)

When you select a system preference from the Current list, its parameters are displayed and available for editing.

2. From the System Preferences dialog box, click Search.
3 Under Search For, enter the string for which to search.

4 From Search In, select a system preference item:
   - Name
   - Label
   - Value
   - Description

5 Click Find.

The results of the search are displayed in the Results area, from which you can then select a system preference to edit. See “System Preferences” on page 28.

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>AddedByProperty</td>
<td>String</td>
<td>When Track Changes is enabled, points to the global string property that indicates the user that added the node</td>
</tr>
<tr>
<td>AddedOnProperty</td>
<td>String</td>
<td>When Track Changes is enabled, points to the global string property that indicates the date/time that node was added</td>
</tr>
<tr>
<td>AllExcl</td>
<td>String</td>
<td>Excludes property categories from the All tab on the property editor</td>
</tr>
<tr>
<td>AllowAsOf</td>
<td>Boolean</td>
<td>Forces capture of core actions and creation of baseline versions</td>
</tr>
<tr>
<td>System Preference</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AllowNextIDGeneration</td>
<td>Boolean</td>
<td>Enables automatic Next ID generations True to enable, otherwise False</td>
</tr>
<tr>
<td>AllowNextIDKeyCreation</td>
<td>String</td>
<td>Enables Next ID key creation level: Admin, Funct, Any, None</td>
</tr>
<tr>
<td>AllowPropCopyOnAdd</td>
<td>String</td>
<td>Enables properties to be copied when a new node is added: Admin, Funct, Any, None</td>
</tr>
<tr>
<td>AllowPru</td>
<td>Boolean</td>
<td>Enables the pruning option which allows a non-admin user to remove a node that has children. Default is True. If False, a non-admin user cannot remove a node that has children. True to enable, otherwise False</td>
</tr>
<tr>
<td>AllowRelaxedMove</td>
<td>Boolean</td>
<td>When a node is moved, enables the new parent to take precedence over any conflicting parental relationships True to enable, otherwise False</td>
</tr>
<tr>
<td>AllowRequestHistoryDelete</td>
<td>String</td>
<td>Security level required for deleting a request via the Request History screen (Admin, Funct, Any, None)</td>
</tr>
<tr>
<td>AllowRequestHistoryQuery</td>
<td>String</td>
<td>Security level required for querying or viewing a request via the Request History screen (Admin, Funct, Any, None)</td>
</tr>
<tr>
<td>AllowSpac</td>
<td>Boolean</td>
<td>Enables spaces in node names True to enable, otherwise False</td>
</tr>
<tr>
<td>ApprovalGroups</td>
<td>String</td>
<td>Comma-delimited string list of approval groups</td>
</tr>
<tr>
<td>ApprovalGroupTrack Properties</td>
<td>String</td>
<td>Delimited string 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>
</tbody>
</table>
| AuthMethod                     | String | User authentication method:  
  - Internal - Users are only authenticated within Data Relationship Management as in previous releases. This is the default setting.  
  - CSS (External) - Users are only authenticated externally. Requires access to Shared Services.  
  - Mixed - Users are authenticated internally or externally based on a setting for each individual user |
<p>| ChangedByProperty              | String | When Track Changes is enabled, points to the global string property that indicates the user that last changed the node                          |
| ChangedOnProperty              | String | When Track Changes is enabled, points to the global string property that indicates the last date/time that the node was changed                |
| CopyLcl                        | Boolean| Copies local values when a node is copied                                                                                                |</p>
<table>
<thead>
<tr>
<th><strong>System Preference</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyPropertiesList</td>
<td>String</td>
<td>The property list for the default selected list of properties to copy when copying properties from an Add Node, Model After or Paste Property operation.</td>
</tr>
<tr>
<td>CurVer</td>
<td>String</td>
<td>Current version of Data Relationship Management for this installation</td>
</tr>
<tr>
<td>Database Name</td>
<td>String</td>
<td>Database name for OLE connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> No longer applicable to current product, retained only for backward compatibility.</td>
</tr>
<tr>
<td>DbSeqSep</td>
<td>String</td>
<td>Sequence separator character</td>
</tr>
<tr>
<td>DefaultCurrentVersion</td>
<td>String</td>
<td>Default current version</td>
</tr>
<tr>
<td>DefaultPreviousVersion</td>
<td>String</td>
<td>Default previous version</td>
</tr>
<tr>
<td>DefaultPropCopyMode</td>
<td>String</td>
<td>Default property copy mode: Overridden, Selected, ForceAll</td>
</tr>
</tbody>
</table>
| DeleteNodeWarning Message                 | String   | Enables confirmation warning message before deleting a node. Enter text for the warning message or leave blank for no warning message.  
Blank = no confirmation                    |
| DetProps                                  | String   | Property types saved when deleting                                                                                                         |
| DfltDir                                   | String   | Default output directory                                                                                                                   |
| DynamicDisplay                            | Boolean  | Enables dynamic display update                                                                                                              |
|                                           |          | True to enable, otherwise False                                                                                                             |
| EnableDeleteDetachedVersionsPrompt        | Boolean  | Enables the Delete Detached Versions prompt that an Admin user gets when logging out  
True to enable prompt, otherwise False  
Default is True                            |
| EnablePropCopyOptions                     | String   | Enables property copy options: Admin, Funct, Any, None                                                                                     |
| EnforceListProps                          | Boolean  | Allow updates to a List Property with values from the pre-defined list only  
Applies only to updates using Automator and Import since the property editor user interface only allows input from the pre-defined list |
| ExcludeFromWebProp                        | String   | **Note:** This system preference applies to Data Relationship Management Web Publishing only.  
Points to a boolean property [Property Abbrev] used to exclude specific nodes from the Web client. If the boolean property is |
<table>
<thead>
<tr>
<th><strong>System Preference</strong></th>
<th><strong>Type</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ExpandTh</td>
<td>Integer</td>
<td>Node count threshold that triggers a warning for the expand tree function</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>FindWildCardAppend</td>
<td>Boolean</td>
<td>Appends an asterisk (*) to the Find criteria when Exact Match is not selected. True to enable, otherwise False</td>
</tr>
<tr>
<td>FindWildCardPrepend</td>
<td>Boolean</td>
<td>Prepends an asterisk (*) to the Find criteria when Exact Match is not selected. True to enable, otherwise False</td>
</tr>
<tr>
<td>GlobalPropLocalOverride</td>
<td>String</td>
<td>List of properties to exclude from local checks on global properties. These are considered when GlobalPropLocalSecurity is enabled.</td>
</tr>
<tr>
<td>GlobalPropLocalSecurity</td>
<td>Boolean</td>
<td>Enforces local security on global properties. If True, changes to Global properties are checked against local security (Node access levels) for the user for all hierarchies that the node is in. Default is False</td>
</tr>
<tr>
<td>HierNodeOptCount</td>
<td>Integer</td>
<td>Number of nodes in a transaction triggering optimization of the HierNode table</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> No longer applicable to current product, retained only for backward compatibility.</td>
</tr>
<tr>
<td>HierSep</td>
<td>String</td>
<td>Hierarchy and Node separator character</td>
</tr>
<tr>
<td>IdleTime</td>
<td>Integer</td>
<td>Number of minutes to time out. Default is 60</td>
</tr>
<tr>
<td>IdleTimeExcludeUsers</td>
<td>String</td>
<td>Comma-separated list of users to exclude from Idle Time session expiration</td>
</tr>
<tr>
<td>Inactivate</td>
<td>String</td>
<td>Level at which to enable node inactivation: Admin, Funct, Any, None</td>
</tr>
<tr>
<td>InactiveChanges</td>
<td>String</td>
<td>Enables changes to inactive node: Admin, Funct, Any, None</td>
</tr>
<tr>
<td>InvDescr</td>
<td>String</td>
<td>Invalid description characters</td>
</tr>
<tr>
<td>InvName</td>
<td>String</td>
<td>Invalid name characters</td>
</tr>
<tr>
<td>LeafEdit</td>
<td>String</td>
<td>Enables editing of the Leaf property: Admin, Funct, Any, None</td>
</tr>
<tr>
<td>LockDesc</td>
<td>String</td>
<td>Message displayed to locked-out users when LockOut Flag is set</td>
</tr>
<tr>
<td>System Preference</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LockOut</strong></td>
<td>Integer</td>
<td>Lockout flag to disable logins for Client Server client tool. [0=no lockout, 1=lockout all, 2=lockout non-Admin only]</td>
</tr>
<tr>
<td><strong>LockoutExcludeFuncAdmin</strong></td>
<td>Boolean</td>
<td>Excludes functional administrators from automatic login lockouts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td><strong>LockoutExcludeSecAdmin</strong></td>
<td>Boolean</td>
<td>Excludes security administrators from automatic login lockouts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td><strong>LockoutExcludeSysAdmin</strong></td>
<td>Boolean</td>
<td>Excludes system administrators from automatic login lockouts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td><strong>LockoutExcludeUsers</strong></td>
<td>String</td>
<td>Comma-separated list of users to exclude from automatic login lockouts</td>
</tr>
<tr>
<td><strong>LockoutInactivity</strong></td>
<td>Integer</td>
<td>Maximum number of inactive days before a user is locked out</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default is 30 (Zero indicates no maximum)</td>
</tr>
<tr>
<td><strong>LockoutInvalidLogins</strong></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 is 6. (Zero indicates no maximum)</td>
</tr>
<tr>
<td><strong>LossLevel</strong></td>
<td>String</td>
<td>Loss level to capture:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defined, All</td>
</tr>
<tr>
<td><strong>LVProps</strong></td>
<td>String</td>
<td>Default properties for list view</td>
</tr>
<tr>
<td><strong>MaxDescr</strong></td>
<td>Integer</td>
<td>Maximum length for the node description</td>
</tr>
<tr>
<td><strong>MaxLeaf</strong></td>
<td>Integer</td>
<td>Maximum length for the leaf name</td>
</tr>
<tr>
<td><strong>MaxLimb</strong></td>
<td>Integer</td>
<td>Maximum length for the limb name</td>
</tr>
<tr>
<td><strong>MaxLstPr</strong></td>
<td>Integer</td>
<td>Maximum number of properties allowed in the list view on hierarchy viewer</td>
</tr>
<tr>
<td><strong>MaxPrpEd</strong></td>
<td>Integer</td>
<td>Maximum number of property editor windows</td>
</tr>
<tr>
<td><strong>MaxSync#</strong></td>
<td>Integer</td>
<td>Maximum difference for node count</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No longer applicable to current product, retained only for backward compatibility.</td>
</tr>
<tr>
<td><strong>MemoWrap</strong></td>
<td>Boolean</td>
<td>Memo editor word wrap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise 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>MustVer</td>
<td>String</td>
<td>Must have at least one of these Data Relationship Management product versions</td>
</tr>
<tr>
<td>NdTpDAll</td>
<td>String</td>
<td>Enables Show All Properties to override node type: Admin, Funct, Any, None</td>
</tr>
<tr>
<td>NodeChangedProperty</td>
<td>String</td>
<td>When Track Changes is enabled, points to the boolean property that indicates whether node is changed</td>
</tr>
<tr>
<td>PasswordDuration</td>
<td>Integer</td>
<td>Number of days that a user's password is valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default is 30</td>
</tr>
<tr>
<td>PasswordMaxLength</td>
<td>Integer</td>
<td>Maximum length for user passwords (Zero indicates no maximum)</td>
</tr>
<tr>
<td>PasswordMinLength</td>
<td>Integer</td>
<td>Minimum length for user passwords (Zero indicates no minimum)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default is 6</td>
</tr>
<tr>
<td>PasswordPolicyEnabled</td>
<td>Boolean</td>
<td>Set to TRUE to enforce that the password 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><strong>Note:</strong> Default 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</td>
</tr>
<tr>
<td>RemoveNodeWarning Message</td>
<td>String</td>
<td>Enables confirmation warning message before removing a node</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enter text for the warning message or leave blank for no warning message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Blank = no confirmation</td>
</tr>
<tr>
<td>RenameLeaf</td>
<td>String</td>
<td>Level required for renaming the Leaf:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Admin, Funct, Any, None</td>
</tr>
<tr>
<td>RenameLimb</td>
<td>String</td>
<td>Level required for renaming the Limb:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Admin, Funct, Any, None</td>
</tr>
<tr>
<td>ReqMerge</td>
<td>Boolean</td>
<td>Requires merge for inactivates or deletes when UseMerge is enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td>Server Name</td>
<td>String</td>
<td>Server name for OLE connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> No longer applicable to current product, retained only for backward compatibility.</td>
</tr>
<tr>
<td><strong>System Preference</strong></td>
<td><strong>Type</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>SharedNodeDelimiter</td>
<td>String</td>
<td>Specifies the delimiter between the node name and the shared node suffix. Default is colon (:)</td>
</tr>
<tr>
<td>SharedNodeIdentifier</td>
<td>String</td>
<td>Specifies the identifier to be used after the shared node delimiter. Default is “Shared”</td>
</tr>
<tr>
<td>SharedNodeMaintenanceEnabled</td>
<td>Boolean</td>
<td>Enables shared nodes. True to enable; default is False.</td>
</tr>
<tr>
<td>SharedNodeNamingType</td>
<td>String</td>
<td>Specifies whether the alternate name is a suffix or prefix for shared nodes. Default is suffix.</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.</td>
</tr>
<tr>
<td>SharedNodeSequenceSeparator</td>
<td>String</td>
<td>Specifies the separator character to be placed after the shared node identifier. Default is dash (-).</td>
</tr>
<tr>
<td>SharedNodeSequenceType</td>
<td>String</td>
<td>Specifies whether to use a numeric uniqueness key or to use the ancestor’s node name as the uniqueness key. Default is numeric.</td>
</tr>
<tr>
<td>SortLimbsFirst</td>
<td>Boolean</td>
<td>Controls whether leaf nodes are sorted last. The Default is True. If this preference is set to True, limbs and leaves are sorted separately with the leaves being last. If this preference is set to False, limbs and leaves can be sorted together. This preference affects hierarchy exports, display, and hot node lists.</td>
</tr>
<tr>
<td>TopNodeParentString</td>
<td>String</td>
<td>Used in Import and Export to denote parent value for a top node.</td>
</tr>
</tbody>
</table>
| TrackChanges          | String   | Enables the change tracking feature: True, False. When using Track Changes, the following system preferences are required:  
  - AddedByProperty  
  - AddedOnProperty  
  - ChangedByProperty  
  - ChangedOnProperty  
  - NodeChangedProperty  
  **Caution!** Set this preference to True only after all change tracking configuration steps have been completed, otherwise the Data Relationship Management service will generate an error. For more information, see “Setting Up Change Tracking” on page 37. |
<p>| TransactionLevels     | String   | Transaction levels to capture. Turning on As Of or specifying result or loss actions forces core actions to be captured. |</p>
<table>
<thead>
<tr>
<th>System Preference</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td></td>
<td>Action types: Logged Action, Core Action, Result Action, Loss Action</td>
</tr>
<tr>
<td>UpName</td>
<td>Boolean</td>
<td>Uses uppercase for the node name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td>UseChangeApproval</td>
<td>Boolean</td>
<td>Use change approval</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td>UseCopyPropertiesList</td>
<td>Boolean</td>
<td>Boolean property to track copy properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If True the system uses the list of properties from CopyPropertiesList as the default selected list on dialogs for copying properties from another node (for example, applicable in Add Node, Paste Properties, Model After).</td>
</tr>
<tr>
<td>UseFiltr</td>
<td>Boolean</td>
<td>Enables use of hierarchy filters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> No longer applicable to current product, retained only for backward compatibility.</td>
</tr>
<tr>
<td>UseMerge</td>
<td>Boolean</td>
<td>Enables use of Merge methodology for inactivates and deletes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td>ValSec</td>
<td>Boolean</td>
<td>Verifies security for validations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td>VerSec</td>
<td>Boolean</td>
<td>Validates security for verifications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True to enable, otherwise False</td>
</tr>
<tr>
<td>WarnHL</td>
<td>Integer</td>
<td>Number of nodes to trigger a warning prompt on the hot list</td>
</tr>
<tr>
<td>WebBook</td>
<td>String</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Book to indicate the exports available on the Web client (the specified book must be visible to the Web user)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For standard books, be sure to include the prefix, &quot;std:&quot;&quot;. For example, std:Web_Book1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If no value is supplied or an invalid book name is entered or a book that is not visible to the Web user is entered, then all exports visible to the Web user are available on the Web client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Do not put a space between the &quot;std:&quot; prefix and book name, otherwise it is considered an invalid book as described above.</td>
</tr>
<tr>
<td>System Preference</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WebExcludeNodeProps</td>
<td>String</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only. Comma-separated list of node properties to exclude from the Web client.</td>
</tr>
<tr>
<td>WebExDir</td>
<td>String</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only. Default export output directory on the Web server. To specify a directory, it must already exist. If blank, then exports write to C: .</td>
</tr>
<tr>
<td>WebExportFailureTriggerFile</td>
<td>String</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only. File name that is created if the Web export fails (the file is created in the export output directory).</td>
</tr>
<tr>
<td>WebExportTrigger</td>
<td>String</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only. File name that is created upon successful completion of the Web export (the file is created in the export output directory).</td>
</tr>
<tr>
<td>WebFindFunction</td>
<td>String</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only. Determines the type of Find mode (basic or enhanced) to use on the Web client. Default is Enhanced.</td>
</tr>
<tr>
<td>WebLoadTrigger</td>
<td>String</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only. Complete file path to trigger file that is created after the Web server refresh is complete.</td>
</tr>
<tr>
<td>WebVers</td>
<td>Integer</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only. The number of versions to load. Default is 2.</td>
</tr>
<tr>
<td>WebVersionList</td>
<td>String</td>
<td><strong>Note:</strong> This system preference applies to Data Relationship Management Web Publishing only. Comma-delimited list of versions to load. If populated, the specified versions are loaded in the order listed. If not populated, then the first n versions are loaded based on the value of WebVers. Default is DefaultCurrentVersion,DefaultPreviousVersion.</td>
</tr>
</tbody>
</table>

### Setting Up Approval Change Tracking

The approval change tracking system in Data Relationship Management provides the ability for the client to define approval groups and tie them to an approval flag that is triggered by a set of...
properties or special actions. This allows for normal users to make changes and then for approvers to see what needs to be approved by running a simple query and then set the approval flag as needed.

The four systems preferences that determine the behavior of the approval change tracking in Data Relationship Management are described below.

**UseChangeApproval**

This preference turns on or off the use of change approval in Data Relationship Management. Valid values are True and False.

**ApprovalGroups**

If UseChangeApproval is True, then this preference defines the approval groups used in the system. It is a comma-delimited list of group names. Note that these are not the Security Node Access Groups, just names to identify the approval groups. For example, Sales, Treasury.

**ApprovalGroupTrackProperties**

If UseChangeApproval is True, then this preference defines properties that are tracked that will trigger a change of the approval flag to false for this group. The format for this preference 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 abbrevs. For example, Sales[SalesGroup,[NodeMoved]],Treasury[AccountDescription,[NodeAdded]].

Special actions that can be placed in the property list are:

- `{NodeAdd}` - Triggers the Approval Needed mechanism on a node being added
- `{NodeInactivate}` - Triggers the Approval Needed mechanism on a node being inactivated
- `{NodeReactivate}` - Triggers the Approval Needed mechanism on a node reactivated
- `{NodeInsert}` - Triggers the Approval Needed mechanism on a node inserted
- `{NodeRemove}` - Triggers the Approval Needed mechanism on a node removed
- `{NodeMove}` - Triggers the Approval Needed mechanism on a node moved

**ApprovalPropertyByApprovalGroup**

If UseChangeApproval is True, then this preference 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 for this preference is xxx:bbbb,yyy:cccc…where xxx and yyy are sales groups defined in the ApprovalGroups preference and bbbb and cccc are the abbrevs for the global boolean properties to be used to store the approval flag for the groups.

For example, Sales:SalesApprovedFlag,Treasury:TreasuryApprovedFlag.

**Setting Up Change Tracking**

The change tracking feature provides the ability to monitor if a node is added or if a change has been made to a node property or relationship. It also records the name of the user who added
or changed the node and the date and time the node was added or changed. Five properties must be created and six system preferences must be configured to enable the track changes feature.

Creating Properties

You must create the five properties that Data Relationship Management uses to capture and record change events. Use the information in the following table to create the properties.

Table 6  Track Changes Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Label</th>
<th>Data Type</th>
<th>Property Level</th>
<th>Property Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddedBy</td>
<td>Added By</td>
<td>String</td>
<td>Global Node</td>
<td>Defined</td>
</tr>
<tr>
<td>AddedOn</td>
<td>Added On</td>
<td>String</td>
<td>Global Node</td>
<td>Defined</td>
</tr>
<tr>
<td>ChangedBy</td>
<td>Changed By</td>
<td>String</td>
<td>Global Node</td>
<td>Defined</td>
</tr>
<tr>
<td>ChangedOn</td>
<td>Changed On</td>
<td>String</td>
<td>Global Node</td>
<td>Defined</td>
</tr>
<tr>
<td>NodeChanged</td>
<td>Node Changed?</td>
<td>Boolean</td>
<td>Global Node</td>
<td>Defined</td>
</tr>
</tbody>
</table>

Note:

When you copy an existing version, there is an option to clear changed properties. Select this option to clear the five change tracking properties for all nodes in the new version. As changes occur in the new version, the properties can be used to identify and report on nodes that are added or changed.

Configuring System Preferences

Following are the six system preferences used to determine change tracking behavior.

- AddedByProperty — stores the name of the property to use to capture the user name when a new node is created. The value in the system preference must match the name (not the label) of an existing string, defined, global node property. The Data Relationship Management service will generate an error if the system preference value does not exactly match the property name, or if the property is not a global node defined string.

- AddedOnProperty — stores the name of the property to use to capture the date/time when a new node is created. The value in the system preference must match the name (not the label) of an existing string, defined, global node property. The Data Relationship Management service will generate an error if the system preference value does not exactly match the property name, or if the property is not a global node defined string.

- ChangedByProperty — stores the name of the property used to capture the user name when a node is modified. The value in the system preference must match the name (not the label) of an existing string, defined, global node property. The Data Relationship Management service will generate an error if the system preference value does not exactly match the property name, or if the property is not a global node defined string.
- ChangedOnProperty — stores the name of the property used to capture the date/time when a node is modified. The value in the system preference must match the name (not the label) of an existing string, defined, global node property. The Data Relationship Management service will generate an error if the system preference value does not exactly match the property name, or if the property is not a global node defined string.

- NodeChangedProperty — stores the name of the property used to capture the fact that a node has been modified since the version was created. The value in the system preference must match the name (not the label) of an existing boolean, defined, global node property. The Data Relationship Management service will generate an error if the system preference value does not exactly match the property name, or if the property is not a global node defined boolean.

- TrackChanges — enables or disables the change tracking feature within Data Relationship Management. This preference should only be set to True when all other configuration steps have been completed properly, otherwise the Data Relationship Management service will generate an error.

**Troubleshooting Change Tracking**

If the Track Changes system preference is enabled before the configuration of properties and system preferences is complete, the Data Relationship Management engine will generate an error. The following message is recorded in the system event log: “Invalid NodeChangedProperty property in system preferences. Contact system administrator.”

To resolve this problem, you can either disable the Track Changes system preference or configure the other required system preferences and restart the service.
About User Management

Data Relationship Management provides four levels of user account privileges:

- System Admin
- Functional Admin
- Security Admin
- User

The multiple administrative levels enable more targeted allocation of Data Relationship Management privileges across the organization. For more information on the Admin user types, see the following sections:

- “System Admin” on page 43
- “Functional Admin” on page 43
- “Security Admin” on page 44

Table 7 Administrative Privileges

<table>
<thead>
<tr>
<th>Function</th>
<th>System Admin</th>
<th>Functional Admin</th>
<th>Security Admin</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Users</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Define Property Categories</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Define Node Access Groups</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Assign Node to Node Access Groups</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Access to the Leaf Access and Limb Access tabs in the property editor</td>
</tr>
<tr>
<td>Function</td>
<td>System Admin</td>
<td>Functional Admin</td>
<td>Security Admin</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------</td>
<td>------------------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Define Properties</td>
<td>Yes</td>
<td>No*</td>
<td>No</td>
<td>*The Functional Admin can modify property list values and lookup tables</td>
</tr>
<tr>
<td>Modify System Preferences</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Define Validations</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Assign Validations</td>
<td>Yes</td>
<td>*Yes</td>
<td>No</td>
<td>*The Functional Admin has access to the Assign Menu on the Version control window and the Validation and Verification tabs in the property editor</td>
</tr>
<tr>
<td>Define User Properties</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Define Node Types</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Define System Categories</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Import</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Blender</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Create Versions (New, Copy, Save, Delete)</td>
<td>Yes</td>
<td>Yes*</td>
<td>No</td>
<td>*The Functional Admin can delete only an expired version</td>
</tr>
<tr>
<td>Change Version Status</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Read Access to all Versions, Hierarchies, Nodes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Read/Write Access to Working &amp; Submitted Versions</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Create Hierarchies (New, Delete)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Change Hierarchy (Name, Description, Order, System Category)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Remove Duplicates, Clear All Below (From main menu)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Delete/Destroy Orphans</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Annul All Below</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>System Admin</td>
<td>Functional Admin</td>
<td>Security Admin</td>
<td>Comments</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>Node Operations (Take, Put, Insert, Remove, Delete, Inactivate)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Create Standard Queries, Compares, Exports and Books</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Edit Global Inheriting Property outside of the Controlling Hierarchy</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Disable Real Time Validations on a Detached Version</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

Keep the following information in mind:

- Deleting a user also deletes the exports saved by the user.
- You cannot rename users.
- Certain system preferences specify the level of user allowed to perform certain actions. The possible values for these system preferences are Admin, Funct, Any, and None.
  - Admin indicates that only system administrators have access to the feature.
  - Funct indicates access for either functional or system administrators.
  - Any indicates access for any user level.
  - None indicates no access.
- A system preference is not set for a security administrator, because this user level is limited to read-only access for hierarchy data.

### System Admin

System Admins have access to all administrative functions in Data Relationship Management. This includes access to all versions, hierarchies, and all nodes within hierarchies and is only limited by the version status of finalized and expired. System Admins are not limited by node access groups or property categories.

System Admins can save and edit standard queries, exports, and compares and access blender, import, and automator functions. System Admins have access to the leaf/limb categories and the validation/verification categories via the Show All Properties pop up menu command.

### Functional Admin

Functional Admins have access to all versions, hierarchies, and all nodes within hierarchies and are only limited by the version status of finalized and expired. Functional Admins are not limited by node access groups or property categories.
Functional Admins do not have access to any of the administrative functions in Data Relationship Management except for properties, where they have read-only access to all property information except for lists and lookup tables.

Functional Admins can save and edit standard queries, exports, and compares and access blender, import, and automator functions.

Functional Admins also have access to the leaf/limb categories and the validation/verification categories via the Show All Properties pop up menu command.

**Security Admin**

Security Admins have read-only access to all versions, hierarchies, and all nodes within hierarchies. They are not limited by node access groups or property categories.

Security Admins do not have access to any of the administrative functions in Data Relationship Management except for user management.

Security Admins do not have access to the following:

- Leaf/limb categories
- Validation/verification categories
- Show All Properties pop up menu command
- Save and edit standard queries, exports, and compares
- Blender and Import functions
Managing Users

➤ To open the User Management dialog box, from the Data Relationship Management main menu, select Admin, then User Management.

For more information, see the following:

- “Defining Users” on page 46
- “Assigning Node Access Groups” on page 47
- “Assigning Property Categories” on page 47

Toolbar buttons are available for common functions:

- Create a user
  See “Creating Users” on page 47.
- Delete a user
- Change a password
- Lockout/Unlock User
**Defining Users**

The Current drop-down list shows all users currently defined. If you select a user from the list, the user parameters are displayed.

You can edit all but two of the following user definition parameters:

- User ID (read only)
- User Name (read only)
- Full Name
- Department
- Phone
- User Type—One of the four available user types: User, Security Admin, Functional Admin, System Admin.

**Assigning an Authentication Method**

You can set up users to be authenticated internally (within Data Relationship Management) or externally (via an authentication repository).

**Note:**

The authentication section of the New User dialog box is enabled only if the AuthMethod system preference is set to mixed.

*External authentication* means that the user login information is stored outside the application. The information is maintained in a central authentication directory, such as Lightweight Directory Access Protocol (LDAP) Directory, Microsoft Active Directory, or Windows NT LAN Manager.

An *authentication directory* is a centralized store of user information such as login names and passwords, and perhaps other corporate information. The repository functions like a telephone directory. The authentication directory probably contains much more than user names and passwords; for example, it may include e-mail addresses, employee IDs, job titles, access rights, and telephone numbers. It may also contain objects other than users; for example, it may contain information about corporate locations or other entities.

In order to use external authentication, your organization must have an authentication directory that contains corporate user information. Additionally, you must modify the XML-based security configuration file associated with your product to specify correct information pertaining to your corporate authentication directory.

The following types of authentication repositories are supported:

- Windows NT LAN Manager (NTLM) on NT 4.0 or higher, Windows 2000, and Windows 2003
- Lightweight Directory Access Protocol (LDAP) version 3 or higher
- Microsoft Active Directory server (MSAD), Windows 2000 sp3 or higher

**Note:**
This feature requires access to the Common Security Services component of Shared Services to perform the authentication.

For each user, select one of the following authentication methods:
- Manage User Within Data Relationship Management—Users are only authenticated internally within Data Relationship Management as in previous releases
- Manage User in Active Directory, LDAP, or NTLM—Users are authenticated by an external system

**Assigning Node Access Groups**
You use the Node Access Groups tab to assign node access groups to a user. The list on the right shows groups to which the user is assigned. The list on the left shows groups to which the user can be assigned.

➢ To include a user in a group or exclude a user from a group, select the group and the appropriate arrow.

**Assigning Property Categories**
You use the Property Categories tab to assign property categories to a user. The list on the right shows categories to which the user is assigned. The list on the left shows categories to which the user can be assigned.

➢ To change a user’s assignment, select a category, and click the appropriate arrow button.

The Read Only check box indicates whether the user has read and write or read only access to the selected category.

**Creating Users**
When creating a user account, you can use the Clone User option. This copies information (Department, Phone, User Type, Node Access Group and Property Category assignments) from an existing user to the new user. Otherwise, the information for the new user can be entered in each field. For more information, see “Defining Users” on page 46.
The password information in the New User dialog box is only used if the AuthMethod system preference is set to internal or mixed.

The password that you assign when you create a user expires based on the following system preferences:

- **PasswordDuration**—Determines the number of days until a password expires.
- **PasswordWarningPeriod**—Determines how a user is warned regarding password expiration.
  - 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 expiration. After the five-day period, the user cannot login without changing the password.

When an administrator creates a user or changes a user’s password, two options are available:

- **User must change password at next logon**—The user must change the password on the next login attempt. The expiration period starts after the change is made.
- **Password never expires**—The user can use the password indefinitely.

If the **PasswordPolicyEnabled** system preference is set to True, the password must contain three of the following elements:

- Uppercase letters
- Lowercase letters
- Numbers
Use change password to set the new 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.

**Locking Out Users**
The Admin can lock out a user and provide a lockout reason by clicking the User Lockout toolbar button in the User Management dialog box. In addition, the Admin can clear a lockout (manual or automatic).

**Security Configuration Best Practices**
When you configure overall security within Data Relationship Management, you must address more than user management. Additional issues that you must address are introduced in the following topics and discussed in detail in “Working with Node Access Groups” on page 50.

**Assigning Data Relationship Management Users**
- Assign every Data Relationship Management user a personal login. Avoid using generic or department-wide logins.
- Determine which users require administrative access (security, functional, or system).

**Grouping Properties into Categories**
- Identify property categories that group properties for the convenience of users that need access to properties
- Use property categories to reflect upstream and downstream systems that interface with Data Relationship Management.
- Use property categories to support special projects such as mergers and acquisitions. Such properties are often removed following the completion of the project.

**Assigning User Access to Categories**
Determine whether users need full access or read-only access to property categories. This may help determine how properties should be categorized.
Assigning Access per Node Access Group

- Determine the need for granular security restrictions for both limbs and leaves.
- Create node access groups and assign users to the groups.
- Determine whether each group needs to be local or global.
- Define access rights for each group wherever required in each hierarchy. Access rights must be defined separately for limbs and leaves.
  - Access rights are an inherited property, so administration is simplified if rights are assigned at the highest appropriate level in each hierarchy.
  - If a user does not have access to at least one node in a hierarchy, the user does not see the hierarchy displayed in the Version control window.
  - If a user has access to any node in a hierarchy, the user has at least read-only access to every node in the hierarchy and can thus navigate the hierarchy.
- If a user does not have any access to at least one node of at least one hierarchy in a version, then the user does not see the version in the Version control window.

Working with Node Access Groups

You use node access groups to control user access to hierarchies and nodes. You can assign Data Relationship Management users to specific groups that have permissions to specific nodes within Data Relationship Management. Security based on groups is more flexible than the on or off access determined by a user’s allocation to a property category because multiple tiers of access levels can be assigned to a specified group.

Table 8  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>Limited Insert</td>
<td>Enables insertion of a node for which the user has (at least) global insert privilege.</td>
<td>Insert</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 or edited</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, copy, 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 Limited Insert access levels are granted. Assignment of the Add access level implies that all other access levels are granted.
● Access levels are maintained separately for limbs and leaves.
● Node access groups are defined only by a system administrator.
● Node access groups by default are a local, inheriting property but can be set to global.
● Global node access groups can be created and must have a controlling hierarchy defined.
● A user can be assigned to multiple node access groups. 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.
● 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.

Using Node Access Groups

➤ To open the Node Access Groups dialog box, from the Data Relationship Management main menu, select Admin, then Node Access Groups.

![Node Access Groups dialog box]

Toolbar buttons are available for these common functions:
● Use to create a node access group

● Use to delete a node access group

**Defining Node Access Groups**

The Current list shows all node access groups that are currently defined. When you select a group from the drop-down list, its parameters are displayed and are available for edit.

**Assigning Users to Node Access Groups**

You use the two lists at the bottom of the dialog box to assign users to a node access group. The Selected Users list shows users that are currently members of the group. The Available Users list shows users that are not currently members.

➤ To include a user in a group or exclude a user from a group, select the user, and click the appropriate arrow.

**Defining Controlling Hierarchies**

This tab is only displayed when a global node access group is selected. You use it to define the controlling hierarchy for the selected group. For example, a user with Limited Insert access to a specified hierarchy can insert nodes for which the user has global insert privileges. This fact has the following implications:

● The user is a member of a global node access group. The group has a controlling hierarchy defined for the specified version.

● The node access group was granted Insert access (or higher) to the controlling hierarchy.

● The node to be inserted exists in the controlling hierarchy.

Data Relationship Management versions are listed on the left of the tab.

➤ To define the controlling hierarchy for the selected group, select a version from the list, and then, on the right, select the desired hierarchy.

**Using Node Access Group Security**

Node Access Group security is applied by a system administrator within Data Relationship Management.

➤ To set node access group access levels in a hierarchy:

1 Ensure that appropriate node access groups are created and appropriate users are assigned to the groups.
2 Log on to Data Relationship Management as a user with administrative privileges.

3 Open a version and hierarchy, and navigate to the node at which you want to define access levels.

4 View properties for the selected node.

5 In the property editor, right-click, and ensure that Show All Properties is selected from the Action menu. Limb Access and Leaf Access tabs should now be visible in the property editor.

6 For the Limb Access and Leaf Access tabs, from the Value list select an access level for each group. Users within the listed groups have access to the node at the specified level.
Using the Properties Dialog Box

To display the Properties dialog box, from the Data Relationship Management main menu, select Admin, then Properties.

Toolbar buttons are available for common functions:

- Use ![to create a property](image)
- Use ![to delete a property](image)

The Current list shows all defined properties. When you select a property from the list, its parameters are displayed and available for edit.
Property Definition Tab

This tab, available for all property types, displays basic property information. Basic property definitions are listed below.

Table 9  Property Definitions Tab

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Read-only system ID assigned by Data Relationship Management when the property is created</td>
</tr>
<tr>
<td>Name</td>
<td>Name used in formulas and Deriver parameters</td>
</tr>
<tr>
<td>Description</td>
<td>Text description</td>
</tr>
<tr>
<td>Label</td>
<td>Label displayed in Data Relationship Management</td>
</tr>
<tr>
<td>Item</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Data Type</td>
<td>The data domain that can be assigned to the property; selected from a list of available property data types (see Table 10 on page 57)</td>
</tr>
<tr>
<td>Property Level</td>
<td>Level of property definition: local node, global node, hierarchy, or version</td>
</tr>
<tr>
<td>Property Type</td>
<td>Available property types:</td>
</tr>
<tr>
<td></td>
<td>● Defined—defined by user in Data Relationship Management</td>
</tr>
<tr>
<td></td>
<td>● Lookup—lookup based on another property and a lookup table</td>
</tr>
<tr>
<td></td>
<td>● Derived—determined by way of a Deriver class</td>
</tr>
<tr>
<td></td>
<td>● RW Derived—derived but can be overridden by a user in Data Relationship Management</td>
</tr>
<tr>
<td>Lookup Property</td>
<td>Property to be used for lookup, if property type is Lookup</td>
</tr>
<tr>
<td>Default Value</td>
<td>Default value for the property</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>Minimum value for the property</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>Maximum value for the property</td>
</tr>
<tr>
<td></td>
<td>If the property data type is string, this is the maximum string length.</td>
</tr>
<tr>
<td>List</td>
<td>Determines whether property values can be selected only from a pre-defined list of values: True or False</td>
</tr>
<tr>
<td>Inheritable</td>
<td>Defines the property as Inheriting; True or False</td>
</tr>
<tr>
<td></td>
<td>Note: This flag has no effect on property types Derived and RW Derived except in the special case where property derivers such as AncestorProp or DualAncestorProp are used and the property is global. (In these cases, although the property is not literally inheriting values, the Inheritable flag should be set to enable the specification of a controlling hierarchy.)</td>
</tr>
<tr>
<td>Hidden</td>
<td>Defines property as hidden in the Data Relationship Management Property Window: True or False</td>
</tr>
<tr>
<td>Column Width</td>
<td>Width for fixed-width columns</td>
</tr>
<tr>
<td>Class Name</td>
<td>The Deriver class name, if the property type is Derived or RW Derived; can be selected from a list of available property deriver classes (see Table 11 on page 59)</td>
</tr>
</tbody>
</table>

The following table describes the property data types.

**Table 10  Property Data Types**

<table>
<thead>
<tr>
<th>Property Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AscGroup</td>
<td>Associated node group. Points to multiple nodes. The nodes point back to the AscGroup node and to each other. Analogy: Fraternity.</td>
</tr>
<tr>
<td></td>
<td>Caution! Associated nodes can only be set when the original node is edited in the user interface. Import files that modify the original node do not change the associated nodes.</td>
</tr>
<tr>
<td></td>
<td>Note: Use this data type with global nodes only.</td>
</tr>
<tr>
<td>AscNode</td>
<td>Associated node. Points to a single other node. The node pointed to points back to the AscNode node. Analogy: Marriage.</td>
</tr>
<tr>
<td>Property Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Caution!</strong></td>
<td>Associated nodes can only be set when the original node is edited in the user interface. Import files that modify the original node do not change the associated nodes.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>Use this data type with global nodes only.</td>
</tr>
<tr>
<td>AscNodes</td>
<td>Associated node group. Points to multiple nodes. The nodes pointed to point back to the AscNodes but not each other. Analogy: Friends.</td>
</tr>
<tr>
<td><strong>Caution!</strong></td>
<td>Associated nodes can only be set when the original node is edited in the user interface. Import files that modify the original node do not change the associated nodes.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>Use this data type with global nodes only.</td>
</tr>
<tr>
<td>Boolean</td>
<td>True or False</td>
</tr>
<tr>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Float</td>
<td>Floating point value</td>
</tr>
<tr>
<td>FormatMemo (FrmtMemo)</td>
<td>Formatted memo — retains all formatting (spaces, tabs, new lines, etc) to the text</td>
</tr>
<tr>
<td>GlobalNode</td>
<td>Points to a global node; when value is assigned it shows node name only in the value field of the property editor</td>
</tr>
<tr>
<td>Group</td>
<td>List of comma-delimited items</td>
</tr>
<tr>
<td>Hier</td>
<td>Points to a hierarchy</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer value</td>
</tr>
<tr>
<td>LeafNode</td>
<td>Points to a leaf node</td>
</tr>
<tr>
<td>LimbNode</td>
<td>Points to a limb node</td>
</tr>
<tr>
<td>ListGroup</td>
<td>Check list of items</td>
</tr>
<tr>
<td>Memo</td>
<td>Memo field — formatting is not saved and data is merged into a single line of text</td>
</tr>
<tr>
<td>MultiNode (MultiNod)</td>
<td>Points to multiple nodes</td>
</tr>
<tr>
<td>Node</td>
<td>Points to a node; when value is assigned it shows hierarchy name and node name in the value field of the property editor</td>
</tr>
<tr>
<td>NodeProps (NodePrps)</td>
<td>Points to the properties of a node</td>
</tr>
<tr>
<td>Property (Prop)</td>
<td>Points to a property</td>
</tr>
<tr>
<td>Query</td>
<td>Points to a query</td>
</tr>
<tr>
<td>RangeList (RangeLst)</td>
<td>Defines a range of values — takes only integer values</td>
</tr>
<tr>
<td>Sort</td>
<td>Used for sorting</td>
</tr>
<tr>
<td>SortProp</td>
<td>Points to a SortProp</td>
</tr>
<tr>
<td>StdQuery</td>
<td>Points to a standard query</td>
</tr>
<tr>
<td>String</td>
<td>String value</td>
</tr>
</tbody>
</table>
Table 11 describes the property deriver classes that are available and indicates which classes are used most frequently. Most of these classes were created for specialized purposes that can now be accomplished using the more generalized Formula class property. However, all of the classes can still be accessed directly.

**Note:**
You can simulate AncestorProp, DualAncestorProp, and AscNodeProp in the Formula class property but it is easier to access these classes directly. InvertedLevel cannot be done using the Formula class property.

<table>
<thead>
<tr>
<th>Property Deriver Class</th>
<th>Usage Frequency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>High</td>
<td>Returns a value based on a specified formula</td>
</tr>
<tr>
<td>AncestorProp</td>
<td>Medium - Special Use</td>
<td>Traverses up the local or controlling hierarchy (based on the Global flag) until it hits a node where the specified property equals the specified value and then returns the specified return property. Although this property is not literally inheriting, the Inheritable flag must be set to enable the specification of the controlling hierarchy.</td>
</tr>
<tr>
<td>AscNodeProp</td>
<td>Medium - Special Use</td>
<td>Goes to the node pointed to by the specified node property and returns the specified property</td>
</tr>
<tr>
<td>DualAncestorProp</td>
<td>Low - Special Use</td>
<td>Traverses up the local or controlling hierarchy (based on the Global flag) until it hits a node where the specified properties equal the specified values and then returns the specified return property (Not literally inheriting, but the Inheritable flag must be set to enable the specification of the controlling hierarchy)</td>
</tr>
<tr>
<td>InvertedLevel</td>
<td>Low - Special Use</td>
<td>Provides an Essbase type level that starts at 0 for the leaf and increments up the ancestor chain (For nodes with multiple branches under them, the level is the maximum of the values from each branch)</td>
</tr>
<tr>
<td>Children</td>
<td>Rare - Use Formula</td>
<td>Returns the list of child nodes in a comma-delimited form</td>
</tr>
<tr>
<td>Concat</td>
<td>Rare - Use Formula</td>
<td>Concatenates the specified properties. String (as a constant) is also allowed for concatenation. <strong>Caution!</strong> If using a string it must be enclosed in double quotes, for example &quot;test&quot;.</td>
</tr>
<tr>
<td>DefaultProp</td>
<td>Rare - Use Formula</td>
<td>Returns the default value for the property</td>
</tr>
<tr>
<td>Property Deriver Class</td>
<td>Usage Frequency</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Format Type</td>
<td>Rare - Use Formula</td>
<td>Formats a string based on the specified Format String Property Name and Property Type. For information on the string property, see &quot;Format String Parameter&quot; on page 60.</td>
</tr>
<tr>
<td>FormattedDate</td>
<td>Rare - Use Formula</td>
<td>Formats a date based on the specified property and date format control.</td>
</tr>
<tr>
<td>Hiers</td>
<td>Rare - Use Formula</td>
<td>Returns a list of hierarchies.</td>
</tr>
<tr>
<td>IntrnPrefix</td>
<td>Rare - Use Formula</td>
<td>Returns a prefix.</td>
</tr>
<tr>
<td>ParentProp</td>
<td>Rare - Use Formula</td>
<td>Returns a property from the parent node.</td>
</tr>
<tr>
<td>SubString</td>
<td>Rare - Use Formula</td>
<td>Returns a substring.</td>
</tr>
<tr>
<td>TotalLeafs</td>
<td>Rare - Use Formula</td>
<td>Returns the number of leaves from the current leaf.</td>
</tr>
<tr>
<td>TotalLimbs</td>
<td>Rare - Use Formula</td>
<td>Returns the number of limbs from the current limb.</td>
</tr>
<tr>
<td>TotalNodes</td>
<td>Rare - Use Formula</td>
<td>Returns the number of nodes from the current node.</td>
</tr>
</tbody>
</table>

**Format String Parameter**

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

Format specifiers use the following form:

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

<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>[&quot;-&quot; ]</td>
<td>Left justification indicator (optional). Left justifies the result by adding blanks after the value. The default is to right-justify the result by adding blanks in front of the value.</td>
</tr>
<tr>
<td>[width]</td>
<td>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.</td>
</tr>
<tr>
<td>[&quot;.&quot; prec]</td>
<td>Precision specifier (optional).</td>
</tr>
<tr>
<td>type</td>
<td>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.</td>
</tr>
<tr>
<td>Type Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>d</strong></td>
<td>Decimal</td>
</tr>
<tr>
<td></td>
<td>The property value must be an integer. The value is converted to a string of decimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has less digits, the resulting string is left-padded with zeros.</td>
</tr>
<tr>
<td><strong>u</strong></td>
<td>Unsigned decimal</td>
</tr>
<tr>
<td></td>
<td>Similar to d but no sign is output.</td>
</tr>
<tr>
<td><strong>e</strong></td>
<td>Scientific</td>
</tr>
<tr>
<td></td>
<td>The property value must be a floating-point value. The value is converted to a string of the form &quot;-d.ddd...E+ddd&quot;. 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 &quot;E&quot; exponent character in the resulting string is always followed by a plus or minus sign and at least three digits.</td>
</tr>
<tr>
<td><strong>f</strong></td>
<td>Fixed</td>
</tr>
<tr>
<td></td>
<td>The property value must be a floating-point value. The value is converted to a string of the form &quot;-ddd.ddd...&quot;. 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.</td>
</tr>
<tr>
<td><strong>g</strong></td>
<td>General</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>n</strong></td>
<td>Number</td>
</tr>
<tr>
<td></td>
<td>The property value must be a floating-point value. The value is converted to a string of the form &quot;-d,ddd,ddd,ddd...&quot;. The &quot;n&quot; format corresponds to the &quot;f&quot; format, except that the resulting string contains thousand separators.</td>
</tr>
<tr>
<td><strong>m</strong></td>
<td>Money</td>
</tr>
<tr>
<td></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><strong>s</strong></td>
<td>String</td>
</tr>
<tr>
<td></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><strong>x</strong></td>
<td>Hexadecimal</td>
</tr>
</tbody>
</table>
The property value must be an integer value. The value is converted to a string of hexadecimal digits. If the format string contains a precision specifier, it indicates that the resulting string must contain at least the specified number of digits; if the value has fewer digits, the resulting string is left-padded with zeros.

**Date-Time Format Strings**

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

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

Creating Properties

➤ To create a property:

1 On the toolbar, click 📊.

The New Property dialog box is displayed:
2 Enter the following values:

- Property Name
- Description
- Label
- Data Type
- Property Level
- Property Type
- Category

3 Click OK to save the new property.

Using the Controlling Hierarchy Tab

This tab is available only if the property is defined as Global and Inheritable. In this case, the controlling version and hierarchy must be defined for the property.
When you select a version from the list, its hierarchies are displayed.

➤ To define the controlling hierarchy:

1. In the **Controlling Hierarchy** tab, select the hierarchy.
2. Click **Apply** to save the change.
3. If necessary, repeat steps 1 and 2 for other versions.

**Using the List Values Tab**

This tab is available only if List is selected in the Properties dialog box. This tab enables a list of valid values for the property to be defined.
To create a list value, enter the value in the field at the bottom and click Add. The new value will then be added to the list.

Use the buttons to the right of the list to complete the following tasks:

- Move a specified item up or down in the list
- Sort the list alphabetically
- Delete an item from the list

**Using the Lookup Table Tab**

This tab is available only if the property has a property type of Lookup. The value of a Lookup property is determined by the value of another property and a lookup table. The purpose of this tab is to define a set of lookup and result values. If the value of the property pointed to by the first property matches a Lookup value, the first property is assigned the corresponding Result value.
To create a Lookup Key, Result Value pair:

1. In the Lookup Table tab, enter the values in the Lookup Key and Result Value text boxes.
2. Click Add.

The new values are added to the list.

To delete a Lookup Key, Result Value pair from the list, select the row to delete and type Ctrl+Del to delete it.

Using the Property Params Tab

This tab is available only if the property is of the derived property type. The value of a derived property is determined using a property deriver class (see Table 11 on page 59). One commonly used class is Formula, which enables the user to create a custom formula to determine the property value.
On the Property Params tab, when using the formula deriver (Label = Formula), select the entry in the Value column and click the [...] button to display the Formula Tester dialog box.

For other types of derivers, where only a string value is needed, select the Value column and click the [...] button to open a text editor which enables editing of the specific value.

**Categories Tab**

This tab is available for all property types and determines the property categories to which the specified property belongs. A property must belong to at least one category.
The list on the right shows the categories to which the property currently belongs and the list on the left shows all other categories. You can use the arrow buttons to include or exclude categories from the lists.

**Property Categories**

Property categories enable the grouping of Data Relationship Management properties and are used to control the allocation of security privileges to sets of properties. The system administrator can create as many property categories as needed.

Data Relationship Management includes the built-in property categories described in the following table:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Properties related to the 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-only access cannot edit values but can view them. Properties cannot be assigned to this category.</td>
</tr>
<tr>
<td>Stats</td>
<td>Properties that provide statistical information about a node such as number of children and number or siblings</td>
</tr>
<tr>
<td>Validation</td>
<td>Validations assigned for the node—one property for each validation</td>
</tr>
<tr>
<td>Verification</td>
<td>Verifications assigned for the node—one property for each verification</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 the system administrator can restrict user access to specific categories and the node types can be filtered. The Validation, Verification, Leaf Access, and Limb Access categories are available only to system administrators.
To display the Property Categories dialog box, from the Data Relationship Management main menu, select Admin, then Property Categories.

Toolbar buttons are available for common functions:

- Use to create a property category
- Use to delete a property category
- Use to change the property category order

Property categories can be ordered individually using the arrow buttons or sorted alphabetically.
Validations and Verifications Overview

Validations and verifications are tests ensuring hierarchy rules are observed, and enabling the enforcement of business rules:

- **Validations**—Run automatically (in real-time) as users are editing the structure and properties of hierarchies. Validations are performed for the node being edited and also for its parent.

- **Verifications**—Run on-demand (as a batch) after users make a set of changes. Users can choose specific verifications or run a set of verifications defined by the system administrator.

For example, assume that ABC Widgets has a business rule that requires that “… a sales representative who is assigned more than twenty customers must obtain special approval from the marketing department.“

- A validation that supports this business rule within Data Relationship Management permits no more than 20 customers to be added to a sales representative node.

- A verification that supports this business rule reports on all of those sales representatives with more than 20 customers assigned.
Using the Validations Dialog Box

➤ To display the Validations dialog box, from the Data Relationship Management main menu, select Admin, then Validations.

![Validations Dialog Box Image]

Toolbar buttons are available for common functions:

- Use ![Create Icon] to create a validation
- Use ![Delete Icon] to delete a validation

The Current list shows all currently-defined validations. When a validation is selected from the list, its parameters are displayed and available for edit.

Validation Definition Tab

This tab displays the definitions for the current validation:

- Current (read only)
- ID (read only)
- Label
- Description
- Inheriting (read only)
- Level (read only)
- Type (read only)
Validation and Verification Levels

The level defines the circumstances and the extent (node, hierarchy, or version) to which the validation or verification runs.

Table 13  Validation and Verification Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>The most commonly used level. Can be a validation or a verification. Can be assigned and run at the node, hierarchy, or version levels.</td>
<td>Use to determine whether a text string has a valid length</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Verification only. Reviews properties in a hierarchy to ensure criteria are met. Can be assigned and run at the hierarchy or version levels (in the Version control window).</td>
<td>Use to ensure that a hierarchy has no more than 10,000 nodes</td>
</tr>
<tr>
<td>Version</td>
<td>Verification only. 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>Remove</td>
<td>Validation only. Fired when an attempt is made to remove or delete a node from a hierarchy. Can be used to prevent specified types of nodes from being deleted. Can be assigned and run at the hierarchy or version level (in the Version control window).</td>
<td>Use to prevent the deletion of cost center nodes from a hierarchy</td>
</tr>
<tr>
<td>Move</td>
<td>Similar to the Remove level. A validation triggered when an attempt is made to move a node. Assigned at the hierarchy level.</td>
<td>Use to prevent the moving of cost centers within a hierarchy</td>
</tr>
<tr>
<td>Global node</td>
<td>Validation or verification, 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 have a name with more than eight characters.</td>
</tr>
<tr>
<td>Merge</td>
<td>Validation only. Similar to the global node validation, but triggered 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>
</tbody>
</table>

Validation Classes

Validations can be created from a set of existing Validation Classes. The default classes and their required parameters are listed in Table 14.

In general, the majority of the validations are based on CustPropQuery. There are many other classes that are valid for special use scenarios. There are a number of classes that are better done by using the Prop Query based classes but have been left in for backwards compatibility. In addition, there are a number of deprecated classes. These classes should not be used since they
depend on specific property names and other non-flexible information. They are only for backwards compatibility and may not be supported in the future. The last set of classes are testing only and should never be used in a production implementation.

Table 14  Validation Classes

<table>
<thead>
<tr>
<th>Validation Class</th>
<th>Usage Frequency</th>
<th>Level</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustPropQuery</td>
<td>High</td>
<td>Node</td>
<td>Verifies using predefined standard query and expected result</td>
<td>Property query name, Failure value</td>
</tr>
<tr>
<td>GlobalPropQuery</td>
<td>Medium</td>
<td>Global</td>
<td>Verifies using predefined standard query and expected result</td>
<td>Property query name, Failure value</td>
</tr>
<tr>
<td>MergeEquiv</td>
<td>Medium - Special Use</td>
<td>Merge</td>
<td>Verifies that the affected node and merge node have the same value for the specified property</td>
<td>Property (Must be Global)</td>
</tr>
<tr>
<td>NoBoolBranch</td>
<td>Medium - Special Use</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>NoPropBranch</td>
<td>Medium - Special Use</td>
<td>Node</td>
<td>Verifies that the specified property is set at least once on a specified branch</td>
<td>Property</td>
</tr>
</tbody>
</table>
| RequiredField    | Medium - Special Use | Node         | Verifies that, for all nodes for which the specified property has a specified value, each property in the required list has a value:  
  ● If the Reject Default Records flag is True, each property in required list must have a value other than the default  
  ● If the Reject Default Records flag is False, then default values are acceptable | Property, Value, Reject Default Records, Required Properties |
<p>| SingleBoolBranch | Medium - Special Use | Node         | Verifies that the specified boolean property is set to True only once per branch | Property                                      |
| SinglePropBranch | Medium - Special Use | Node         | Verifies that the specified property is set only once per branch            | Property                                      |
| UniquePropBranch | Medium - Special Use | Node         | Verifies that the specified property has unique value within a branch       | Property                                      |
| UniqueProp       | Medium - Special Use | Node         | Verifies that the specified property has no duplicate values within a hierarchy (If the Include Defaults flag is False, nodes with the default value are not included) | Property, Include Defaults                   |
| VersionUniqueProp| Medium - Special Use | Global       | Verifies that the specified property has no duplicate values within a version (If the Include Defaults flag | Property, Include Defaults                   |</p>
<table>
<thead>
<tr>
<th>Validation Class</th>
<th>Usage Frequency</th>
<th>Level</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateRangeCheck</td>
<td>Low - Special Use</td>
<td>Node</td>
<td>Verifies that From Date is earlier than or equal to To Date</td>
<td>From Date Property, To Date Property</td>
</tr>
<tr>
<td>MaxHierNodes</td>
<td>Low - Special Use</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>MaxVersionNodes</td>
<td>Low - Special Use</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>MergePropSet</td>
<td>Low - Special Use</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>NoDefaults</td>
<td>Low - Special Use</td>
<td>Node</td>
<td>Verifies that no default values are used for the specified property</td>
<td>Property</td>
</tr>
<tr>
<td>PropRemove</td>
<td>Low - Special Use</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>VersionUnique2Prop</td>
<td>Low - Special Use</td>
<td>Global</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>BoolNodeInHier</td>
<td>Low - Use CustPropQuery</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>InvalidNameLength</td>
<td>Low - Use CustPropQuery</td>
<td>Node</td>
<td>Verifies that the node name is not equal to a specified length.</td>
<td>Length</td>
</tr>
<tr>
<td>MaxChildren</td>
<td>Low - Use CustPropQuery</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>PropLength</td>
<td>Low - Use CustPropQuery</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>StrandedParent</td>
<td>Low - Use CustPropQuery</td>
<td>Node</td>
<td>Verifies that all limb nodes have children</td>
<td>none</td>
</tr>
<tr>
<td>Validation Class</td>
<td>Usage Frequency</td>
<td>Level</td>
<td>Description</td>
<td>Parameters</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>StrPropEqual</td>
<td>Low - Use CustPropQuery</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>ContainAllProp</td>
<td>Low - Use GlobalPropQuery</td>
<td>Hierarchy</td>
<td>Verifies that the specified hierarchy contains all nodes where the specified property is True</td>
<td>Hierarchy, Property</td>
</tr>
<tr>
<td>ContainAllWith</td>
<td>Low - Use GlobalPropQuery</td>
<td>Hierarchy</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>HierFail</td>
<td>Do not use - testing only</td>
<td>Hierarchy</td>
<td>Automatically fails at hierarchy level for testing purposes</td>
<td>none</td>
</tr>
<tr>
<td>NodeFail</td>
<td>Do not use - testing only</td>
<td>Global</td>
<td>Automatically fails nodes at the version level for testing purposes</td>
<td>none</td>
</tr>
<tr>
<td>NodeFailRandom</td>
<td>Do not use - testing only</td>
<td>Node</td>
<td>Automatically fails the specified percentage of nodes for testing purposes</td>
<td>Failure percentage</td>
</tr>
<tr>
<td>ContainAllCC</td>
<td>Deprecated - do not use</td>
<td>Global</td>
<td>Verifies that a hierarchy contains all nodes marked as CostCenter and with FCFlag = False</td>
<td>Hierarchy</td>
</tr>
<tr>
<td>PropEquivCC</td>
<td>Deprecated - do not use</td>
<td>Node</td>
<td>Verifies that for all nodes marked as CostCenter, the first specified property equivalent to the second specified property</td>
<td>First property, Second property</td>
</tr>
<tr>
<td>PropEquivLeaf</td>
<td>Deprecated - do not use</td>
<td>Node</td>
<td>Verifies that, for all leaf nodes, the first specified property is equivalent to the second specified property</td>
<td>First property, Second property</td>
</tr>
<tr>
<td>VersionFail</td>
<td>Do not use - testing only</td>
<td>Version</td>
<td>Automatically fails at the version level for testing purposes</td>
<td>none</td>
</tr>
</tbody>
</table>

**Parameters Tab**

This tab enables validation parameters to be edited. The parameters control the behavior of the various validation classes. Thus, a validation can be generic and have its implementation controlled through its parameters. When you click in the Value column, an editor window is displayed, enabling you to modify the value.
Creating Validations and Verifications

➤ To create a validation or verification:

1 In the Validations dialog box, click on the toolbar.

   The New Validation dialog box is displayed.

2 In Class, select an existing validation class.

3 In the Level, select the appropriate level.

4 Enter the name, label and description.

5 For Validation Type, select either Validation (Real-time) or Verification (Batch).

   See “Validations and Verifications Overview” on page 71 for information explaining the difference between a verification and validation.

6 Click OK.

   The new validation or verification is ready for use.
Assigning Validations and Verifications

After you create validations and verifications, you can assign them to versions, hierarchies, and nodes.

**Note:**

When assigned at the version level, validations and verifications are inherited by all hierarchies and nodes within the version. When assigned at the node level, validations and verifications are inherited by all levels under the node.

To assign a validation or verification to a version or hierarchy:

1. **Right-click on the version or hierarchy and select one of the menu commands:**
   - Assign, then Validation
   - Assign, then Verification

   ![Assign Validations Dialog Box](image)

   ![Assign Verifications Dialog Box](image)

2. Select the validations or verifications to assign to the version or hierarchy and click **OK**.
To assign a validation or verification to a node:

1. Double-click a node to open the property editor dialog box.
2. Right-click on the tabs and select Multi-Line Tabs so you can easily see all tabs.
3. Select the Validation or Verification tab.

4. Select the validations or verifications that you want to apply to the node and click Save.
Managing System Categories

In This Chapter

Versions and Hierarchies Overview ................................................................. 81
Using the System Categories Dialog Box .......................................................... 81

Versions and Hierarchies Overview

System categories provide a way to group hierarchies within a version. In the Version Control Window, hierarchies assigned to a system category will be displayed under a folder for that category.

Data Relationship Management groups sets of hierarchies into Versions. A version represents a single, independent set of data that is arranged into related hierarchies. All Data Relationship Management maintenance is performed within a single version - users are not allowed to copy or move nodes across versions. The only Data Relationship Management features that work across versions are Compare and some Exports.

Versions are typically used for the following purposes:

- Represent a set of hierarchies used during a particular month (or other business cycle period.) Each month a new version is created.
- Differentiate between real data and test data during system testing.
- Make comparisons between different versions, to identify changes that are being made to the hierarchies across time.

New versions may be created by copying existing versions. However, once a version is created it is independent of changes to other versions.

Nodes and properties within a version are shared amongst the hierarchies within the version.

Using the System Categories Dialog Box

Note:

If you add, delete, or rename system categories, you must restart the Data Relationship Management service before the changes are available to all users. Changing the system category assignment of a hierarchy does not require a restart.
To display the System Categories dialog box, from the Data Relationship Management main menu, select Admin, then System Categories.

Toolbar buttons are available for common functions:

- Use to create a system category
- Use to delete a system category
- Use to reorder system categories

You can order categories individually by using the arrow buttons, or you can sort them alphabetically.
Note:

Only the system category name can be edited.

Hierarchies

This area shows the hierarchies currently assigned to the selected system category. When a version is chosen from the list, the Selected list shows all hierarchies within the version that are assigned to the system category. The Available list shows all hierarchies within the version that are not assigned to the system category. You can use the arrows to assign or unassign hierarchies for the category.

Hierarchies that are members of another system category have their hierarchy values displayed in braces ({}). Because a hierarchy can belong to only one system category, assigning a hierarchy to the current category removes its previous assignment. When you save the changes, they are displayed in the Version control window after it is refreshed.

You can also assign a hierarchy to a category by setting the appropriate system property for the specified hierarchy in the property editor.
### Property Editor

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserDisplayBy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UserHierFilter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UserHierSort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hier ID</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>Hier Name</td>
<td>Geography</td>
<td></td>
</tr>
<tr>
<td>Hier Descr</td>
<td>Geographical areas</td>
<td></td>
</tr>
<tr>
<td>Hier System C...</td>
<td>Chart</td>
<td></td>
</tr>
<tr>
<td>Hier Sort Order</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Hier Sort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hier Filter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top Node</td>
<td>North America</td>
<td></td>
</tr>
<tr>
<td>Top Node ID</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>
Categorizing Nodes by Node Types

- Determine the node types that are necessary to categorize nodes within a hierarchy.
- Identify properties that are relevant (or not relevant) to each node type.
- Identify validations and verifications that are relevant (or not relevant) to each node type.

Node Types

Node types are used to group nodes that share basic characteristics. The nodes within a specified node type share the following elements:

- Properties
- Validations
- Glyphs

Examples of node type usage are to identify cost centers, rollup nodes, forecast nodes, and so on. This enables enforcement of business rules such as “A cost center cannot be added to a GL account.”
Using Node Types

To open the Node Types dialog box, from the Data Relationship Management main menu, select Admin, then Node Types.

Toolbar buttons are available for common functions:

- Create a new node type
- Delete a node type

The Current list shows all node types currently defined. When a type is selected from the list, its parameters are displayed in the dialog box and are available for edit.

Using Properties

The list on the right shows assigned properties. The list on the left shows available properties.
To change the assignment, select a property from the relevant list, and click the appropriate arrow.

**Using Validations**

Any number of Validations can be assigned to a node type. As with the Properties tab, validations may be selected from the list of Available Validations and moved to and from the list of Selected Validations.

**Configuring Node Types**

To configure a node type:

1. Identify or create an existing node-level property (typically Local and Inheriting) that is used for each node to indicate its node type.

   For example, a property can be named Type of Node.

2. Using the Node Types dialog box, create one or more node types and assign a node glyph.

![Node Types dialog box](image)
3 From the available lists, assign the desired properties and validations.

4 For each node in the hierarchy, set the TypeOfNode property to a valid node type.

For example, you can set the TypeOfNode property to Cost Center. Leaving the TypeOfNode property blank indicates no node typing for this node and all available properties are displayed. Specifying an invalid node type also displays all available properties.

- Because the TypeOfNode property is Inheriting, it is necessary to set the property for a child of a node only if the child is to have a value different from the parent.
- Alternatively, you can assign TypeOfNode as a Derived property that automatically determines the appropriate value based on other characteristics of the specified node.
- At the hierarchy level, a default property named HierarchyNodeType points to the node-level property that indicates node type.

In this example, the HierarchyNodeType property points to the TypeOfNode property. After HierarchyNodeProperty is set, the node types are enforced for the hierarchy. When you edit properties for a node, only the properties assigned to the specified node type are shown.

**Note:**
You may need to refresh the hierarchy for node type changes to take affect.

**Disabling Node Types**

You disable a node type for a specified hierarchy by removing its value.

➤ To disable the use of node types for a specified hierarchy:
1 Right-click the HierarchyNodeType property.
2 Select Remove Value.

**Adding, Modifying, and Deleting Node Glyphs**

You can create new node glyphs and delete glyphs that you no longer want to use. You can also modify existing glyphs.

➤ To create a new node glyph:
1 Select Admin, then Node Type Glyphs.
2 Click 📥.
3 Enter a name for the new glyph.
4 Click Browse and select the bitmap that you want to use for the new glyph.

**Note:**
Glyph bitmaps must use a pixel size of 16 x 16.

➤ To modify an existing node glyph:
1 Select Admin, then **Node Type Glyphs**.
2 Double-click a glyph.
3 In the Edit Node Glyph dialog box, do any of the following actions:
   ● Edit the name of the glyph
   ● Click Browse and select a different bitmap

➤ To delete a node glyph:
1 Select Admin, then **Node Type Glyphs**.
2 Select the glyph that you want to delete and select .
3 Click OK to confirm deletion of the selected glyph.
Formula Derivers

In Data Relationship Management, formulas are achieved using a derived property class called *Formula Deriver*. Formulas are composed of string literals and functions—all with their parameters. The parameters can consist of string literals and other functions. Formulas enable the Data Relationship Management administrator to create custom properties.

Creating and Editing Formulas

System administrators use the Formula Tester dialog box.

➤ To create a formula property:

1. In the Properties dialog box, create a property with Property Type set to Derived or RWDerived and Class Name set to Formula.

2. Select the Property Params tab, select the Value column, and click ![...](image).

The Formula Tester dialog box is displayed.
3 Enter a formula by typing directly into the Formula box or by selecting from the lists:

- All existing functions are listed with their required parameters and their result types. To insert a function into the formula, select it from the list and click the insert function. Then replace each required parameter with the appropriate value.

- All currently defined properties (including system properties) are listed with their labels. To insert a property into the formula, select it from the list, and click the insert property.

**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. Because 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 flag an error, but too few parameters do.

Because the syntax validation does not evaluate the formula, 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.

**Formula Evaluation**

Formulas can be tested using the *Evaluate* button. The evaluation uses 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.

When evaluating a formula, there are two options for supplying test data:

- Evaluate using Node
- Evaluate using Scratch Pad

For the Evaluate using Node option, clicking displays the Select Nodes dialog box. After a node is selected, its current property values are used in the formula.

For the Evaluate using Scratch Pad option, property values can be entered manually. Values can also be copied from a node and then modified for the evaluation.

**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 rather the result of a function.

Consider these examples:

- The formula `PropValue(Concat(Abb,rev))` 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,Abbrev,Label))`.

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(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 **NodeType** property. Thus, the formula cannot be edited directly in the grid, but must be edited using *Formula Tester* and a user-supplied property value.

### Functions

This section discusses rules about using functions. For more information see:

- “Function Names” on page 94
- “Function Parameters” on page 94
- “Literals” on page 94
- “Function Definitions” on page 95

### Function Names

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

### Function Parameters

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.

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

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

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

### Literals

Any value that is not a valid function name followed by parentheses is considered a literal. A literal can be a string, integer, floating-point or boolean literal. In a string literal, spaces are treated a character. Therefore, extra spaces should not be used in formulas.
**Formula Evaluation**

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. Because implicit circular reference validating is not performed by the Formula Deriver, a circular reference causes a stack overflow and abnormally terminates the Data Relationship Management application.

**Function Definitions**

An alphabetical listing of available functions.

**Abbrev**

Description

Returns the Abbrev (name) of the current node.

Syntax

Abbrev(): String

Example

Current Node Name = 100 then Abbrev()

Return value is 100.

**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 Added By username.

Syntax

```
AddedBy(): String
```

**AddedOn**

Description

Returns the Added On Date/Time in the server's regional format converted from the internal format of yyyy/mm/dd hh:mm:ss am/pm.

Syntax

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

**AddFloat**

Description

Adds two specified float values and returns the result.

Syntax

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

Example

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

The return value is 5.89.

**And**

Description

Returns True if both Boolean expressions specified evaluate to True.

Syntax

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

Example

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

Return value is True
**ArrayCount**

**Description**
Returns the number of items in the list.

**Syntax**
ArrayCount(List: String; Delimiter: String): Integer

**Example**
ArrayCount(Diet Cola; Orange Cola; Root Beer; Cola,;)
Return value is 4

**ArrayIndex**

**Description**
Returns the position of the first occurrence of the specified item within the list. Returns 0 if the item is not found.

**Syntax**
ArrayIndex(Item: String; List: String; Delimiter: String): Integer

**Example**
ArrayIndex(Cola, Diet Cola; Orange Cola; Root Beer; Cola,;)
Return value is 4

**ArrayItem**

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

**Syntax**
ArrayItem(List: String; Delimiter: String; Index: Integer): String

**Example**
ArrayItem(Diet Cola; Orange Cola; Root Beer; Cola,; 4)
Return value is Cola
**BoolToStr**

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

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

**Example**
```
BoolToStr(1)
```
Return value is True

**ChangedBy**

**Description**
Returns the Changed By username.

**Syntax**
```
ChangedBy(): String
```

**ChangedOn**

**Description**
Returns the Changed On Date/Time in the server’s regional format converted from the internal format of yyyy/mm/dd hh:mm:ss am/pm.

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

**Concat**

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

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

**Example**
```
Concat(Abbrev, -, Descr())
```
If current node name is 100 and current node description is Colas, then return value is 100–Colas.

**ConcatWithDelimiter**

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

**Syntax**
ConcatWithDelimiter(Delimiter: String; SkipBlanks: Boolean; Item1, Item2, ... ItemN: String): String

**Example**
ConcatWithDelimiter(;,1,Item1,Item2,Item3,Item4)
Return value is Item1; Item2; Item3; Item4

**Descr**

**Description**
Returns the description of the current node.

**Syntax**
Descr(): String

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

**Divide**

**Description**
Divides two specified integer values and returns the result.

**Syntax**
Divide(Int1,Int2: Integer): Integer

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

**Description**
Divides the two specified floats and returns the result.

**Syntax**
```
Divide(Float1, Float 2: Float): Float
```

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

**Equals**

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

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

**Example**
```
Equals(integer, 01, 1)
Return value is True.
```

**FlipList**

**Description**
Returns a string representing the reverse of the specified list.

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

**Example**
```
FlipList(DietCola; Orange Soda; Root Beer; Lemonade, ;)
Return value is Lemonade; Root Beer; Orange Soda; Diet Cola.
```
**FloatToStr**

**Description**
Returns a string representing the specified floating-point value.

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

**Example**
```
FloatToStr(01.001)
```

Return value is 1.001.

**Format**

**Description**
Invokes Format function using a specified format string, param type-identifier and param value of the specified type. This function is limited to one value parameter. The format string used is described in the format string section. See “Format String Parameter” on page 60.

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

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

Return value is 123.46

**GreaterThan**

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

**Syntax**
```
GreaterThan(Value1, Value2: Integer(Default); [ParamType: String]): Boolean
```

**Example**
```
GreaterThan(1,2)
```

The return value is False.
**GreaterThanOrEqual**

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

**Syntax**
GreaterThanOrEqual(Value1, Value2: Integer(Default); [ParamType: String]): Boolean

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

**HasChildWith**

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

**Syntax**
HasChildWith(Expression1: Boolean): Boolean

**Example**
HasChildWith(GreaterThan(ID(),200))
If child IDs = 20, then return value is False.

**HasParentNode**

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

**Syntax**
HasParentNode(ALocalNodeAbbrev: String): Boolean

**Example**
HasParentNode()
If the parent ID is 200, then the return value is True.
**HasSiblingWith**

**Description**
Returns True if the specified expression is True for any sibling. Formula using this function must be in a local property.

**Syntax**
\[ \text{HasSiblingWith}(\text{Expression1}: \text{Boolean}): \text{Boolean} \]

**Example**
\[ \text{HasSiblingWith}(\text{PropValue}(\text{Leaf})) \]
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**
\[ \text{HierNodePropValue}(\text{HierAbbrev}, \text{NodeAbbrev}, \text{PropAbbrev}: \text{String}): \text{String} \]

**Example**
\[ \text{HierNodePropValue}(\text{Assets}, 1000, \text{Description}) \]
If the description for node 1000 in the Assets hierarchy is “Banking”, then the return value is Banking.

**ID**

**Description**
Returns the integer ID of the current node.

**Syntax**
\[ \text{ID}(): \text{Integer} \]

**Example**
\[ \text{ID}() \]
If the current node ID is 2000, then the return value is 200.
**If**

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

**Syntax**
\[ \text{If}(\text{Expression}: \text{Boolean}; \text{ResultIfTrue}, \text{ResultIfFalse}: \text{String}): \text{String} \]

**Example**
\[ \text{If}(\text{Equals}\left(\text{String},\text{Descr()},\right),\text{Abbrev()},\text{Concat(Abbrev,},-,\text{Descr()})) \]
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.

**Intersection**

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

**Syntax**
\[ \text{Intersection(List1}: \text{String}; \text{List2}: \text{String}; \text{Delimiter}: \text{String}) : \text{String} \]

**Example**
\[ \text{Intersection(A;}B;C;D;E,C;E;F;A,;)) \]
The return value is A;C;E.

**IntToStr**

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

**Syntax**
\[ \text{IntToStr(Int}: \text{Integer}) : \text{String} \]

**Example**
\[ \text{IntToStr(12345)} \]
The return value is 12345.
**IsAlpha**

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

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

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

**IsNumeric**

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

**Syntax**
IsNumeric(AString: String; [AllowBlanksAsNumeric]): String

**Example**
IsNumeric(12345)
The return value is True.

**IsRangeListSubset**

**Description**
Returns True if the subset exists in the Range List.

**Syntax**
IsRangeListSubset(RangeList, SubsetRangeList: RangeList, Delimiter: String): Boolean

**Example**
IsRangeListSubset(1–10,2–3;4–1,;)
The return value is True.
**Length**

**Description**

Returns the number of characters in the specified string.

**Syntax**

\[ \text{Length(AString: String): Integer} \]

**Example**

\[ \text{Length(Desc)} \]

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

**LessThan**

**Description**

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

**Syntax**

\[ \text{LessThan(Value1, Value2: Integer(Default); [ParamType: String]): Boolean} \]

**Example**

\[ \text{LessThan(1,2)} \]

The return value is True.

**LessThanOrEqual**

**Description**

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

**Syntax**

\[ \text{LessThanOrEqual(Value1, Value2: Integer(Default); [ParamType: String]): Boolean} \]

**Example**

\[ \text{LessThanOrEqual(3,3)} \]

The return value is True.
**ListAncestors**

**Description**
Returns a comma-delimited list of the current node’s ancestors starting from the top node. The current node must be a local node for this function to work. To ensure that the node is local, any formula containing this function must be part of a local derived formula property.

**Syntax**
ListAncestors(): String

**Example**
ListAncestors()

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

---

**ListChildren**

**Description**
Returns a comma-delimited list of the current node’s immediate children.

**Syntax**
ListChildren(): String

**Example**
ListChildren()

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

---

**ListContains**

**Description**
Returns a Boolean value indicating whether a value is contained in a delimited list.

**Syntax**
ListContains(List, Item, Delimiter: String): String

**Example**
ListContains(T-Bone;Rib-Eye;Sirlois;Filet,Filet,;)

The return value is true.
ListDescendants

Description
Returns a comma-delimited list of all of the current node’s descendants.

Syntax
ListDescendants(): String

Example
ListDescendants()

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

ListPeers

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

Syntax
ListPeers(): String

Example
ListPeers()

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

LowerCase

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

Syntax
LowerCase(AString: String): String

Example
LowerCase(HOBBES)

The return value is hobbes.
LTrim

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

Syntax
LTrim(AString: String): String

Example
LTrim("   101203   ")

The return value is 101203.

Modulus

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

Syntax
Modulus(Int1, Int2: Integer): Integer

Example
Modulus(5,2)

The return value is 1.

Multiply

Description
Multiplies two specified integers and returns the result.

Syntax
Multiply(Int1, Int2: Integer): Integer

Example
Multiple(2,5)

The return value is 10.
**MultiplyFloat**

**Description**
Multiplies two specified floats and returns the result.

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

**Example**
MultiplyFloat(4.76, 2.3)
The return value is 10.948.

**NextSibling**

**Description**
Returns the next sibling for the specified node.

**Syntax**
NextSibling(): String

**Example**
NextSibling()
If A, B, C, and D are children of Z and the current node is B, then the return value is C.

**NodeAccessGroups**

**Description**
Returns a comma-delimited list of node access groups for which the currently logged in user has rights.

**Syntax**
NodeAccessGroups(UserName: String): String

**Example**
NodeAccessGroup()
The return value is Accounts, Finance.
**NodeExists**

**Description**
Returns True if the specified node exists.

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

**Example**
`NodeExists(2000)`
If node 2000 exists, then the return value is True.

**NodeInHier**

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

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

**Example**
`NodeInHier(2000, Assets)`
If the node 2000 is in the Assets hierarchy, then the return value is True.

**NodeIsLeaf**

**Description**
Returns True if the current node is a leaf (that is, it can not contain children.)

**Syntax**
`NodeIsLeaf(): Boolean`

**Example**
`NodeIsLeaf()`
If the current node is a leaf, then the return value is True.
**NodePropValue**

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

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

Example
`NodePropValue(2000, Abbrev)`
Return value is 2000.

**Not**

Description
Returns the Boolean opposite of the specified Boolean expression.

Syntax
`Not(Expression: Boolean): Boolean`

Example
`Not(NodeIsLeaf)`
If the node is a limb, then the return value is True.

**Now**

Description
Returns the Current Date and/or time. The Optional DateTimeType Parameter can be [date, time, or datetime].

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

Example
`Now()`
Returns the current date and time, for example 3/25/2008 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 3/25/2008.

**NumChildWith**

**Description**

Returns the number of children where the specified expression is True.

**Syntax**

NumChildWith(Expression: Boolean): Integer

**Example**

NumChildWith(NodeIsLeaf())

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

**NumDescendantsWith**

**Description**

Returns the number of descendants where the specified expression is True.

**Syntax**

NumDescendantsWith(Expression: Boolean): Integer

**Example**

NumDescendantsWith(NodeIsLeaf())

If the node has two children and each child has 10 leaf children, then the return value is 12.

**Or**

**Description**

Returns True if any of the specified Boolean expressions are True.

**Syntax**

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

**Example**

Or(NodeIsLeaf(), PropValue(Leaf))

If the current node is a leaf and has property value leaf, then the return value is True.
**OrigPropValue**

Description

Returns the value of the specified property for the originating node of the function. When using a function such as HasChildWith, the originating node can be referenced inside that function by way of the ParentPropValue function. However, when using a function such as HasSiblingWith or NumDescedentsWith, the OrigPropValue function must be used.

Syntax

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

Example

```
HasChildWith(GreaterThan(OrigPropValue(ID), ID()))
```

If the current ID is 200, then the return value is True.

**PadChar**

Description

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

Syntax

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

Example

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

The return value is 000102.

**PadList**

Description

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

Syntax

```
PadList(String, DelimChar, PadChr: String, PadLeft: Boolean, NewLength: Integer): String
```
Example
PadList(1;2;3;4;5,;,0,1,3)
The return value is 001;002;003,004,005.

**ParentPropValue**

**Description**
Returns the value of the specified property of the current node’s parent node. The current node must be a local node for this function to work. To ensure that the node is local, any formula containing this function must be part of a local derived formula property.

**Syntax**
ParentPropValue(PropAbbrev: String): String

**Example**
ParentPropValue(Abbrev)
If the parent node name is Colas, then the return value is Colas.

**Pos**

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

**Syntax**
Pos(ASubstring, AString: String): Integer

**Example**
Pos(D, ABCDEFG)
The return value is 4.

**PreviousSibling**

**Description**
Returns the previous sibling for the current node.

**Syntax**
PreviousSibling(String): String
Example
PreviousSibling(String)
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 Assets.

**PropDefaultValue**

Description
Returns the default value of the specified property.

Syntax
PropDefaultValue(PropAbbrev: String): String

Example
PropDefaultValue(Currency)
The return value is USD.

**PropertyCategories**

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

Syntax
PropertyCategories(AccessType: String) :String

Example
PropertyCategories(Both)
The return value is System, All, Essbase, Enterprise, HFM, Planning.

**PropMaxValue**

Description
Returns the maximum value of the specified property.

Syntax
PropMaxValue(PropAbbrev: String): String

Example
PropMaxValue(Volume)
The return value is 10.

**PropMinValue**

Description
Returns the minimum value of the specified property.

Syntax
PropMinValue(PropAbbrev: String): String

Example
PropMinValue(Volume)
The return value is 1.

**PropValue**

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

Syntax
PropValue(PropAbbrev: String): String

Example
PropValue(Volume)
The return value is 2.
**RangeListContains**

**Description**

Returns True if the Range List contains the specified value.

**Syntax**

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

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

**Example**

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

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

**Note:**

If you change MyRangeList to "1-5,6-10,101-1000", Data Relationship Management replaces this value with "1-10,101-1000" because it checks the validity of the RangeList and combines ranges with contiguous boundaries.

**ReplacementAbbrev**

**Description**

If the current node is inactive, this function returns the Abbrev (name) of the current node’s replacement node.

**Syntax**

`ReplacementAbbrev(): String`

**Example**

`ReplacementAbbrev()`

**ReplacePropValue**

**Description**

If the current node is inactive, this function returns the value of the specified property of the current node’s replacement node.

**Syntax**

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

**ReplaceStr**

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

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

Example
ReplaceStr(ABACADAEAFAG, A, 2, 1)
The return value is 2B2C2D2E2F2G.

**RTrim**

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

Syntax
RTrim(AString: String): String

Example
RTrim("   100   ")
The return value is “100”.

**StripPadChar**

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

Syntax
StripPadChar(AString, PadChar: String; StripCount: Integer): String
Example

StripPadChar(0003333,0,6)

The return value is 3333.

**StrToBool**

Description

Returns a Boolean value based on the specified string.

- If the string starts with a Y, T, or 1 (one), regardless of case or following characters, the function returns True.
- If the string starts with N, F, or 0 (zero), regardless of case or following characters, the function returns False.
- If the string starts with any other character or is empty, the default value is returned.

Syntax

StrToBool(AString: String; Default: Boolean): Boolean

Example

StrToBool(0)

The return value is False.

**StrToFloat**

Description

Returns the floating-point value of the specified string. If the string does not represent a valid floating-point value, the default is returned.

Syntax

StrToFloat(AString: String; Default: Float): Float

Example

StrToFloat(11.101)

The return value is 11.101.

**StrToInt**

Description

Returns the integer value of the specified string. If the string does not represent a valid integer value, the default is returned.
Syntax
StrToInt(AString: String; Default: Integer): Integer

Example
StrToInt(101)

The return value is 101.

Stuff

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

Syntax
Stuff(APropertyName, ACharsToReplace, AReplacement: string): String

Example
Stuff(Abbrev,GEO,RIO)

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

SubString

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

Syntax
SubString(AString: String; Index, Count: Integer): Integer

Example
SubString(Colas,1,2)

The return value is Co.

Subtract

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

Syntax
Subtract(Minuend,Subtrahend: Integer): Integer
Example
Subtract(10,2)
The return value is 8.

**SubtractFloat**

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

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

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

**Trim**

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

**Syntax**
Trim(AString: String): String

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

**UpperCase**

**Description**
Returns the specified string converted to all upper case.

**Syntax**
UpperCase(AString: String): String

**Example**
UpperCase(smaller)
The return value is SMALLER.
**UserName**

**Description**

Returns the user name for the currently logged in user.

**Syntax**

```
UserName(): String
```

**Example**

```
UserName()
```

**XOr**

**Description**

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

**Syntax**

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

**Example**

```
XOr(NodeIsLeaf(), PropValue(Leaf))
```

If the node is a leaf, then the return value is False.
Transaction requests are proposed data changes to a Data Relationship Management version. Requests consist of multiple transactions that are validated against and committed to a version as a group. Transaction requests greatly extend the ability to integrate approval change management with Data Relationship Management by allowing transactions within a request to be grouped as a single entity and allow the request to be validated without committing changes to the Data Relationship Management version.

You can create transaction requests to:

- Add and delete hierarchies
- Add and delete nodes
- Insert and move nodes
- Activate, inactivate and remove nodes
- Update properties
- Remove property values

Transaction requests can be stored in a draft state for approval and validated against a Data Relationship Management version without committing the changes to the version immediately. Transaction requests in this pending approval state can be updated by multiple users at different times and re-validated as needed. The transactions in a request are committed to a Data Relationship Management version when the request is approved.

**Note:**

After a transaction request has been approved, the request cannot be modified and the request cannot be deleted until the associated version is deleted.

A transaction request consists of the following elements:

- Target Data Relationship Management version
- Owner of the request — consists of a valid Data Relationship Management user ID
- Custom workflow ID — identifier for the request in a workflow application
Custom workflow label — short description for the request in a workflow application

Custom workflow status — manages the status of the request in a workflow application

Custom workflow info — stores extra information needed by a workflow application

Request comments — annotation for the request

Additional verifications that should be applied to only the actions in the request during a validate or approve operation

List of action items that affect hierarchies and nodes for the current request

Transaction requests are exposed via the Data Relationship Management API and can be leveraged by third-party applications to support request and approval change management processes. For more information on requests, see the Oracle Hyperion Data Relationship Management N-tier Application Programming Interface Guide.

Viewing Request History

All transaction requests are logged in the Request History. This data can be viewed by selecting Tools, then Request History from the Data Relationship Management main menu. Each request is recorded with timestamps, the user ID of the request owner, the actions being performed, the validation and approval status, and other request level information. When you first access the Request History screen, a default query displays all unapproved requests that you own.

The top section of the Request History screen displays the list of transaction requests that meet the current query criteria. The bottom section allows you to define query criteria. Query criteria are grouped into four categories:

- Summary
  - Request ID — cannot be changed; automatically generated when the request is created.
  - Owned By user ID — cannot be changed; defined when request is created.
Committed By user ID — cannot be changed; defined when request is approved.

Target version for the request.

Perform exact match on string values option.

Maximum number of records to return setting.

- **Workflow** — all elements are defined by the workflow application
  - Custom ID
  - Custom Label
  - Custom Status
  - Comments

- **History**
  - Created By, Updated By, and Approved By fields for managing user-related information for the workflow application.
  - Create Date, Update Date, and Approval Dates for when a request is created, updated, and approved.

- **Detail**
  - A hierarchy in the selected target version.
  - A node in the selected target version.

## Specifying Request History Range and Filters

The requests displayed can be restricted to a desired range by specifying either:

- Create Date/Time
- Update Date/Time
- Approve Date/Time

After setting the desired range, click the Refresh button to display requests within the range.

The data can also be filtered on other columns including:

- Request ID
- Version
- Owned By
- Committed By
- Workflow ID
- Workflow Label
- Workflow Status
- Validated
- Approved
- Hierarchy
The Max Records value can be set to limit the number of transactions displayed.

**Note:**
For unlimited records, enter zero for the value.

**Viewing Transaction Request Details**
To view details of a transaction request, double-click a request in the list.

**Deleting Transaction Requests**
To delete a transaction request, select the request and click Delete.

An approved request can only be deleted when the version it was approved for is deleted from the system. HDRM automatically deletes approved requests for a version when the version is deleted. To delete a version and all of its approved requests, right-click a version in the Version Control window and select Delete.

**System Preferences for Transaction Requests**
The following System Preferences work with transaction requests:

- AllowRequestHistoryQuery — Security level required for querying or viewing a request via the Request History screen (Admin, Funct, Any, None).
- AllowRequestHistoryDelete — Security level required for deleting a Request via the Request History screen (Admin, Funct, Any, None).
In This Chapter

- Configuration Settings ................................................................. 130
- Extracting Metadata ................................................................. 131
- Loading Metadata ................................................................. 132
- Comparing Metadata ................................................................. 133
- Viewing Metadata ................................................................. 134
- Generating Reports ................................................................. 135

The Data Relationship Management Migration Utility provides system administrators the ability to move certain metadata components between Data Relationship Management instances.

You access the Migration Utility from a URL such as http://Webservername/mdm_migration/Default.aspx where Webservername is the name of the Web server computer.

Note:

The system administrator must be a System Admin user type for all Data Relationship Management instances that are to be accessed by the Migration Utility.

You can use the migration utility to:

- Migrate metadata changes from Development/Test/QE to production Data Relationship Management instances.
- Backup and compare metadata changes (historically or between instances.)
- Update list properties with large lists in an automated way.
- Provide support-specific metadata for issues without having to send an entire database.

There are four features of the Migration Utility:

- Extract metadata components from a Data Relationship Management instance to an XML file and generate an HTML report from the results
- Load metadata from an XML file into a Data Relationship Management instance
- 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
• System Categories
• Queries (Standard only)
• Compares (Standard only)
• Exports (Standard only)
• Imports
• Blenders
• System Preferences

Configuration Settings

The following table describes Migration Utility configuration settings in the appSettings section of the web.config file.

<table>
<thead>
<tr>
<th>Table 15 Configuration Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key</strong></td>
</tr>
<tr>
<td>configuredServers</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>maximumExceptionsOnImport</td>
</tr>
<tr>
<td>useEncryptedPasswords</td>
</tr>
<tr>
<td>showExceptionDetail</td>
</tr>
<tr>
<td>Key</td>
</tr>
<tr>
<td>------------------------</td>
</tr>
<tr>
<td>Caution!</td>
</tr>
</tbody>
</table>

**enableAboutPage**

Specifies whether the About page is enabled. The About page displays the version of the Migration Utility and system components; for greater security, this page is disabled by default. To check the version of the Migration Utility you can enable this page.

To enable the page but restrict access to administrators, edit the Discretionary Access Control List (DACL) on the /Forms/About.aspx file. See the IIS documentation for more information about how DACLs, Directory Security, and anonymous access interact to control access to webpages. Specify True to show the About page, otherwise False. The default value is False.

---

**Extracting Metadata**

You can select the types of metadata to extract from a Data Relationship Management instance. You extract the information into an XML file which you can then view, load into another Data Relationship Management instance, compare to another XML file, or compare to another Data Relationship Management instance. 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 instance:

1. **On the Main Menu, click Extract or press Alt+M.**
2. **Enter Data Relationship Management connection information and click Log In or press Alt+N.**
3. **Select the components or sub-components to extract and click Next.**

   **Note:**

   Click the plus sign in the hierarchy tree to see sub-components. Select the checkbox for a component to select the component and all of its sub-components, or select the checkbox for the sub-component that you want to extract. Click on a sub-component name to display the component definition in a new window.

4. **Optional: Click Find or press Alt+I to search for a metadata component or sub-component.**

   **Note:**

   Any component 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 components 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, include all excluded dependencies, or exclude all dependencies.

Note:
Increasing the page size allows you to define the number of components 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 or press Alt+N.

8 Do any of the following:
- Click Download the Metadata File or press Alt+W — to open or save the XML file.
- Click View the Metadata File or press Alt+I — to view the XML file details.
- Click Load the Metadata File or press Alt+L — to load the XML file into a Data Relationship Management instance. For more information, see “Loading Metadata” on page 132.
- Click Generate Reports for the Metadata File or press Alt+P — to generate a report from the XML file. For more information, see “Generating Reports” on page 135.

Loading Metadata

Only files with the Data Relationship Management XML format can be loaded into a Data Relationship Management instance. A log file is created after a load is performed and displays the following severities of data: audit, information, warning, and error message.

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

1 On the Main Menu, click Load or press Alt+L.
2 Click Browse, select the XML file that you want to load, and click Upload or press Alt+U.
3 Review the uploaded file information and click Next or press Alt+N.
4 Enter Data Relationship Management connection information and click Log In or press Alt+N.
5 Select the components or sub-components to load and click Next or press Alt+N.
Review the summary information and click Next or press Alt+N.

Page size allows you to define the number of components to view on a page.

Optional: Select Continue Load After Error for the load to continue even if errors are encountered.

Click Run Load or press Alt+N.

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.

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 instances, between two XML files, or between a Data Relationship Management instance 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 component 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 or press Alt+I.
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 or press Alt+N.
   - If you selected XML File, click Browse and select the XML file that you want to use in the comparison and click Upload or press Alt+U.
4. If you uploaded a file, review the uploaded file information and click Next or press Alt+N. Otherwise, skip to the next step.
5. Repeat steps 2–4 for Source #2.
6. Click Next or press Alt+N.
Select the components to include in a difference file by using the following actions:

- Select an appropriate filter from the Filter drop-down list.
- Click > to select a component from Source #1.
- Click < to select a component from Source #2.
- Click X to deselect a component.
- 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 components to view on a page.

Click **Create Difference File** or press Alt+N.

Do any of the following:

- Click **Download the Metadata Difference File** or press Alt+W — to open or save the XML file.
- Click **View the Metadata Difference File** or press Alt+I — to view the XML file details.
- Click **Load the Metadata Difference File** or press Alt+L — to load the file into an Oracle Hyperion Data Relationship Management, Fusion Edition instance. For more information, see “Loading Metadata” on page 132.
- Click **Generate Reports for the Metadata File** or press Alt+P — to generate a report from the XML file. For more information, see “Generating Reports” on page 135.

**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 or press Alt+W.**
2. **Click Browse and select the XML file that you want to view and click Upload or press Alt+U.**
3. **Review the uploaded file information and click Next or press Alt+N.**
4. **Click the plus signs in the hierarchy tree to view metadata sub-components.**
5. **Optional: Click Find or press Alt+I to search for an item in the file.**

**Note:**

Any component containing the text is returned. To navigate to a particular object in the results, click the Jump To link.
Optional: Click the Reports tab or press Alt+P 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 or press Alt+P.
   - After viewing a metadata file, click Reports or press Alt+P.

2. Do one of the following:
   - Click View Report to display the report.
   - Click Download Report to save the report.
Index

A
adding node glyphs, 88
authentication method, 46
Automator
   defined, 16

B
Blender
   defined, 16

categories tab, 68
Compare, 81
compare
   defined, 17
   property, 18
   renamed node, 18
   types, 18
configuring system preferences, 27
controlling hierarchy, 15
controlling hierarchy tab, 64
creating validations and verifications, 77

defined, 16
defining users
   authentication method, 46
deleting node glyphs, 88
export
   defined, 17
Exports, 81
external authentication, 46

F
formula evaluation, 93
formula functions, 94
formula syntax checks, 92
formulas
   creating and editing, 91
derivers, 91
Functional Admin overview, 43
functions, 94

G
global node, 14

H
hierarchy
   defined, 13

I
import, 16
inheritance
   defined, 15

L
list values tab, 65
local node, 14
lookup table tab, 66

N
new user dialog box, 47
node
   defined, 13
node access groups
   defining, 51
   levels, 50
   security, 52

Index 137
node glyphs
  adding and deleting, 88
node types
  configuring, 87

P
properties
  categories, 69
  data types, 57
properties dialog box, 55
property
  defined, 14
property categories, 69
    dialog box, 69
property compare, 18
property data types, 57
property definition tab, 56
property params tab, 67

R
referential integrity
  defined, 14
renamed node compare, 18

S
Security Admin overview, 44
System Admin overview, 43
system architecture, 11
system categories dialog box, 81
system preferences
  configuring, 27

U
user definitions, 46
user management
  overview, 41
    user management dialog box, 45

V
validation and verification levels, 73
validation classes, 73
validations, 71
  assigning, 78
  creating, 77
validations dialog box, 72

parameters tab, 76
validation definition tab, 72
verifications, 71
  assigning, 78
  creating, 77
version
  defined, 12
  status, 13
Version Status, 81