

Application Tools Guide

9.4.1 Release

Copyright © 2026 OneStream Software LLC. All rights reserved.

All trademarks, logos, and brand names used on this website are the property of their respective owners. This document and its contents are the exclusive property of OneStream Software LLC and are protected under international intellectual property laws. Any reproduction, modification, distribution or public display of this documentation, in whole or part, without written prior consent from OneStream Software LLC is strictly prohibited.

Table of Contents

Application Tools 1

Application Security Roles 2

Application-Level User Interface Roles 5

Application Properties 9

General Properties 9

Global Point of View 9

Company Information 9

Workflow Channels 10

Formatting 10

Currencies 11

Transformation 11

Certification 12

Dimension Properties 12

Time Dimension 12

User Defined Dimensions (Descriptions) 12
Standard Report Properties 13
Logo 13
Title 13
Header Labels 14
Header Bar 14
Footer 14
Business Rules 16
Ways to Write Calculation Logic 16
Business Rules 16
Complex Expression 16
Member Formula 17
Business Rule Utility Groups 17
BRApi 17
API 17
ARGS 17

Business Rule Encryption and Decryption 18
Encrypt a Business Rule 18
Decrypt a Business Rule 19
Business Rule Search 19
Business Rule Toolbar and Hot Keys 21
Business Rule Client Image Types 22
Business Rule Properties 22
General 22
Security 24
Business Rules Editor Overview 24
Business Rule Types 25
Finance Rule 25
Parser Rule 29
Connector Rule 30
Conditional Rule 34
Derivative Rule 36

Common Usage 36

Derivative Business Rule Example 37

Cube View Extender 38

Common Usage 38

Dashboard DataSet 39

Common Usage 39

Dashboard Extender 40

Common Usage 40

Dashboard XFBRString 41

Extensibility Rule 41

Extensibility Business Rule Example 42

Extender 47

Available Operations 47

Journals Event Handler 49

Save Data Event Handler 49

Forms Event Handler 49

Data Quality Event Handler 50
Data Management Event Handler 51
Workflow Event Handler 51
Business Rules 52
Platform Engines 53
Formulas 56
Formula Structure 57
Microsoft Visual Basic .NET with OneStream API and Member Scripts 57
Common Finance APIs that Use Member Scripts 57
Basic Dynamic Cell Calculation 58
Basic Stored Calculation 58
Applying Formulas 60
Member Formulas 60
Finance Business Rules 61
Formula Types 61

Performance Considerations 62
Reference and Aggregation Considerations 62
Formula Execution 63
Cube Processing 64
Data Units and Formula Process 65
Data Unit Calculation Sequences (DUCS) 66
Formula for Statutory Consolidation Sequence 67
Formula Level Controls 67
Writing Formulas 67
Writing Dynamic Cell Calculations 68
Dynamic Calc Formula Examples 68
Return Types for Dynamic Calc Formulas 69
Using Math in Dynamic Calc Formulas 70
Division in Dynamic Calc Formulas 71
Days Sales Outstanding 73

Dynamic Simulation of Consolidation: GetEntityAggregationDataCell 74

Create a Cube View 76

Dynamic Calc Member Formula 76

Annotations on Dynamic Calc Members 77

Writing Stored Calculations 77

Basic Stored Formula Examples 78

Copy Data from another Account 78

Copy Data from Another Scenario 80

Out Of Balance 81

CTA Account Formula Example 82

Reading a Specific Data Cell in a Stored Formula 82

Advanced Stored Formulas using Eval 83

Advanced Stored Formulas Using GetDataBuffer and SetDataBuffer 86

Comparing Two DataBuffers Using Eval2 87

Referencing a Business Rule from a Member Formula or Business Rule 89

Conditionally Apply Formulas to Entities and Consolidation Members 90

Focus Consolidation to Calculate Only When Needed 91

Parent-Child Relationships 91

At Specific Level of Consolidation 92

Formulas for Calculation Drill Down 92

Global Entity for Driver Storage 93

Avoiding Data Explosion 93

No Explosion 93

Some Explosion 94

Large-scale Explosion when Setting to a Constant 95

Writing Formulas to Avoid Data Explosion 95

Avoiding Data Explosion in Stored Formulas 96

Key Functions 99
Account 99
Consolidation 101
Data 105
Entity 114
Flow 115
FX 116
Journals 119
Member 120
Scenario 123
Status 124
Time 125
Functions 125
Examples of Key Functions in Use 127
Calculate Data 127
Clear Calculated Data 128

DataBuffer 128

GetDataBufferUsingFormula 130

Remove Functions in Formulas 132

Dynamic Calc Using GetDataCell 135

Dynamic Filters and Aliases 135

Math Functions 135

Member List Functions 136

POV Object 137

Retrieve Time Varying Property of Relationship's Percent Consolidated Entity Attribute 138

SetDataCell 138

Translate 139

Unbalanced Math 139

Finance Function Types 141

Create Two Accounts 147

BRAPI Rule Functions 149
Spreadsheet or Excel Functions 149
Member Filters 151
Member Filter Builder 152
Member Script Abbreviations and Examples 154
Member Script Constants 156
Member Expansion Functions 157
Member Expansions 157
Reverse Order Member Expansions 172
Time Member Expansions 175
Workflow Member Expansions 177
Member Expansion Where Clause 182
Time Functions 187
Time Functions and Pov Member References 187
Time Functions and Workflow References 191
Time Functions and Global References 192

Time Functions and General References 194
Substitution Variables 203
Substitution Variables from Current POV 203
Workflow Substitution Variables 206
Global Substitution Variables 210
Cube View Substitution Variables 210
General Substitution Variables 212
Member Filter Substitution Variables 214
Custom Substitution Variables 216
Samples of Member Filter Functions 216
Samples of Member Expansions 217
Samples of Where Clause Expressions 217
Samples of Time Expressions 217
Samples of GetaDataCell Expressions 217
Samples of Column and Row Expressions 222
Samples of Custom Member List Expressions 223

Samples of XFBR, XFCell, and XFMember Property 227

Parameter and Param Display 231

Client Updater 232

Data Management 234

Rename, Copy, and Paste 235

Search 235

Sequences 236

General (Sequence) 236

Notification 236

Task Activity 238

Parameter Substitutions 1-8 239

Exposing Data Management Automation through OneStream Web API 240

OneStream Web API endpoints 240

Steps 242

Calculate 242

Execute Scenario Dynamic Source Data Copy 244
Clear Data 244
Copy Data 246
Custom Calculate 248
Execute Business Rule 251
Export Data 251
Export File 254
Export Report 256
Reset Scenario 257
Run a Sequence or Step 258
Load/Extract Application Artifacts 259
File Types 259
Extracting and Loading Dimensions 263
Extracting Dimension Changes 265
Loading the XML File 271
Common Load Errors and Resolutions 272

Project Extract and Load 274
Project File 274
File Extract 276
Zip Extract 278
File Load .xfProj 279
Zip Load 281
Spreadsheet 282
Microsoft Excel 2016 Chart Types 285
Chart Types 285
Retrieve Functions 291
Open and Save As Functionality 292
Reduce Cell Text Size 294
OneStream Ribbon 295
File Operations 296
Explore 296
General 296

Task Scheduler 298
Manage Tasks 299
Edit a Task 307
Task Scheduler Details 311
Task Manager Security Roles 313
Logon Activity 315
Calendar View 316
Load and Extract 320
Text Editor 323
Text Editor Ribbon 324
File 324
Home 325
Insert 327
Design 328
Layout 329
Format 332

Page Layout 336
References 337
View 340
Review 341
OneStream 346
Text Editor Watermarks 347
Create an Image Watermark 347
Create a Text Watermark 348
Modify a Watermark 348
Remove a Watermark 349

Application Tools

Use Application Tools to manage how you interact with the OneStream application and configure it to meet your needs. These tools enable you to define security roles, set application properties, and schedule data management sequences. This guide explains how to access and use the available tools to effectively manage your application environment.

Application Security Roles

The following Application security roles define what users can access or manage within the application:

Administer Application: Enables users to administer the application and load .zip files in environments with multiple applications managed by different administrator groups.

Administer Application Database: Enables a small group of users to mass delete metadata and data, primarily from the Database page. This role type is unlike most other roletypes because administrators are not automatically granted access to operations that require this role.

Administer Application Workspace Assemblie**s**: Enables users to manage Application Workspace assemblies associated with the application.

AnalyticsApi: Enables users to access OneStream connectors to pull data into external systems and control who can use the connectors. The role becomes available when access to the Power BI Connector is granted and may be available for use with additional connectors in the future.

Open Application: Enables users to view and open the application.

Modify Data: Enables users to modify data. Without this role, user access remains read-only throughout the application.

View All Data: Enables users to view all data in the application.

Create Audit Attachments: Enables users to create data attachments for supporting documentation.

Create Footnote Attachments: Enables users to create footnote attachments for supporting documentation.

Certify and Lock Descendants: Enables users to certify and lock descendants from the Workflow. This is typically an administrator function.

Unlock and Uncertify Ancestors: Enables users to unlock and uncertify ancestors from the Workflow. This is typically an administrator function.

Preserve Import Data: Enables administrators to lock a workflow and retain its imported data. The workflow can be unlocked later if needed.

Restore Import Data: Enables administrators to restore imported data to its original state from the Preserve Import process.

Unlock Workflow Unit: Enables users to unlock a Workflow Unit. Users must also have Workflow Execution access to lock a Workflow Unit.

View Source Data Audit: Enables users to view the Source Data Audit Report within the Import Workflow.

Encrypt Business Rules: Enables users to encrypt and decrypt a rule from the Business Rule screen in the Application tab.

Manage Application Properties: Enables users to update the application properties.

Manage Metadata: Enables users to edit metadata under the Dimension Library for the application.

Manage FX Rates: Enables users to update foreign exchange (FX) rates.

Lock FX Rates: Enables users to lock foreign exchange (FX) rates to prevent modification.

Unlock FX Rates: Enables users to unlock foreign exchange (FX) rates.

Manage Data: Enables users to manage data in all aspects, including exporting and clearing data through Data Management. This is typically an administrator function.

Manage Cube Views: Enables users to create new Cube Views and manage Cube View Groups and Profiles.

Manage Data Sources: Enables users to create new Data Sources.

Manage Transformation Rules: Enables users to create new transformation rules and manage transformation rule groups and profiles.

Manage Confirmation Rules: Enables users to create new confirmation rules and manage confirmation rule groups and profiles.

Manage Certification Questions: Enables users to create new certification questions and manage certification question groups and profiles.

Manage Workflow Channels: Enables users to create new Workflow Channels.

Manage Workflow Profiles: Enables users to create new Workflow Profiles.

Manage Journal Templates: Enables users to create new journal templates and manage journal groups and profiles.

Manage Task Scheduler: This role enables users to create their own tasks, view tasks created by all users, edit their own tasks and other users' tasks, and delete or turn off their own tasks. Users can also load and extract. Users cannot change the task name.

Task Scheduler: This role enables users to create their own tasks, view tasks created by all users, edit their own tasks and other users' tasks, and delete or turn off their own tasks. Users can also load and extract. Users cannot change the task name.

Task Schedule Page: This role is required to view the Task Scheduler page and all user tasks.

Manage Form Templates: This role enables users to create new form templates and manage form groups and profiles.

Manage Application Workspaces: Enables users to create new application Workspaces and manage Workspace groups and profiles.

Manage Application Database Files: Applies to the two file systems stored in the Framework database (System database) and each Application database. Users in the ManageSystemDatabaseFiles and ManageApplicationDatabaseFiles roles have full read and write access to their user folders in these systems. These folders are private to the user, and access is restricted to the user and managers. Security cannot be edited for a user folder. Users can be given read or write access to specific folders in the database file systems using the individual folder's security settings, excluding user folders and subfolders.

Manager Smart Links: Enables users to manage Smart Links within the application.

Manager Task Scheduler: Enables you to create your own tasks, view tasks created by all users, edit your own tasks and other users' tasks, and delete or turn off your own tasks. You can load and extract. You cannot change the task name.

Task Scheduler: Enables you to create new tasks, edit your tasks, and validate tasks if the setting is on. You can view all user tasks, but you can only edit your own tasks. You will not have access to load and extract. You cannot change the task name.

TaskSchedulerPage: Required to view the Task Scheduler page and view all user tasks.

Application-Level User Interface Roles

The following Application user interface roles define which pages users can access within the application:

Application Load Extract Page: Grants access to the Load/Extract screen located in Application >Tools. This is typically restricted to administrators.

Application Properties Page: Grants access to the Application Properties screen located in Application >Tools. This is typically restricted to administrators.

Application Security Roles Page: Grants access to the Application Security screen located in Application > Tools. This is typically restricted to administrators.

OnePlace Pane: Grants access to the OnePlace pane within the application user interface.

Book Admin Page: Grants access to the Book Designer screen located in Application > Presentation. This is typically restricted to administrators or users who create Report Books.

Business Rules Page: Grants access to the Business Rules screen located in Application > Tools. This is typically restricted to administrators.

Certification Questions Page: Grants access to the Certification Questions screen located in Application > Workflow. This is typically restricted to administrators.

Client Updater Page: Grants access to the Client Updater screen.

Confirmation Rules Page: Grants access to the Confirmation Rules screen located in Application > Workflow. This is typically restricted to administrators.

Cube Admin Page: Grants access to the Cube Admin screen located in Application > Cube. This is typically restricted to administrators.

Cube Views Page: Grants access to the Cube Views screen located in Application > Presentation. This is typically restricted to administrators or users who create Cube Views.

Data Management Admin Page: Grants access to the Data Management Admin screen located in Application > Tools. This is typically restricted to administrators.

Task Scheduler Page: Grants access to the Task Scheduler screen.

Data Sources Page: Grants access to the Data Sources screen located in Application > Data Collection. This is typically restricted to administrators.

Dimension Library Page: Grants access to the Dimension Library screen located in Application > Cube. This is typically restricted to administrators.

FX Rates Page: Grants access to the FX Rates screen located in Application > Cube. This is typically restricted to administrators.

Form Templates Page: Grants access to the Form Templates screen located in Application > Data Collection. This is typically restricted to administrators.

Journal Templates Page: Grants access to the Journal Templates screen located in Application > Data Collection. This is typically restricted to administrators.

Spreadsheet Page: Grants access to spreadsheet-based pages within the application.

Text Editor: Grants access to the text editor.

Time Dim Profiles Page: Grants access to the Time Dimension Profiles screen.

Transformation Rules Page: Grants access to the Transformation Rules screen located in Application > Data Collection. This is typically restricted to administrators.

Workflow Channels Page: Grants access to the Workflow Channels screen located in Application > Workflow. This is typically restricted to administrators.

Workflow Profiles Page: Grants access to the Workflow Profiles screen located in Application > Workflow. This is typically restricted to administrators.

> **NOTE:** Select the ellipsis icon and begin entering the Security Group name in the blank field. As the first few letters are entered, the Groups are filtered making it easier to find and select the desired group. Once the group is selected, select **Ctrl** and double-click to enter the correct name in the field.

Selecting Security Groups: Begin entering the Security Group name in the blank field to filter results. After the desired group displays, select **Ctrl**and double-click to enter the name into the field.

Application Properties

This is where default properties are set for the application and for properties that differ by scenario type:

- General Properties
- Dimension Properties
- Standard Report Properties

General Properties

Global Point of View

These properties are enabled when forcing Global Scenario and Global Time through Transformation settings. An initial value should be configured even if the Transformation setting is not being used.

Global Scenario: The default scenario users will view in Workflow.

Global Time: The default time users will view in Workflow.

Company Information

Company Name: Enter the company name to display it on automatically generated reports from Cube Views.

Logo File (PNG, ~50 pixels high): Attach a .png logo file that is approximately 50 pixels high to display it on Cube Views and reports.

Workflow Channels

UD Dimension Type for Workflow Channels: The origin dimension controls data load, but in some cases, other User Defined dimensions require their own locking layer. For example, if a company plans by entity and product, one entity may have multiple products handled by different people. Each channel can be locked separately to protect that layer of data instead of locking the entire entity.

Formatting

Number Format: Controls how numeric values display throughout the application and enables you to configure additional precision to the right of the decimal point. This setting can be overridden through Cube View formatting.

N0: Displays no decimals or trailing zeros.

N1-N6: Displays the specified number of decimals.

> **Example:** If N2 is chosen, two decimals are displayed, N5 will display five decimals.

> **NOTE:** The N in the above settings indicates that these settings are international.

For example, the following formats use 10000.001.

#,###,0\%: Returns 10,000% and -10,000%

#,###,0: Returns 10,000 and -10,000

The three sections in a number format, separated by semicolons, represent the format for positive numbers, negative numbers, and zeros.

#,###,0;(#,###,0);0: Returns 10,000 and (10,000)

#,###,0.00%: Returns 10,000.00% and -10,000.00%

#,###,0.00: Returns 10,000.00 and -10,000.00

#,###,0.00;(#,###,0.00);0.00: Returns 10,000.00 and (10,000.00)

> **NOTE:** To vertically align positive and negative numbers in reports where parentheses are used for negative values, include trailing spaces in the positive number format. This will account for the trailing parenthesis used by negative numbers.
> Example: #,###,0.00 ;(#,###,0.00);0.00

> **NOTE:** Select the ellipsis icon, select the desired format, then select **Ctrl** and double-click to enter the correct format in the field.

Currencies

Currency Filter: All currencies used in the application must be listed here to assign them to entities, perform currency translation, or enter rates. The list includes pre-euro and phased out currencies for historical data loading purposes. Contact OneStream Support if a currency is not listed and is needed for the application.

Transformation

Enforce Global POV: If set to True, the Global Scenario and Global Time settings are enforced, and users cannot change their workflow view. If the Global POV is enforced,  will display on the Import task during the workflow.

Allow Loads before Workflow View Year: If set to True, the current Workflow View setting is enforced to allow data loading to time periods before the current workflow year. If set to False, data cannot be imported to time periods before the current workflow year and  will display on the Import task during the workflow.

Allow Loads after Workflow View Year: If set to True, the current Workflow setting is enforced to allow data loading to time periods after the current workflow year. If set to False, data cannot be imported to time periods after the current workflow year and  will display on the Import task during the workflow.

Certification

Lock After Certify: If set to True, the workflow automatically locks after certification.

Dimension Properties

Time Dimension

Start Year: The starting year of the application.

End Year: The ending year of the application.

User Defined Dimensions (Descriptions)

Applies a custom description to User Defined dimensions. The description applies to the dimension type, not each individual dimension. The description is visible to users in OnePlace Application interfaces. The descriptions display in the tooltips, modal windows, and other dimension interfaces. These are viewable in the following locations:

- Point of View
- Dimension Library
- Cube View Member Filters
- Drill Down Dimension headers
- Excel Add-in / Spreadsheet
- Journals

UD1-UD8 Description: Enter a generic description that represents the purpose of each User Defined dimension.

Standard Report Properties

These settings will be applied when automatically generating a report from a Cube View.

Logo

Height: Enter a numerical value for the height of the report in pixels, for example, 105.

Bottom Margin: Enter a numerical value for the bottom margin.

Title

Top Margin: Enter a numerical value for the top margin.

Font Family: The font displayed in the title of the report.

Font Size: Enter a numerical value for the font size.

Bold: If set to True, the title will be bold in the report.

Italic: If set to True, the title will be italic in the report.

Text Color: Select the ellipsis icon to choose a text color for the report title.

Header Labels

Define the default Header Label properties for all the reports in the application.

Top/Bottom Margin: Enter a numerical value for the top/bottom margin.

Font Family: The font displayed in the header labels of the report.

Font Size: Enter a numerical value for the font size.

Bold: If set to True, the header labels will be bold in the report.

Italic: If set to True, the header labels will be italic in the report.

Text Color: Select the ellipsis icon to choose a text color for the header labels.

Header Bar

Background Color: Select the ellipsis icon to choose a header background color.

Line Color: Select the ellipsis icon to choose a header line color.

Footer

Text: A field to enter footer text.

Font Family: The font displayed in the footer.

Font Size: Enter a numerical value for the font size in the footer.

Show Line: If set to True, the report will display a line in the footer.

Show Date: If set to True, the report will display the date in the footer.

Show Page Numbers: If set to True, the report will display page numbers in the footer.

Line Color: Select the ellipsis icon to choose the color of the line in the footer.

Text Color: Select the ellipsis icon to choose a text color in the footer.

Business Rules

Several areas of the product use the same rule syntax and control how data is imported, how cubes are calculated, and how other operations are performed. When this syntax is understood, logic can be written.

See the *API Overview Guide*.

Ways to Write Calculation Logic

There are multiple ways to write calculation logic in OneStream:

Business Rules

Business rules are found under Application Tab > Tools. There are nine types of business rules. These rules can be stored, secured, and assigned to multiple areas of the product, allowing them to be reused.

Complex Expression

Logic can be written as a business rule or as a complex expression within an application artifact, such as a data source. The syntax is the same. The difference is that a business rule can be shared across multiple application artifacts, whereas a complex expression is contained within the artifact.

Member Formula

The same business rule syntax can be applied to Member Formulas. This logic is stored with the member and cannot be shared.

Business Rule Utility Groups

Different utility groups are available when writing business rules:

BRApi

BRApi provides application programming interface to commonly used functions involving all areas of the product where a business rule can be used.

API

The more specific API provided as a parameter to a business rule supplies functions specific to the type of business rule being written. For example, when implementing a business rule for a finance-oriented task, the API refers to the functions used for processing calculation logic and other capabilities related to processing a cube data and metadata.

ARGS

An argument represents a value passed to a business rule when the procedure is called and the calling code supplies the arguments. For example, if a Parser business rule is assigned to the Account dimension, args supplies the Account dimension data and a set of functions available to use against that data. Different args are provided depending on the type of business rule used.

Business Rule Encryption and Decryption

Business rules can be encrypted and decrypted with password protection to prevent unauthorized access. Authorized users can encrypt a business rule by selecting the Encrypt Business Rule button.

> **IMPORTANT:** Save the password used for each encrypted business rule. If the password is lost, the business rule cannot be decrypted for further changes without OneStream Support.

To decrypt a business rule, an authorized user selects the Decrypt Business Rule button. The Decrypt Business Rule dialog box displays, prompting the user for a password. The user enters the password and selects the OK button. The system decrypts the business rule, displays the Business Rule text editor, and returns the to editor mode to read/write.

Set the proper access to the EncryptBusinessRules security role to allow an authorized user to encrypt and decrypt a rule from the Business Rule screen in the Application tab.

Encrypt a Business Rule

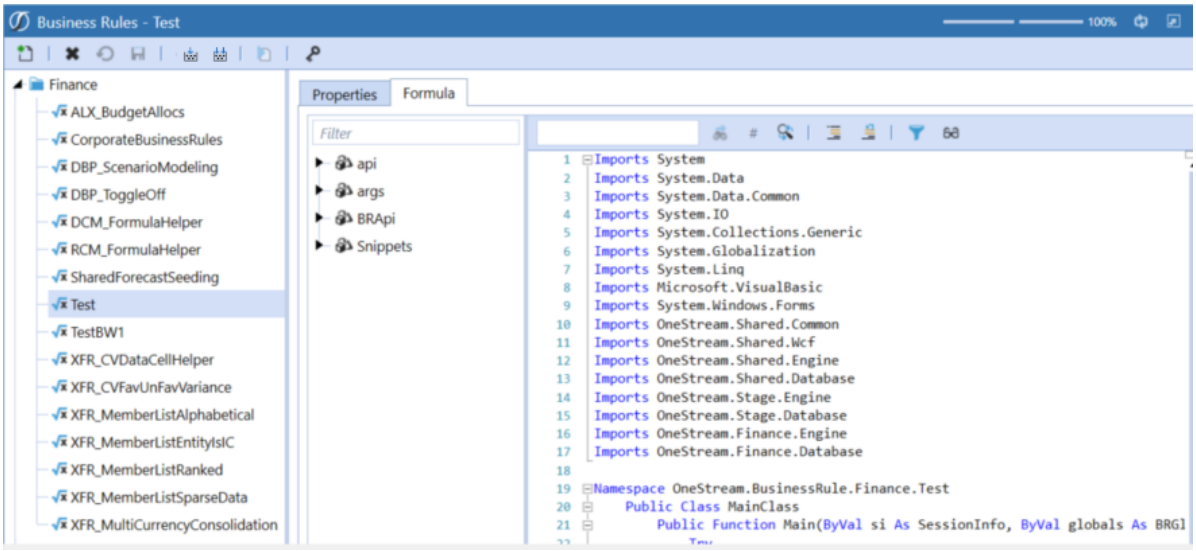
1. Go to **Application** tab > **Tools** > **Business Rules**.
2. Refresh the screen. The **Encrypt Business Rule** button displays in the menu.
3. Select the business rule to encrypt.
4. Select the **Encrypt Business Rule** button.
5. In the Encrypt Business Rule dialog box, enter a password.
6. Reenter the password and select the **OK** button.

The system encrypts the Business Rule, displays the Business Rule Is Encrypted message in the editor, and updates the editor to read-only mode. The Decrypt Business Rule button is now displayed in the menu bar.

Decrypt a Business Rule

1. Select the encrypted business rule.
2. Select the Decrypt Business Rule button.
3. Enter the password and select the **OK** button.

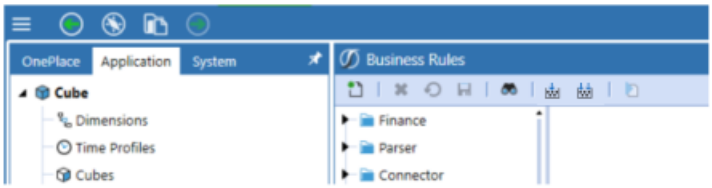
The system decrypts the businessrRule, displays the Business Rule text editor, and returns the editor to read/write mode.



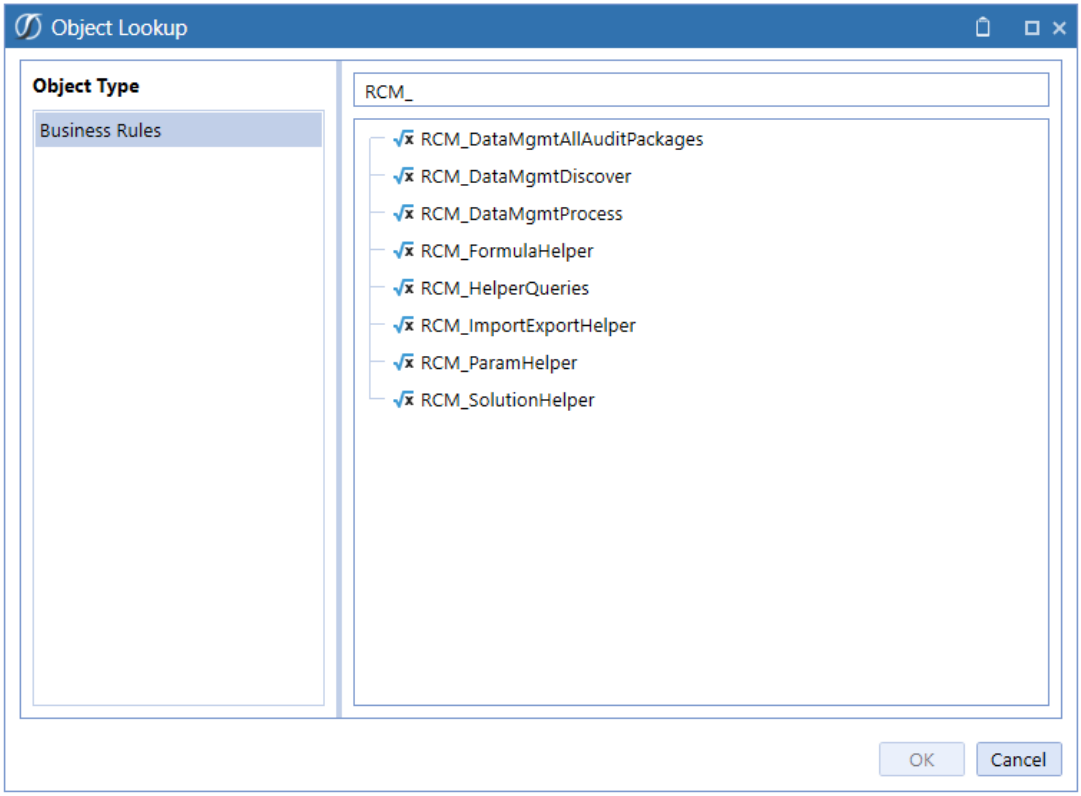
Business Rule Search

Find a business rule quickly by performing a search instead of scrolling through the list.

1. Select the **Application** tab.
2. From **Tools**, select **Business Rules**.



3. Select the **Business Rule Search** icon. The **Object Lookup** dialog box opens.
4. Enter the start of the business rule name. All business rules that match display in the list.



5. Select the business rule from the list, then select the **OK** button.

Business Rule Toolbar and Hot Keys



Compile Business Rule to Check Syntax

Compile a selected business rule to check its syntax.



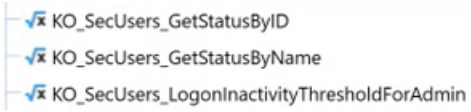
Compile All Business Rules and Formulas to Check Syntax

Compile all business rules and formulas to check the syntax. This is most commonly used after installing a software upgrade where some rule or formula syntax may have changed. This feature is available only to application administrators.



Execute Extender

Runs the selected Extender business rule



Encrypt Business Rule

Encrypt the business rule formula by selecting this button to prompt the user to enter a password in the Encrypt Business Rule dialog box to complete the encryption process.



Decrypt Business Rule

Decrypt the business rule formula by selecting this button to prompt the user to enter a password in the Decrypt Business Rule dialog box to complete the decryption process.

Ctrl + M

Expand or collapse all regions and methods. After selecting the business rule, click in the script to use the hot key.

Business Rule Client Image Types

Client images are specified uisng the XFClientImageType Class in a business rule. These images can display on certain Dashboard Parameter components, for example, Buttons, Maps, SQL Table Editors, and Grid Views.

These are the Image types:

- StatusGrayBall
- StatusWhiteBall
- StatusOrangeBall
- StatusBlueBall
- StatusRedBall
- StatusLightGreenBall
- StatusGreenBall
- StatusGrayCheckMark
- StatusGreenCheckMark
- StatusLockedWithCheckMark
- StatusLockedWithFolder

Business Rule Properties

General

Name: The name of the business rule.

Type: The type of business rule. See Business Rule Types.

Language Type: Programming language used, either Visual Basic or C#.

Contains Global Functions for Formulas: Set to **True** to write a function in a business rule and call it from Member Formulas or other business rules. This is useful when the same code must be copied to multiple Member Formulas. Instead, a public function with two lines of code written in a business rule can be reused. Only use this setting for business rules referenced from Member Formulas because it takes on additional overhead when the system compiles Member Formulas. See the *Understanding the Financial Model Guide*.

Referenced Assemblies: Enter a list of referenced assembly names separated by semicolons.

Reference Another Business Rule: To reference another business rule, enter BR\ followed by the other business rule name.

> **NOTE:** A rule may reference either a VB.NET or C# rule.

BR\SharedFunctionsBR

BR\SharedFunctionsBR

```
Dim sharedFinanceBR As New OneStream.BusinessRule.Finance.SharedFinanceFunctions.MainClass
Dim myResult As String = sharedFinanceBR.Test(si, api, args)
```

See Applying Formulas for more information on referencing a business rule from a Member Formula or business rule.

Reference a DLL File

DLL files can be stored in the file share's Business Rule Assembly folder, an application server configuration file, or a shared network folder referenced by multiple application servers. To reference a DLL file, enter XF\ followed by the DLL filename.

XF\ThirdPartyFunctions.dll
Use no prefix and enter the full path and filename of any DLL file stored on the application server file system.

Is Encrypted: Indicates whether the business rule is encrypted (True) or not (False).

Security

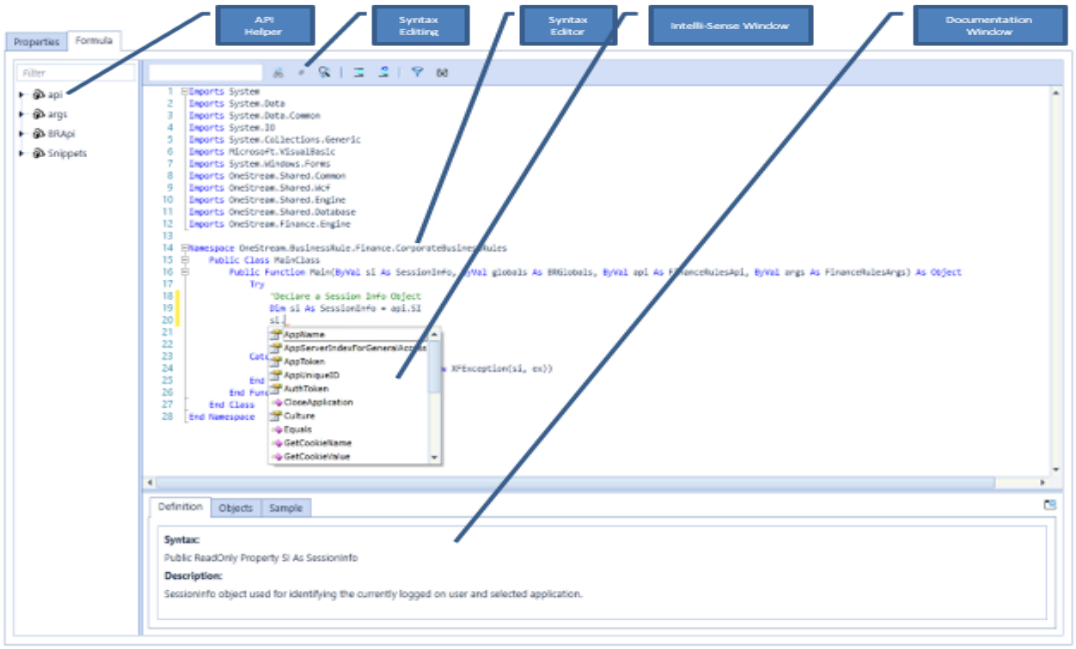
Access Group: Members of this group have access to the business rule.

Maintenance Group: Members of this group have permission to maintain the business rule.

> **NOTE:** Select the ellipsis icon and begin entering the name of the security group in the field. The Groups are filtered making it easier to find and select the desired group. After the group is selected, select the Ctrl keyand double-click to enter the correct name into the appropriate field.

Business Rules Editor Overview

The OneStream Business Rule editor is an in-solution development environment that provides integrated API context help, IntelliSense-based syntax editing, and full outlining capabilities. The syntax content and business rule structure are covered in detail in subsequent sections of this document. The following image highlights the major regions and elements of the Business Rule editor.



Business Rule Types

Finance Rule

Finance business rules are used to generate multidimensional calculations. These business rules are written as shared business rules and applied to a cube or Member Formulas. Eight of these can be assigned to run at the cube level during consolidations.

Example APIs:

- **api.FunctionType**: Used when special logic needs to be run and a specific process needs to be isolated.

Finance Function Type

- **Calculate**: Additional logic during calculation of Entity, Consolidation, Scenario, and Time. Sets one or more values (left side of formula) equal to another (right side) for a qualified Point of View. This is the most common function used.
- **Translate**: Additional logic that uses custom translation.
- **FXRate**: Custom logic used to determine foreign exchange rates for any intersection.
- **Consolidate Share**: Additional logic used during the custom calculation of the share member.
- **Consolidate Eliminiation**: Additional logic used during the custom calculation of the elimination member.
- **Calculation Drill Down Member Formula**: Provides custom drill down results.
- **Conditional Input Rule**: Makes data cells read-only. The settings for this can be done directly on the cube, but using a Conditional Input business rule offers more flexibility and allows the use of the cube settings. This rule can send these returns:
    - `ConditionalInputResultType.Default`
    - `ConditionalInputResultType.NoInput`
    - `ConditionalInputResultType.NoInputAllowCellDetail`
    - `ConditionalInputResultType.NoCellDetailAllowInput`

  The following business rule example will make all cells for the Account 6000 read-only. This should be added to a business rule attached to a cube.

```
Case Is = FinanceFunctionType.ConditionalInput
    If api.Pov.Account.Name.XFEqualsIgnoreCase("6000") Then
        Return ConditionalInputResultType.NoInput
    End If
    Return ConditionalInputResultType.Default
```

- **Confirmation Rules**: Special logic that runs with confirmation rules.
- **Data Cell**: Named GetDataCell calculations that can be reused, for example, a Better/Worse calculation in Cube Views.
- **Dynamic Calc Account**: Special logic to use in Dynamic Calc members.
- **Member List and Member List Headers**: A custom list of members for use in Cube Views and other areas.

Finance Business Rule Example

VB.NET

```
Namespace OneStream.BusinessRule.Finance.CorporateBusinessRules
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As FinanceRulesApi, ByVal args As FinanceRulesArgs) As Object
            Try
                Select Case api.FunctionType
                    Case Is = FinanceFunctionType.Calculate

                        ' Populate Headcount Flow
                        If api.pov.Scenario.name = "Actual" Then

                            api.data.calculate
("A#Headcount:F#BegBalHC=A#Headcount:F#None:T#POVPrior1")

                            api.data.calculate("A#Headcount:F#EndBalHC=A#Headcount:F#None")

                            api.data.calculate
("A#Headcount:F#CheckHC=A#Headcount:F#BegBalHC+A#Headcount:F#TotalMovementHC-
A#Headcount:F#EndBalHC")
                        End If

                        If api.pov.Scenario.name = "Variance" Then
                            api.data.Calculate("F#BudgetEBITDA:A#[EBITDAVar] =
A#EBITDA:S#Budget:F#None")
                            api.data.Calculate("F#ActualEBITDA:A#[EBITDAVar] =
A#EBITDA:S#Actual:F#None")
                            api.data.Calculate("F#[Bad Debt]:A#[EBITDAVar] =
(A#54100:S#Budget:F#None-A#54100:S#Actual:F#None)")

                            api.data.Calculate("F#VarianceCheckSum:A#
[EBITDAVar]=F#BudgetEBITDA:A#[EBITDAVar]+F#VarianceEBITDA:A#[EBITDAVar]-F#ActualEBITDA:A#
[EBITDAVar]")
                        End If
                End Select

                Return Nothing
            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function
    End Class
End Namespace
```

Parser Rule

Parser business rules are used to evaluate and modify field values processed by the Stage Parser Engine as it reads source data. These business rules are written as Shared business rules or logical expressions and applied to a Data Source dimension.

Example API

- **args.Line**: Returns the entire record being processed from the data source.
- **args.Value**: Returns the dimension the business rule is assigned to in the data source.

Common Usage

- Custom parsing logic
- Field value concatenation
- Field value bypassing
- Evaluate field other than current field being parsed

Parser Business Rule Example

VB.NET

```
Namespace OneStream.BusinessRule.Parser.XFR_ParseDelimitedTextAndGetField
    Public Class MainClass
        '----------------------------------------------------------------------------------------
        ' Reference Code: XFR_ParseDelimitedTextAndGetField
        ' Description:    Parse a delimited string and return a selected field from the
delimited list.
        ' Usage:          Parser Business Rule intended to be used on any field within a
Data Source.
        '----------------------------------------------------------------------------------------
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As ParserDimension, ByVal args As ParserArgs) As Object
            Try
                ' Parse the line
                Dim fields As List(Of String) = StringHelper.SplitString(args.Line, ",",
StageConstants.ParserDefaults.DefaultQuoteCharacter)
                If fields.Count >= 2 Then
                    ' Return the 2nd field (Zero based index)
                    Return fields(1)
                Else
                    ' Line invalid, just return incoming value and let parser engine handle
the evaluation
                    Return args.Value
                End If
            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function
    End Class
End Namespace
```

Connector Rule

Connector business rules are used to communicate with, collect data from, and drill back to external systems. These Business Rules are written as Shared Business Rules and applied to a data source.

See the *Data Collection Guide* for more information on using connectors.

Example API

- **args.ActionType**: Returns one of the four available Connector action types.
- **ConnectorActionTypes.GetFieldList**: Runs the SQL query to return the available field list to the data source for dimension assignment.
- **ConnectorActionTypes.GetData**: Runs the SQL query to retrieve the source data and return it to the Stage based on the Data Source dimension assignment.
- **ConnectorActionTypes.GetDrillBackTypes**: Returns a list of the specified drill back types, such as File or Data Grid.
- **ConnectorActionTypes.GetDrillBack**: Runs the required SQL query against the source system and provides more detail than what was originally imported.

Connector Business Rule Example

`NamespaceOneStream.BusinessRule.Connector.RevenueMgmtHouston`

VB.NET

```
Public Class MainClass
    Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As
Transformer, ByVal args As ConnectorArgs) As Object
        Try
            ' Get the query information
            Dim connectionString As String = GetConnectionString(si, globals, api)

            ' Get the Field name list or load the data
            Select Case args.ActionType
                Case Is = ConnectorActionTypes.GetFieldList
                    ' Return Field Name List
                    Dim fieldListSQL As String = GetFieldListSQL(si, globals, api)
                    Return api.Parser.GetFieldNameListForSQLQuery(si, DbProviderType.OLEDB,
connectionString, True, fieldListSQL, False)

                Case Is = ConnectorActionTypes.GetData
                    ' Process Data
                    Dim sourceDataSQL As String = GetSourceDataSQL(si, globals, api)
                    api.Parser.ProcessSQLQuery(si, DbProviderType.OLEDB, connectionString,
True, sourceDataSQL, False, api.ProcessInfo)
                    Return Nothing

                Case Is = ConnectorActionTypes.GetDrillBackTypes
```

```
                    ' Return the list of Drill Types (Options) to present to the end user
                    Return Me.GetDrillBackTypeList(si, globals, api, args)

                Case Is = ConnectorActionTypes.GetDrillBack
                    ' Process the specific Drill-Back type
                    Return Me.GetDrillBack(si, globals, api, args,
args.DrillBackType.DisplayType, connectionString)
            End Select

        Catch ex As Exception
            Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
        End Try
    End Function

    ' Create a Connection string to the External Database
    Private Function GetConnectionString(ByVal si As SessionInfo, ByVal globals As
BRGlobals, ByVal api As Transformer) As String
        Try
            ' Named External Connection in App Server Config
            '-----------------------------------------------------
            Return "RevenueMgmtSystem"

        Catch ex As Exception
            Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
        End Try
    End Function

    ' Create the field list SQL Statement
    Private Function GetFieldListSQL(ByVal si As SessionInfo, ByVal globals As BRGlobals,
ByVal api As Transformer) As String
        Try
            ' Create the SQL Statement
            Dim sql As New Text.StringBuilder

            sql.Append("SELECT Top(1)")
            sql.Append("TransID, PlantCode, CustId, CustName, InvNo, InvYear, InvMonth,
InvDesc, GLAccount, WorkDay, ProdModel, BomCode, UnitPrice, Units, Amount, DestinationCode
")

            sql.Append("FROM InvoiceDocumentDetail ")
                        Return sql.ToString

        Catch ex As Exception
            Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
        End Try
    End Function

    ' Create the data load SQL Statement
    Private Function GetSourceDataSQL(ByVal si As SessionInfo, ByVal globals As BRGlobals,
ByVal api As Transformer) As String
        Try
            ' Create the SQL Statement
            Dim statement As New Text.StringBuilder
            Dim selectClause As New Text.StringBuilder
            Dim fromClause As New Text.StringBuilder
            Dim whereClause As New Text.StringBuilder
            Dim orderByClause As New Text.StringBuilder
```

```
            selectClause.Append("SELECT ")
            selectClause.Append("TransID, PlantCode, CustId, CustName, InvNo, InvYear,
InvMonth, InvDesc, GLAccount, WorkDay, ProdModel, BomCode, UnitPrice, Units, Amount,
DestinationCode ")

            fromClause.Append("FROM InvoiceDocumentDetail ")

            whereClause.Append("WHERE ")
            ' Get the YEAR from the current XF Workflow Unit TimeKey
            whereClause.Append("(")
            whereClause.Append("InvYear = " & TimeDimHelper.GetYearFromId
(api.WorkflowUnitPk.TimeKey).ToString)
            whereClause.Append(")")

            ' Get the MONTH from the current XF Workflow Unit TimeKey
            whereClause.Append(" And ")
            whereClause.Append("(")
            whereClause.Append("InvMonth = 'M" & TimeDimHelper.GetSubComponentsFromId
(api.WorkflowUnitPk.TimeKey).Month.ToString & "'")
            whereClause.Append(") ")

            ' Select Houston Plant Codes
            whereClause.Append(" And ")
            whereClause.Append("(")
            whereClause.Append("PlantCode IN('H200','H210')")
            whereClause.Append(") ")

            orderByClause.Append("ORDER BY ")
            orderByClause.Append("PlantCode, CustId, WorkDay, ProdModel, DestinationCode")

            ' Create the full SQL Statement
            statement.Append(selectClause.ToString)
            statement.Append(fromClause.ToString)
            statement.Append(whereClause.ToString)
            statement.Append(orderByClause.ToString)

            Return statement.ToString

        Catch ex As Exception
            Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
        End Try
    End Function

    ' Create the drill back options list
    Private Function GetDrillBackTypeList(ByVal si As SessionInfo, ByVal globals As
BRGlobals, ByVal api As Transformer, ByVal args As ConnectorArgs) As List(Of
DrillBackTypeInfo)
        Try
```

Conditional Rule

Conditional rules (mapping) are used to conditionally evaluate mapping criteria during the data transformation process. These business rules are written as Shared business rules or logical expressions and applied to a transformation rule definition. They are applicable to transformation rules with the type of Composite, Range, List, or Mask either as a business rule or complex expression.

Example APIs

- **args.GetSource**: Returns the source value for the specified dimension.
- **args.GetTarget**: Returns the mapped or transformed value for the specified dimension.
- **args.OutputValue**: Returns the originally mapped target value from the transformation rules.

Conditional Business Rule Example

> **Example:** Conditional rule for dimension member mapping based on source values.

> **NOTE:** If the business rule does not call args.OutputValue, the target field can be empty and will not be considered.

VB.NET

```
Namespace OneStream.BusinessRule.ConditionalRule.XFR_Conditional_Source
    Public Class MainClass
        '-------------------------------------------------------------------------------------
-----------------
        ' Reference Code: XFR_Conditional_Source
        ' Description:    Demonstrates how to map a flow Dimension based on the source
account.
        ' Usage:          Used for conditional flow Member mapping based on source account.
        '-------------------------------------------------------------------------------------
```

```
-----------------
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Transformer, ByVal args As ConditionalRuleArgs) As Object
            Try
                ' Grab the source account value
                Dim account As String = args.GetSource("A#")

                ' Assign the flow Member based on the source account
                If account.StartsWith("1190") Then
                    Return "Additions"
                Else
                    ' Just return value assigned to the Transformation Rule
                    Return args.OutputValue
                End If

                Return Nothing

            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function
    End Class
End Namespace
```

Conditional Business Rule Example

> **Example:** Conditional rule for dimension member mapping based on target values.

VB.NET

```
Namespace OneStream.BusinessRule.ConditionalRule.XFR_Conditional_Target
    Public Class MainClass
        '-------------------------------------------------------------------------------------
-----------------
        ' Reference Code: XFR_Conditional_Target
        ' Description:    Demonstrates how to map a flow Dimension based on the target
account.
        ' Usage:          Used for conditional flow Member mapping based on target account.
        '-------------------------------------------------------------------------------------
-----------------
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Transformer, ByVal args As ConditionalRuleArgs) As Object
            Try

                ' Grab the target account value
                Dim account As String = args.GetTarget("A#")

                ' Assign the flow Member based on the first character of the target account
```

```
                Select Case Left(account, 1)
                    Case "1", "2", "3"
                        Return "End_Inp"
                    Case Else
                        Return "None"
                End Select

            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function
    End Class
End Namespace
```

Derivative Rule

Derivative rules, which derive data befpre mapping, are used to evaluate or calculate values during the data derivation process. These business rules are written as Shared business rules or logical expressions and applied to a derivative rule definition. They are only applicable to transformation rules with the type of Derivative either as a business rule or complex expression.

Common Usage

- Calculate mathematical expressions
- Look up value from transformation cache for use in calculations
- Look up value from cube for use in calculations
- Source system check rule logic (validation rules on source data)

Derivative Business Rule Example

VB.NET

```
Namespace OneStream.BusinessRule.DerivativeRule.XFR_ExecuteStageCacheQuery
    Public Class MainClass
        '------------------------------------------------------------------------------------------
        ' Reference Code: XFR_ExecuteStageCacheQuery
        '
        ' Description:    Retrieves a value from the transformation cache calculated by
another derivative rule
        '                 and uses the value as an input to calculate the ratio of one
account to another.
        '
        ' Usage:          Derivative Business Rule intended to demonstrate how to retrieve a
value from the
        '                 transformation cache and use it as input to a calculation.
        '------------------------------------------------------------------------------------------
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Transformer, ByVal args As DerivativeRuleArgs) As Object
            Try
                ' Get Cash Value by querying the cache
                Dim valueCash As Decimal = Decimal.Zero
                Using dtCash As DataTable = api.QueryDataCache(si, "Ac = 'Cash'", False,
True)
                    If Not dtCash Is Nothing Then
                        For Each row As DataRow In dtCash.Rows
                            valueCash = row("Am")
                            Exit For
                        Next
                    End If
                End Using

                ' The current value is AR
                Dim valueAR As Decimal = args.ColumnValue

                ' Return the ratio of AR/Cash
                If (valueAR <> Decimal.Zero) And (valueCash <> Decimal.Zero) Then
                    Return (valueAR / valueCash)
                Else
                    ' Just return the value from the Stage
                    Return args.ColumnValue
                End If
                Return Nothing

            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function
    End Class
End Namespace
```

Cube View Extender

Cube View Extender rules are used to apply advanced Cube View formatting to any Cube View while in report view.

> **NOTE:** These rules do not apply to Cube View formatting in the data explorer grid view.

The Extender rule is used with the Execute Cube View Extender Business Rule setting on the Cube View. See "General Settings" in the *Presentation Guide*.

See the *API Overview Guide*.

Common Usage

The following are key uses for Cube View Extender business rules in formatting reports.

- Update headers and footers
    - Display different logos on select reports based on conditional logic or security and manage their placement and size
    - Customize the page number in the header or footer

      > **NOTE:** Page numbers can be on the top or bottom row of a report and the horizontal position can be specified for rows. This only applies to the top or bottom rows.

    - Format individual header and footer fields
    - Control the left, right, center subtitle widths
    - Control the font size of title and subtitles
    - Customize the date display

- Update column and row headers
    - Customize bottom text alignment
    - Control word wrap
- Apply conditional formatting to data cells
    - Format cells based on their contents
    - Change the text color of a value to effectively hide the result

Dashboard DataSet

Dashboard DataSet Rules are used to create programmatic query results This rule type combines multiple types of data into a single result set using the full syntax capability of VB.Net. These business rules are written as Shared business rules and applied to dashboard data adapters or dashboard parameters.

Common Usage

- Combine different types of data for a report
- Build programmatic data queries, for example, analytic plus SQL
- Conditionally build data query reports
- Conditionally build data query for parameters
- Create geographical data to display using a Map parameter component in a dashboard
- Create a data series to display using a Chart (Advanced) parameter component in a dashboard

Dashboard Extender

Dashboard Extender rules are used to perform a variety of tasks associated with custom dashboards and OneStream Solutions. These business rules are part of multipurpose rules and make up the majority of the code written in a OneStream Solution. In addition, they are written as Shared business rules and applied to application dashboard parameter components, for example, Buttons and Combo Boxes.

Common Usage

- Runs task when the user selects a button
- Perform a task and show a message to the user
- Perform a custom calculation
- Upload a file from the end user's machine
- Automate a workflow
- Build a custom workflow
- Create custom data tables
- Include page state to store parameters and values about a specific dashboard page instance

Dashboard XFBRString

Dashboard XFBRString rules are used to process conditional dashboard parameters. These rules inspect and update a dashboard parameter value using the full syntax capabilities of VB.Net or C#. Dashboard string functions are written as Shared business rules and called by using a XFBR (BusinessRuleName, FunctionName, UserParam=[UserValue]) specification anywhere a standard dashboard parameter is used. After a dashboard BRString rule is created, create a Dashboard component to call the BRString using the following script.

```
BRString(brRuleName, funcNameWithinBRRule, optionalName1 = var1, optionalName2 = var2)
```

Enter the business rule as a parameter using |!BRString!|. The return value from the business rule will be used in the Dashboard component.

> **NOTE:** This business rule can be applied to any dashboard or Cube View property where a parameter is used.

Extensibility Rule

Extensibility rules include Extender and Event Handlers. Full administrator rights are required to edit these rules.

- **Extender**: Most generalized type of business rule. They are used to write a simple utility function or a specific helper function run in a data management job.
- **Event Handlers**: Runs logic before or after a certain operation occurs within the system.

Extensibility Business Rule Example

> **Example:** Event Handler that sends an email notification after a ProcessCube event.

VB.NET

```
Namespace OneStream.BusinessRule.DataQualityEventHandler.DataQualityEventHandler
    Public Class MainClass
        '------------------------------------------------------------------------------------------
        ' Reference Code: DataQualityEventHandler
        '
        ' Description:    Event handler method that provides an opportunity to supplement a normal data
        '                 quality action with your own custom functionality.
        '                 (Example: email after ProcessCube or publish report to SharePoint after
        '                 failed Confirmation).
        '
        ' Usage:          Executes when a Data Quality action is run and fires this Business Rule. If you have
        '                 written code in that handles the specified event operation the code will be executed.
        '------------------------------------------------------------------------------------------
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As Object, ByVal args As DataQualityEventHandlerArgs) As Object
            Try
                ' Define a switch to control event processing, since many of these are reference examples we do not want them to run all the time
                Dim processEvents As Boolean = False

                ' Set the default return values
                Dim returnValue As Object = args.DefaultReturnValue
                args.UseReturnValueFromBusinessRule = False
                args.Cancel = False

                ' Evaluate the operation type in order to determine which subroutine to process
                Select Case args.OperationName
                    Case Is = BREventOperationType.DataQuality.ProcessCube.NoCalculate
                        ' Execute a Data Management job after process Cube runs
                        Me.XFR_HandleProcessCubeNoCalculate(si, globals, api, args)

                    Case Is = BREventOperationType.DataQuality.Certify.FinalizeSetCertifyState
                        ' Send an email after a Workflow Profile executes its certification
                        Me.XFR_HandleFinalizeSetCertifyState(si, globals, api, args)

                End Select
```

```
                Return returnValue

            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function

#Region "ProcessCube.NoCalculate Helpers"

        Private Sub XFR_HandleProcessCubeNoCalculate(ByVal si As SessionInfo, ByVal globals
As BRGlobals, ByVal api As Object, ByVal args As DataQualityEventHandlerArgs)
            '------------------------------------------------------------------------------
--------------------
            ' Reference Code: XFR_HandleProcessCubeNoCalculate
            '
            ' Description:    Run a Data Management Sequence after the Workflow process Cube
task is run.
            '
            '                 Note: the Data Management Sequence name is assigned to a
Workflow Profile
            '                 CalcDef filter field so this event does not have to be
modified, the user
            '                 can simply edit the CalcDef grid for a Workflow Profile and
this Business
            '                 Rule will execute the specified sequence.
            '
            ' Usage:          Used to supplement the standard "ProcessCube" functionality
associated
            '                 with a Workflow Profile by allowing a Data Management Sequence
to be
            '                 executed for the Workflow Profile as well.
            '------------------------------------------------------------------------------
--------------------
            Try
                ' Get the DataUnitInfo from the Event arguments so that we can get the name
of the DataManagement sequence to process.
                Dim calcInfo As DataUnitInfo = DirectCast(args.Inputs(2), DataUnitInfo)
                If Not calcInfo Is Nothing Then
                    ' Make sure that a Sequence name as assigned to the filter value of the
Calc Definition of the executing Workflow Profile
                    If calcInfo.FilterValue <> String.Empty Then

                        ' Now, execute the DataMgmt Sequence that was specified in the
FilterValue (in a background thread)
                        BRApi.Utilities.StartDataMgmtSequence(si, calcInfo.FilterValue,
Nothing)

                    End If
                End If

            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Sub
```

```
#End Region

#Region "FinalizeSetCertifyState Helpers"
        ' Apply tag this to Text1 of users that should be notified when a child Workflow
certifies
        Public Const NotifyCertifyCompleted As String = "#NCC"

        Private Sub XFR_HandleFinalizeSetCertifyState(ByVal si As SessionInfo, ByVal globals
As BRGlobals, ByVal api As Object, ByVal args As DataQualityEventHandlerArgs)
            '------------------------------------------------------------------------------
---------------------
            ' Reference Code: XFR_HandleFinalizeSetCertifyState
            '
            ' Description:    Send an email to users assigned the Workflow execution group
of the Parent
            '                 Workflow Profile of the Workflow Profile that just certified.
            '
            ' Usage:          Used to provide email notification when a dependent Workflow
Profile
            '                 certifies.
            '------------------------------------------------------------------------------
---------------------
            Try

                ' Check the before / after flag, we want to handle the AFTER event
                If (args.IsBeforeEvent = False) Then

                    ' Check to see if the profile was certified, add send an email all
Parents of the certifying profile
                    Dim certifyInstance As CertifyInstanceInfo = DirectCast(args.Inputs(2),
CertifyInstanceInfo)
                    If Not certifyInstance Is Nothing Then
                        If certifyInstance.IsCertified(si) Then

                            ' Get the list of users assigned to the Parent of the Workflow
profile so that we know who should get an email
                            Dim notificationUsers As List(Of UserInfo) =
BRApi.Workflow.General.GetUsersInWorkflowGroupForParents(si,
certifyInstance.Workflow.WfUnitPk.CreateWorkflowUnitClusterPk(),
SharedConstants.WorkflowProfileAttributeIndexes.WorkflowExecutionGroup, True, True,
NotifyCertifyCompleted)
                            For Each userSummary As UserInfo In notificationUsers
                                ' Send the email
                                Me.CreateMessageAndSendMail(si, certifyInstance,
userSummary)
                            Next
                        End If

                    End If

                End If

            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Sub
```

```
        Private Sub CreateMessageAndSendMail(ByVal si As SessionInfo, certInstance As
CertifyInstanceInfo, userSummary As UserInfo)
            '---------------------------------------------------------------------------
---------------------
            ' Reference Code: CreateMessageAndSendMail
            '
            ' Description:    Create the email message body and execute the "Send Mail"
function.
            '---------------------------------------------------------------------------
---------------------
            Try
                Using dbConnApp As DbConnInfo = BRApi.Database.CreateApplicationDbConnInfo
(si)

                    Dim emailInfo As New Text.StringBuilder
                    emailInfo.AppendLine("To: " & userSummary.User.Email)

                    ' Create the message title
                    Dim messageTitle As New Text.StringBuilder
                    messageTitle.Append("OneStream Workflow [")
                    messageTitle.Append(certInstance.WorkflowProfileName)
                    messageTitle.Append(", ")
                    messageTitle.Append(ScenarioDimHelper.GetNameFromId(dbConnApp,
certInstance.Workflow.WfUnitPk.ScenarioKey))
                    messageTitle.Append(", ")

                    messageTitle.Append(TimeDimHelper.GetNameFromId
(certInstance.Workflow.WfUnitPk.TimeKey))
                    messageTitle.Append("] is ready for you to review.")
                    messageTitle.AppendLine("")

                    ' Create the message body
                    Dim messageBody As New Text.StringBuilder
                    messageBody.AppendLine("Certification Information:")
                    messageBody.AppendLine("-----------------------------------------------
------------------")

                    messageBody.AppendLine("Profile Name........ " &
certInstance.WorkflowProfileName)

                    Dim signOff As CertifySignOffInstanceInfo = Nothing
                    If certInstance.SignOffsInDescendingTimestampOrder.Count > 0 Then
                        signOff = certInstance.SignOffsInDescendingTimestampOrder(0)
                        messageBody.AppendLine("User Name........... " & signOff.UserName)

                        messageBody.AppendLine("Time Completed...... " &
signOff.TimeStamp.ToLocalTime & " Local Time")

                        messageBody.AppendLine("Comments............ " & signOff.Comments)

                    End If
                    messageBody.AppendLine("")

                    messageBody.AppendLine("Dependent Information:")
                    messageBody.AppendLine("-----------------------------------------------
------------------")
```

```
                    For Each relativeInfo As CertifyRelativeInfo In certInstance.Dependants
                        messageBody.AppendLine(relativeInfo.WorkflowProfile.Name & "  (" &
relativeInfo.Workflow.GetOverallStatusText("Not Started", "Completed", "Locked", "Running",
"") & ")")
                    Next
                    messageBody.AppendLine("")

                    ' Test the email by writing a log message
                    ErrorHandler.LogMessage(si, emailInfo.ToString & vbCrLf &
messageTitle.ToString & vbCrLf & messageBody.ToString)

                    ' Send an email message
                    ' Me.SendMail(si, userSummary.User.Email, messageTitle.ToString,
messageBody.ToString)

                End Using

            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Sub

        Private Sub SendMail(ByVal si As SessionInfo, ByVal userEmailAddress As String,
ByVal subject As String, ByVal messageBody As String)
            '--------------------------------------------------------------------------------
--------------------
            ' Reference Code: SendMail
            '
            ' Description:    Create the mail connection and send.
            '--------------------------------------------------------------------------------
--------------------

            Try
                ' Prepare the message
                Dim emailConnectionName As String = "Office365Mail"
                Dim toEmail As New List(Of String)
                toEmail.Add("userEmailAddress")
                Dim subject As String = "Subject"
                Dim body As String = "Body"
                Dim attachments As New List(Of String)
                attachments.Add("FileLocationString")

                ' Send the Message
                BRApi.Utilities.SendMail(si, emailConnectionName, toEmail, subject, body,
attachments)

            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Sub

#End Region

    End Class
End Namespace
```

Extender

This can be used to automate custom tasks, for example, running external scripts, backups, and report publishing.

WCF Event Handler
Enables a direct interaction with the Microsoft Windows Communication Foundation which means it listens to communication between the client and the web server. The rule will intercept the communication, analyze it, and if certain criteria is met, it will run its logic. This is quite flexible and has a variety of uses, for example, creating, reading, deleting, and updating different types of objects in the system for users in a group or transformation rule changes. For example, a rule can be created to email an auditor about every metadata change as it happens.

Available Operations

Transformation Event Handler
Runs at various points from import through load. These are the available operations:

- StartParseAndTransForm
- InitializeTransFormer
- ParseSourceData
- LoadDataCacheFromDB
- ProcessDerivativeRules
- ProcessTransformationRules
- DeleteData
- DeleteRuleHistory
- WriteTransFormedData

- SummarizeTransFormedData
- CreateRuleHistory
- EndParseAndTransForm
- FinalizeParseAndTransForm
- StartRetransForm
- EndRetransForm
- FinalizeRetransForm
- StartClearData
- EndClearData
- FinalizeClearData
- StartValidateTransForm
- ValidateDimension
- EndValidateTransForm
- FinalizeValidateTransForm
- StartValidateIntersect
- EndValidateIntersect
- FinalizeValidateIntersect
- LoadIntersect
- StartLoadIntersect
- EndLoadIntersect
- FinalizeLoadIntersect

Journals Event Handler

Run before, during, or after a journal operation, for example, submission, approval, or post. These are the available operations:

- SubmitJournal
- ApproveJournal
- RejectJournal
- PostJournal
- UnpostJournal
- StartUpdateJournalWorkflow
- EndUpdateJournalWorkflow
- FinalizeUpdateJournalWorkflow

Save Data Event Handler

Run to track all save events in an application.

Forms Event Handler

Run before, during, or after an operation, for example, form save. These are the available operations:

- SaveForm
- CompleteForm
- RevertForm
- StartUpdateFormWorkflow
- EndUpdateFormWorkflow
- FinalizeUpdateFormWorkflow

Data Quality Event Handler

Run before, during, or after data quality events, for example, confirmation and certification. These are the available operations:

- StartProcessCube
- Calculate
- Translate
- Consolidate
- EndProcessCube
- FinalizeProcessCube
- PrepareICMatch
- StartICMatch
- PrepareICMatchData
- EndICMatch
- StartConfirm

- EndConfirm
- FinalizeConfirm
- SaveQuestionResponse
- StartSetQuestionnaireState
- SaveQuestionnaireState
- EndSetQuestionnaireState
- StartSetCertifyState
- SaveCertifyState
- EndSetCertifyState
- FinalizeSetCertifyState

Data Management Event Handler

Run before or after a data management sequence or step runs. These are the available operations:

- StartSequence
- ExecuteStep
- EndSequence

Workflow Event Handler

Run before or after a workflow execution step. These are the available operations:

- UpdateWorkflowStatus
- WorkflowLock
- WorkflowUnlock

Business Rules

A business rule is a VB.Net class meaning each business rule is an independent object encapsulating VB.Net code. A business rule can be a one-line call to write a log message, or it can be a full code library containing other custom VB.Net classes, methods, and properties.

This section provides a detailed explanation of the following topics:

- Platform engines
- Business rule structure and fundamentals
- Business rule classifications
- Specific business rule types
- Business rule organization
- OneStream business rule framework
- Best practices for business rule architecture
- Business rule organization and referencing

> **CAUTION:** When creating custom business rules that write to the error log, do not write sensitive information to the log. When writing to the error log, the OneStream application attempts to filter sensitive information and redact it. Information in the log that is filtered is replaced with the text: `[Redacted]`.

> **IMPORTANT:** For OneStream, partners, and customers, we strongly advise against any sensitive or confidential information be included in business rules.

See the API Overview Guide.

Platform Engines

The platform contains multiple processing engines. These engines have distinct responsibilities with respect to system processing and, consequently, expose different API interfaces to the business rules they call. This section provides a brief overview of each engine in the platform and describes the core responsibilities.

Workflow Engine

The workflow engine controls and tracks the status of the business processes defined in the workflow hierarchies. This engine is primarily accessed through the BRApi and can be called from other engines to check workflow status during processes. The workflow engine provides an event model that allows each workflow process to be evaluated and reinforced with customer-specific business logic, if required.

See "Appendix 2: Event Listing" in the *API Overview Guide*.

Stage Engine

The stage engine sources and transforms external data into valid analytic data points. The main responsibility of this engine is to read source data (files or systems) and parse the information into a tabular format. This allows the data to be transformed or mapped to valid members defined by the finance engine. The stage engine is an in-memory, multi-threaded engine that enables interaction with source data as it is being parsed and transformed. In addition to parsing and transforming data, the stage engine includes calculation capabilities that enable data to be derived and evaluated based on incoming source data. The stage engine provides quality services to source data by validating, mapping, and running derivative check rules.

Finance Engine

The finance engine is an in-memory financial analytic engine. The main responsibility of this engine is to enrich and aggregate base data cells into consolidated multi-dimensional information. The finance engine enables sophisticated financial calculations through centralized business rules and member-specific business rules (Member Formulas). It works concurrently with the stage engine to validate incoming intersections and works with the data quality engine to run confirmation rules used to validate analytic data values.

Data Quality Engine

The data quality engine controls the data confirmation and certification processes. The confirmation engine defines and controls the sequence of data value checks required to assert that information submitted from a source system is correct. The certification engine manages user certifications and determines the completion status of workflow dependents. This engine is primarily accessed through the BRApi and may be called from other engines to check data quality status during processes.

Data Management Engine

The data management engine provides task automation services to the platform. This engine runs batches of commands that are organized into sequences that contain steps. Steps represent entry points or mechanisms to run features of other engines.

> **Example:** The clear data step uses the services of the finance engine.

In addition, the data management engine can run a business rule step that runs a custom business rule as part of a data management sequence. This is an incredibly powerful capability because it provides the ability to string together any combination of predefined processing steps with custom business rule steps.

Presentation Engine

The presentation engine provides data visualization services to the platform. The presentation engine contains the following component engines: Cube View engine, dashboard engine, parameter engine, book engine, and Extensible Document engine. The presentation engine manages and delivers content to end users and provides a development environment for custom user interface elements. This engine enables OneStream Solution application development capabilities and continues to evolve with each product release. Like the data management engine, the presentation engine interacts with and can call the services of all other engines in the platform.

Formulas

A formula is a set of calculation instructions to compute values. Formulas are written using Microsoft Visual Basic .NET procedures that use OneStream API function libraries and member script expressions. These capabilities provide a powerful programming environment delivering reliable compiled formula definitions.

See the following sections:

- Formula Structure
- Applying Formulas
- Formula Types
- Formula Execution
- Writing Formulas
- Writing Stored Calculations
- Avoiding Data Explosion

Formula Structure

Microsoft Visual Basic .NET with OneStream API and Member Scripts

All formulas and business rules run as compiled VB.NET code. In a VB.NET function or subroutine, calls are made to specific API functions that enable the rule writer to interact with the analytic engine. Specific API functions are used to process member script expressions and create calculated values in the analytic data model.

Common Finance APIs that Use Member Scripts

```
api.Data.GetDataCell("A#Cash")
```

This returns a single numeric cell value.

```
api.Data.GetDataCellEx("A#Cash")
```

This returns a single numeric or text cell value.

```
api.Data.SetDataCell("A#Cash", 5.00, False)
```

This saves and writes a single data cell value.

```
api.Data.Calculate("A#Cash = A#Sales * 0.10")
```

This saves multiple data cells as stored values.

Basic Dynamic Cell Calculation

The following examples return a single cell.

Calculate Current Ratio

```
Return api.Data.GetDataCell("A#15000:O#Top / A#22000:O#Top")
```

Formula Composition

```
VB.NET    Language Keyword(s)
Return
```

OneStream API Function

```
api.Data.GetDataCell("Member Script”)
```

OneStream Member Script

```
A#15000:O#Top / A#22000:O#Top
```

> **TIP:** When using a nested function (YearPrior(|POVTime|) in a GetDataCell call, you must enclose the function in square brackets []. This is necessary because GetDataCell and the functions use “(“and “)” as indicators of the start points and end points in the script. The inner functions must be wrapped in square brackets. GetDataCell("S#[|!ScenarioActual_GRT!|]:T#[YearPrior1(|CVTime|)][Period (|CVTime|)]"):Name("PYR Var")

Basic Stored Calculation

The following examples calculate and store multiple cells.

Pull Prior Period Retained Earnings

```
api.Data.Calculate("A#28000 = A#28999:T#POVPrior1")
```

Formula Composition

Formula Composition

```
OneStream API    Function
api.Data.Calculate("Member Script”)
```

Member Script

```
A#28000=A#28999:T#POVPrior1
```

You can use formula variables in member scripts to significantly improve performance when the same formula is used for multiple members. When using formula variables, the formula text remains the same, so there is no need for continued parsing and evaluation.

Using variables can also improve performance if a member ID is used instead of a member name because the ID can also be used as the value in a formula variable. To use a formula variable in a member script, use a dollar sign $ instead of a pound sign # before the member name, and use the variable name after the dollar sign.

Example 1

```
api.Data.FormulaVariables.SetTextVariable(“variableAccount”, “8150”)
api.Data.Calculate(“A#8250=A$variableAccount * 10”)
```

Example 2

```
Dim acctMember As Member = api.Members.GetMember(DimType.Account.Id, “8150”)
```

```
api.Data.FormulaVariables.SetMemberVariable("variableAccount",acctMember)
api.Data.Calculate("A#8250= A$variableAccount * 100")
```

Example 3

```
Dim acctMember As Member = api.Members.GetMember(DimType.Account.Id, "8150")
Dim acctId As Integer = acctMember.MemberId
api.Data.FormulaVariables.SetIntegerVariable("myAccount",acctId)
api.Data.Calculate("A#8250 = A$myAccount * 1000")
```

Applying Formulas

An administrator can create a formula on a dimension member or in a business rule.

Member Formulas

The preferred and more common approach is to write formulas on dimension members. Member Formulas are written using the Formula property on individual Scenario, Account, Flow, or User Defined members in the Dimension Library. The primary reasons for writing formulas on dimension members are that they provide intuitive formula organization and promote reusability of dimensions and members with their associated calculations across multiple cubes. They also enable parallel processing for performance optimization using advanced multi-threading that runs multiple formulas concurrently. Writing formulas on dimension members supports drill-down from a calculated amount to the amounts used as the source for the calculation and supports the ability to vary formulas by Scenario and Time. This is useful because changes can be made without affecting calculations for older data or data in other scenarios.

Finance Business Rules

Formulas can also be written using one or more business rule files. While this is typically not the preferred approach, it provides the ability to place all formulas in a single location, similar to the approach used by some older competitive products. The primary reasons for writing formulas in a Finance business rule are when a formula requires extensive cross-dimensional dependencies and it is not clear on which member an equivalent Member Formula should be written or when the formula requires complex sequential logic, variables, or conditional statements that affect multiple dependent calculations. Also, Finance business rules may be used when an application requires custom algorithms for currency rranslation, share, or intercompany eliminations.

> **TIP:** Due to the sophisticated built-in translation and consolidation algorithms, most applications require only Member Formulas for the Data Unit Calculation Sequence (DUCS), for example, chart logic. Custom business rules for translation, share, and intercompany eliminations are not typically needed.

> **NOTE:** Due to the way OneStream stores its data, Decimal should always be used instead of Double or Integer when declaring variables that return a number within a business rule or Member Formula.

```
Dim myNumber as Decimal = Api.Data.GetDataCell("A#Sales - A#CostOfSales").CellAmount
```

See Business Rules.

Formula Types

These types of formulas can be applied to dimensions members or members in a business rule file:

- **Dynamic Cell Calculation (Dynamic Calc)**: An in-memory calculation that runs on demand when a cell containing a dynamic member formula is requested. A Dynamic Calc formula computes a value for a single cell and runs whenever the cell needs to be displayed without storing the result.
- **Stored Calculation**: A persisted calculation that runs as part of the Data Unit Calculation Sequence (DUCS). With these formulas, you can calculate many cells simultaneously, for example, data buffer.

Performance Considerations

Dynamic Cell Calculation (Dynamic Calc): Dynamic cell calculations enhance the consolidation process because the amount is calculated when requested for display and is not written to the database. For reporting, performance may be impacted because data is calculated on demand. Dynamic calculations are usually used for ratio or percentage calculations.

Stored Calculation: Consolidation performance is directly impacted by the volume and complexity of stored calculations. Carefully consider each stored calculation since one poorly written rule can cause large amounts of data to be written to the cube, negatively impacting consolidation performance. If you use many Member Formulas or if data volumes are not considered in Member Formulas, over 100,000 stored numbers may be generated from just 1,000 initially loaded numbers. The quantity of stored numbers is a critical factor to consider to optimize consolidation performance.

Reference and Aggregation Considerations

Dynamic Cell Calculations: Dynamic cell calculations can reference other dynamic cell calculations and stored calculations but should not be used by stored calculations because they do not naturally aggregate parent members. For example, if a parent member in an Account, Flow, or User Defined dimension has a child member calculated by a dynamic calculation formula, the parent excludes the child member from the aggregated amount. Achieve aggregation by writing another dynamic calculation on the parent member.

Stored Calculations: Stored calculations can reference other stored calculations, and parent members aggregate naturally.

> **TIP:** You do not need to write a stored calculation to add or subtract individual members. Instead, create an alternative member hierarchy and use the Aggregation Weight property, set to -1.0 to negate. This aggregated value is dynamic and supports drill-down.

Stored Calculation Evaluation Tools: You can access detailed process logging information. Run **Force Consolidate With Logging** and select **Task Activity** to analyze steps in the consolidation process to identify bottlenecks or errors for performance optimizations. See the following examples:

- Calculate with Logging
- Translate with Logging
- Consolidate with Logging

Formula Calculation Threshold Monitoring: Calculation time threshold values can be set in the Application Server Configuration File using the NumSecondsBeforeLoggingSlowFormulas setting to log formulas exceeding the specified threshold time.

Formula Execution

This section defines the types of calculations and calculation sequences used during different analytic processing routines.

Cube Processing

Calculation: This runs the standard calculation sequence for a single Data Unit. A Data Unit refers to a group of data cells for a specific Cube, Entity, Parent, Consolidation, Scenario, and Time member. See "Data Units" in the Workflow Guide for more details on Data Units. Except for dynamic cell calculations, all Member Formulas are written to run as part of a Data Unit Calculation Sequence.

Translation: This runs a currency translation that occurs when the data for an Entity's local currency needs to be translated to a foreign currency. The translation step runs after the system has run the Data Unit Calculation Sequence for an Entity's local currency. After this is completed, the default translation algorithms use foreign exchange (FX) rates to generate and store a corresponding translated Data Unit. The Data Unit Calculation Sequence on the translated Data Unit is run to produce the final translated amounts.

Consolidation: The analytic engine provides pre-built financial intelligence through a statutory Consolidation dimension that defines a sequence of Data Unit calculations and aggregations that include currency translations, parent-level adjustments, complex ownership computations, and intercompany eliminations. See Consolidation in the Financial Model Guide.

Dynamic Cell Calculation: Dynamic cell calculations are a special type of Member Formula for Account, Flow, or User Defined members. They are used to generate amounts for a member on the fly (the results are calculated on-demand not stored). Dynamic cell calculations are often used for metric accounts, for example, ratios involving other accounts, and are appropriate when the result of the dynamic calculation is not needed as the source number for another stored calculation. The use of dynamic cell calculations can result in improved consolidation performance because they do not generate stored numbers and are typically only run when a number needs to be displayed, not during the consolidation process.

Data Units and Formula Process

OneStream runs formulas at a specific unit of work call a Data Unit. This section details the order and combination of logical processes that run for a Data Unit. See Using Data Units in the Workflow Guide.

Guidelines on Formula Passes: There are 16 formula passes and one dynamic calculation formula type available on each member. Formulas that do not depend on one another can exist in the same formula pass and will be calculated in parallel using the OneStream advanced parallel formula processing engine. As a basic guideline, customers should think about organizing formulas by account and collection type across the available formula passes using the following examples.

Formula Pass 1 - 8: Trial Balance

Formula Pass 5 - 7: Translation logic related to Cash Flow

Formula Pass 8 or 9: Balance Account and CTA account

Formula Pass 9 - 16: Non-trial Balance

All formulas in a pass are processed at the same time, so they cannot have dependencies on one another.

Calculation Sequence of a Single Data Unit: The following items detail the specific list of tasks run for each Data Unit calculation process. As an example, the following steps are run for a single Data Unit when a user selects Calculate for a single Entity, Scenario, and Time period.

Data Unit Calculation Sequences (DUCS)

1. Clear previously calculated data for the Data Unit.
2. Run the Scenario Member Formula, which is typically used for seeding Scenario data from another Scenario or from a prior year.
3. Run reverse translations by calculating Flow members from other Alternate Currency Input Flow members. This is part of the built-in ability for an Entity to accept input using multiple currencies.
4. Run business rules (1 and 2). Up to 8 business rule files can be attached to each cube.
5. Run formula passes (1–4) for the Account Dimension members on the cube, then Flow members, and then User Defined members. The formula pass is specified using the Formula Type property for each member in the Dimension Library.
    - Account, Flow, UD1, UD2, … UD8 (run Member Formula)
6. Run business rules (3 and 4).
7. Run formula passes (5–8).
    - Account, Flow, UD1, UD2, … UD8 (run Member Formula)
8. Run business rules (5 and 6).
9. Run formula passes (9–12).
    - Account, Flow, UD1, UD2, … UD8 (run Member Formula)
10. Run business rules (7 and 8).
11. Run formula passes (13–16).
    - Account, Flow, UD1, UD2, … UD8 (run Member Formula)

Every time a data cell is written to the database, information with the stored cell about how it was stored, for example,manually entered, calculated, or consolidated, is included. If a number was calculated and stored as a result of a formula, it will always get cleared regardless of the metadata settings. The AllowInput property specifies whether a cell can be written to and therefore if a formula stored a number. There would be information if the cell was calculated. If AllowInput is set to True and a new number is typed over the same cell, it is stored as manually entered instead of a calculated cell.

Formula for Statutory Consolidation Sequence

The consolidation process is run for a hierarchy of Entities.See Consolidation in the Financial Model Guide.

Formula Level Controls

In addition to the Cube-level settings described, VB.NET If statements are often used in a formula to run only for certain types of Data Units. The most common usage is to contain checks for base-level Entities and local currencies. The following If statement will cause the formula to run only for base-level Entities and the default currency of the Entity. This is often used when the results of the calculation in the base-level Entity is intended to be consolidated to parent-level Entities.

```
If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyForEntity())) Then
End If
```

Writing Formulas

The following section provides information on writing formulas.

Writing Dynamic Cell Calculations

When writing a formula for a Dynamic Calc Account, Flow, or User Defined member in either a Member Formula or a business rule (using FinanceFunctionType.GetDataCell), the goal is to return an amount for a single data cell. In that case, the system knows the full 18 dimensions of the data cell it needs to display. Therefore, use any of the API methods that refer to a specific Account, Flow, Intercompany, or User Defined dimension.

For example, a user selected a Cube View to view some numbers. When a Cube View displays a cell, it determines whether the cell requires a custom formula. If it does, the Cube View initializes the API so that the formula has access to all 18 dimensions. The Member Formula then displays an amount on the Cube View.

Dynamic Calc Formula Examples

The simplest Dynamic Calc formula is to return a constant, which is the same number regardless of the intersection. For example, select an Account (or a Flow or User Defined member) in the Dimension Library, set the Formula Type property to DynamicCalc, and then enter the following line in the member Formula property for the default scenario type and default time member. After typing the code, select the Compile button on the Formula Editor toolbar to make sure the VB.NET syntax is correct. Select the OK button to close the dialog boxes, and then select the Save button.

```
Return 123.4
```

If a Cube View is used to display the result of the previous calculation for the Dynamic Calc account for any numeric intersection (any Entity, Scenario, Time, or UD for View set to YTD), the Cube View will always display 123.4. If the account value is displayed for any text intersection (any Entity, Scenario, Time, or UD for View set to Annotation), the Cube View will display an empty cell because the formula returned a numeric value, not a text value. To make the formula work for both numeric and text View members, use an If statement to check which type of View member is currently being requested. Then, return a text value surrounded by double quotation marks when the View member is an annotation type. After saving, run the Cube View using multiple View members (for example, YTD, Periodic, Annotation, or Assumptions). The corresponding cell will display either of the constants specified.

```
objViewMember As ViewMember = ViewMember.GetItem(api.Pov.View.MemberId)
If objViewMember.IsAnnotationType Then
   Return "My first OneStream Member Formula"
Else
    Return 123.4
End If
```

Return Types for Dynamic Calc Formulas

As shown in the previous example, the return value for a numeric data cell can be a number, for exmaple, a constant or a decimal variable, and the return type for a text intersection can be a text value, for example, words within double quotation marks or a string variable. However, when those types are returned, the underlying engine converts them to a DataCell or a DataCellEx object automatically. Therefore, if a DataCell or DataCellEx object was already contained by calling `api.Data.GetDataCell`, then it is recommended to return the full object instead of just the number contained within the DataCell object, which could be accessed using objDataCell.CellAmount. To specify status, for example, whether the cell is NoData, return a DataCell or DataCellEx and use its CellStatus property for those types of settings.

A DataCell object is a wrapper for a DataCellPk object that defines the 18 dimension intersection, a decimal accessed using the CellAmount property to store the number, and a CellStatus containing other information about the cell, for example, NoData and Invalid Status.

A DataCellEx object is a wrapper for a DataCell object and a text property called DataCellAnnotation, which is used for setting a string for an Annotation type View member. It also contains some additional properties for CurrencyId and AccountTypeId filled in and can be ignored when creating a DataCellEx object in a Dynamic Calc Member Formula.

The following example accomplishes the exact same result as the previous example, except this uses DataCell and DataCellEx objects to illustrate what to do if a return value containing cell status is needed.

```
Dim objViewMember As ViewMember = ViewMember.GetItem(api.Pov.View.MemberId)
If objViewMember.IsAnnotationType Then
   Dim objDataCellEx As DataCellEx = New DataCellEx()
   objDataCellEx.DataCell.CellStatus = New DataCellStatus(True)
   objDataCellEx.DataCellAnnotation = "My first OneStream Member Formula"
   Return objDataCellEx
Else
   Dim objDataCell As DataCell = New DataCell()
   objDataCell.CellStatus = New DataCellStatus(True)
   objDataCell.CellAmount = 123.5
   Return objDataCell
End If
```

Using Math in Dynamic Calc Formulas

The OneStream `api.Data.GetDataCell` function supports a powerful script parser that allows math equations to be written and operate on one or more data cells or constant amounts to calculate the values for a new data cell. For example, if a Dynamic Calc account needs to display Cash plus AcctsRec increased by 10%, this one-line Member Formula can do it.

```
Return api.Data.GetDataCell("(A#Cash + A#AcctsRec) * 1.10")
```

Notice the `api.Data.GetDataCell` function accepts one string parameter in double quotation marks, which represents a member script equation. Each operand of the equation, for example,, A#Cash, takes the unspecified dimensions using the data cell currently being calculated. Therefore, every operand points to a specific data cell identified by using an 18 dimension intersection.

The following examples show the current data cell being calculated and the first operand even though "A#Cash" was specified.

Current Data Cell Being Calculated

```
"E#CT:C#USD:S#Actual:#2013M1:V#YTD:A#NewAccount:F#None:O#Top:I#None:U1#None, …",
```

First Operand

```
"E#CT:C#USD:S#Actual:#2013M1:V#YTD:A#Cash:F#None:O#Top:I#None:U1#None, …"
```

When writing Dynamic Calc formulas, specify any or all the 18 dimensions if necessary.

```
(e.g., "A#Cash:U1#AllProducts + A#AcctsRec:U1#None")
```

Division in Dynamic Calc Formulas

Using member script equations in `api.Data.GetDataCell` as shown in the previous example is powerful for Dynamic Calc formulas, but there are occasions when all the math cannot be performed reliably in one line of script. For example, use this formula to divide by a data cell when the data cell being used as the denominator is zero or NoData, which means that a number was never entered. Since dividing by zero results in infinity and is not a valid operation for computers, the formula needs some extra checking. To illustrate additional concepts, here are different examples of performing division in a Dynamic Calc formula.

This first example uses the division operator (/) to calculate a data cell from one account divided by a data cell from another account. If the denominator (A#AcctsRec) is zero or NoData, it will automatically return a very large number (for example, 9999999999999999999.0) as the result. This is because dividing by zero in mathematics results in infinity, and the large number to approximate infinity allows subsequent functions or math operators that refer to the result to continue to be processed is used.

```
Return api.Data.GetDataCell("A#Cash / A#AcctsRec")
```

Although an extremely large number is the best mathematical approximation for infinity, it is typically not what administrators want to display in their financial system when source numbers are not available. A Divide function that produces a NoData cell if either the numerator or the denominator is NoData is available.

```
Return api.Data.GetDataCell("Divide(A#Cash, A#AcctsRec)")
```

The built-in Divide function is typically used when performing division in Dynamic Calc formulas. However, for completeness and to provide some insight about how to create more complex formulas, the following example shows how to implement a formula that performs safe division. Notice that the Member Formula is performing division using two DataCell objects to create a resulting DataCell (Return numeratorDataCell / denominatorDataCell). This is a powerful capability that allows any type of math to be performed using any number of DataCell objects.

```
Dim numeratorDataCell As DataCell = api.Data.GetDataCell("A#Cash")
Dim denominatorDataCell As DataCell = api.Data.GetDataCell("A#AcctsRec")
If ((Not numeratorDataCell Is Nothing) And (Not denominatorDataCell Is Nothing)) Then
   If ((Not numeratorDataCell.CellStatus.IsNoData) And _
        (Not denominatorDataCell.CellStatus.IsNoData) And _
        (denominatorDataCell.CellAmount <> 0.0)) Then
     Return numeratorDataCell / denominatorDataCell
   End If
End If
Return Nothing
```

> **TIP:** When using VB.NET, use the underscore character at the end of a line to continue a statement on the next line. For example, refer to the underscore after And in the previous code. This was done because the full statement did not fit on one line in this document. However, in the Member Formula editor, the If statement would be kept on one long line.

Days Sales Outstanding

Days sales outstanding (DSO) is a common formula that is a required calculation for many applications. Consequently, OneStream has provided a pre-built function to encapsulate the logic required for this function.

```
Return api.Functions.GetDSODataCell("AcctsRec", "Sales")
```

The following example demonstrates a possible customized version of the DSO calculation.

```
Dim numDaysSum as Integer = 0
Dim currTimeId as Integer = api.Pov.Time.MemberPk.MemberId
Dim acctsRec as decimal = api.Data.GetDataCell("A#AcctsRec - A#SaleTax").CellAmount
If (acctsRec > 0.0) Then
    Dim salesSum as Decimal = 0.0
    Dim numPeriodsInYear = api.Time.GetNumPeriodsInYearForSameFrequency(currTimeId)
    For (numPeriodsToSubtract As Integer = 0 To numPeriodsInYear)
        Dim timeId as Integer
        If numPeriodsToSubtract = 0 Then
           timeId = currTimeId
        Else
           timeId = api.Time.AddTimePeriods(-1 * numPeriodsToSubtract, True)
        End If
        Dim timeName As String = api.Time.GetNameFromId(timeId)
        Dim numDaysInTimePeriod As Integer = api.Time.GetNumDaysInTimePeriod(timeId)

        Dim MemberscriptBldr = New MemberscriptBuilder("A#Sales:V#Periodic").SetTime
(timeName)
        Dim Memberscript As String = MemberscriptBldr.GetMemberscript()
        Dim salesForTimePeriod as Decimal = api.Data.GetDataCell(Memberscript).CellAmount

        If (salesForTimePeriod + salesSum >= acctsRec) Then
            Dim ratio As Decimal = (acctsRec - salesSum) / salesForTimePeriod
            numDaysSum = numDaysSum + (ratio * numDaysInTimePeriod)
            'We are done
```

```
            Exit For
        Else
            numDaysSum = numDaysSum + numDaysInTimePeriod
            salesSum = salesSum + salesForTimePeriod
        End If
    Next
End If
Return api.Data.CreateDataCellObject(numDaysSum, False, False)
```

Dynamic Simulation of Consolidation: GetEntityAggregationDataCell

Use a function called GetEntityAggregationDataCell for pseudo/approximate consolidation of a data cell with a Cube View when requested for display. The intention is not to produce a value that would tie to a formally consolidated number if custom business rules for ownership or translation exist for such a financial model, but this on-the-fly dynamic value is presented to the user instantly as a convenience, typically during data entry.

This function employs standard currency translation using the Entity in the local currency in the cell POV as the source and a parent Entity target and standard percent consolidation on the Relationship properties to calculate share. It does not take intercompany elimination into account unless that value was already consolidated and stored.

```
api.Functions.GetEntityAggregationDataCell(memberScript as string,
 Optional useStoredAmountsWhereCalcStatusIsOK as Boolean, Optional
 fxRateTypeRevenueExpOverride as String, Optional
fxRuleTypeRevenueExpOverride as FxRuleType, Optional
 fxRateTypeAssetLiabOverride as String, Optional
fxRuleTypeAssetLiabOverride as FxRuleType)
```

When useStoredAmountsWhereCalcStatusIsOK is set to True (the default setting), the algorithm checks if CalcStatus is OK for the intersection and uses numbers that are already calculated, translated, ownership share calculated, eliminated, consolidated, and stored. This setting is also useful when displaying multiple translated currencies for consolidated parent Entities dynamically. If the parent Entity Local amount is OK and fully consolidated, the only dynamic calculation would be the last translation step.

The last four optional settings let you specify alternate named FX rate types (for example, “AverageRate”) and FX rule types (FxRuleType.Direct or FxRuleType.Periodic) to perform what-if simulations.

If any of these four are specified, useStoredAmountsWhereCalcStatusIsOK is ignored and is treated as False because consolidated amounts would not have been stored using the alternate FX rates.

The following example usesa Dynamic Calc member (for example, UD8):

```
Return api.Functions.GetEntityAggregationDataCell("A#NetIncome:S#Budget:UD8#None")
Return api.Functions.GetEntityAggregationDataCell("UD8#None", True, “AverageRate”,
FxRuleType.Periodic)
Return api.Functions.GetEntityAggregationDataCell("UD8#None", True, “AverageRate”,
FxRuleType.Periodic,
    “HistoricalRate”, FxRuleType.Direct)
```

The following example usesGetEntityAggregationDataCell to enter inputs and instantly see aggregated results after saving a form.

Create a UD8 member named EntityAggregation and set its formula type to Dynamic Calc. Use this as the UD8 Member Formula:

```
Return api.Functions.GetEntityAggregationDataCell("UD8#None")
```

Create a Cube View

1. Set Cube View POV tab to have Scenario set to Budget or the value on the Form and UD8 member of EntityAggregation.
2. Define rows that display a hierarchy of entities.
3. Define columns that display a few key accounts. Include a base-level account that supports input for both UD8#EntityAggregation and for input.

   > **Example:** A#Sales, A#Sales:UD8#None:Name("Sales Input")

4. Change Cube View settings for General Settings / Common to set Can Modify Data and Can Calculate to True.
5. Associate this Cube View with a form template that is assigned to a Workflow Profile.

The user enters a number on the form in the account that accepts input and selects Save. The user will see the dynamically aggregated entity results without having to run a consolidation.

Dynamic Calc Member Formula

This formula returns the name and description of the Entity UD1 default member. Other UD8 formulas can be created, for example, for the Entity UD2 default. After adding the UD8 member, the Entity default UD1 member name can be displayed using XFGetCell or in a Cube View by accessing the cell: E#MyEntity:V#Annotation:U8#EntityDefaultUD2Name.

This assumes that the UD8 member with the formula is named EntityDefaultUD2Name. If XFGetCell is being used in Excel, use None or any valid member for all the other dimensions. This displays the member name of the Entity default UD1 member using the Annotation View members.

```
If api.View.IsAnnotationType(api.Pov.View.MemberId) Then
   Dim text As String = String.Empty
   Dim udId As Integer = api.Entity.GetDefaultUDMemberId(api.Pov.Entity.MemberId,
DimType.UD1.Id)
   If udId <> DimConstants.Unknown Then
Dim udMember As Member = api.Members.GetMember(DimType.UD1.Id, udId)
If Not udMember Is Nothing Then
text = udMember.NameAndDescription
End If
    End If
    Return text
End If
'If this is a numeric View Member (e.g., Periodic, YTD), display the number from the U8#None
Member.
Return api.Data.GetDataCell("U8#None")
```

Annotations on Dynamic Calc Members

To support the ability to calculate text or pull it from an external system to display in a Cube View, DynamicCalc or DynamicCalcTextInput formulas can be used. If the user wants the formula to calculate the annotations, use a DynamicCalc formula to display text, for example, Pass and Fail.

The DynamicCalcTextInput Formula Type works the same as a DynamicCalc formula, but it allows users to input annotations on Cube View cells without having to use the Data Attachment dialog box. When this formula is used, the user can make annotations on Dynamic Calc members following the same method as a non-calculated member.

Writing Stored Calculations

When writing a Member Formula or a business rule for a stored calculation, the new calculated numbers are being determined to store for that cube, entity, parent, cons, scenario, and time combination, that is, a Data Unit.

Return is never seen in a Member Formula for formula pass. Instead of being returned, many numbers are being calculated and stored. When running a calculation, translation, or consolidation, the Member Formula calls for an entire Data Unit. It does not tell with which Account, Flow, or User Defined the numbers are being saved. This is the responsibility of the user. Initially, this may be confusing because Member Formulas are often written in an account's Formula property, which could cause administrators to believe it will only allow that specific Member Formula to write to that specific account. However, putting a Member Formula in an account's Formula property is only for organizational purposes. When it calls that formula, it is currently calculating a Data Unit and it will initialize the API with only the Data Unit dimensions.

Basic Stored Formula Examples

The formula examples in this section demonstrate how to calculate basic stored values driven by formulas consisting of OneStream member script expressions.

Copy Data from another Account

The following formula would be implemented as a Member Formula on the Sales1 account. It is run as part of the DUCS during the formula pass that was specified in the account's Formula Type setting.

```
api.Data.Calculate("A#Sales1 = A#Sales2”)
```

Stored formula passes use data buffer math, not the data cell math that occurs for the single cell Dynamic Calc formulas. Stored formulas are multi-dimensional. For example, the formula is run for an entire Data Unit, for example, Location1 Entity, USD Consolidation Member, Actual Scenario, January 2013 time period. That Data Unit is a portion of a cube where the UD1 dimension could contain 1,000 products to keep track of sales by product. Therefore, the data for the Sales2 account could contain a separate number for every product (that is, UD1), or if the Location1 Entity only sells some of the products, there might be 200 numbers for Sales2 and the other 800 products for Sales2 are NoData. That set of 200 numbers is called a data buffer. Data buffers can get much larger and more complicated when multiple dimensions are used for detailed analyses. However, since the same concepts still apply, it is easier to think about a smaller set of dimensions as in this example.

The formula A#Sales1 = A#Sales2 is equivalent to: Take the 200 numbers stored in the Sales2 data buffer and copy them to a new data buffer, but change the account to Sales1, then store the new Sales1 data buffer in the database. That one-line formula calculated and stored an additional 200 numbers that did not exist before the formula was run.

The following formula reads the data buffer for the Sales2 account (200 numbers) and then adds 50.0 to each of those 200 numbers to create a new data buffer that also contains 200 numbers. The account for each of the 200 numbers in the new data buffer is changed to Sales1, and it is then stored in the database.

```
api.Data.Calculate("A#Sales1 = A#Sales2 + 50.0”)
```

The newly modified formula in the following example uses three accounts. That formula reads the data buffer for the Sales2 account (200 numbers) and then reads the data buffer for the Sales3 account. For example, the Sales3 account contains 100 numbers broken out by product in UD1, and 25 of those 100 numbers use the same UD1 members as some of the numbers from Sales2. The other 75 Sales3 numbers are for other products not used by the Sales2 account. OneStream automatically combines the numbers from Sales2 and Sales3 and adds the Sales2 and Sales3 numbers that share a common intersection and adds the additional non-common intersections. The result is a new data buffer containing 275 numbers stored in the database for the Sales1 account.

```
api.Data.Calculate("A#Sales1 = A#Sales2 + A#Sales3”)
```

Copy Data from Another Scenario

The following formula would be implemented as a Member Formula on the Forecast scenario. It copies all the data from the Actual scenario and stores the results in the Forecast scenario. The If statement in this example causes the data to be copied only for base-level entities and their local currency. This is because the example application wants the numbers for parent entities and foreign currencies to be determined using the OneStream consolidation and translation algorithms, not by copying directly from another scenario. Limit the formula to only copy data for certain dimension members by adding specific members in the formula member script. For example, if the Forecast scenario needs to copy only the Import members, add O#Import to both sides of the equation. If the Forecast scenario needs to start with the sum of the Actual scenario import plus forms plus AdjInput data and copy that into the Forecast scenario O#Import Member, then use “S#Forecast:O#Import = S#Actual:O#Top”. When writing stored formulas, any Data Unit dimension not explicitly specified uses the dimension member for the Data Unit currently being calculated. For any Account, Flow, Origin, IC, or User Defined dimensions not explicitly specified, OneStream will use #All, which is the syntax that represents all existing data for that dimension.

```
If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyForEntity()))
Then
    api.Data.Calculate("S#Forecast = S#Actual")
End If
```

To copy data from another cube or scenario that uses different Dimensionality, see the following example of the scenario formula:

```
'Convert dimensionality
Dim destinationInfo As ExpressionDestinationInfo = api.Data.GetExpressionDestinationInfo("")
Dim sourceDataBuffer As DataBuffer = api.Data.GetDataBuffer
(DataApiScriptMethodType.Calculate,
"Cb#AnotherCube:S#AnotherScenario",destinationInfo)
Dim convertedDataBuffer As DataBuffer = api.Data.ConvertDataBufferExtendedMembers
("AnotherCube",
"AnotherScenario", sourceDataBuffer)
api.Data.SetDataBuffer(convertedDataBuffer, destinationInfo)
```

To drill down on this formula, use the following example in the scenario formula for calculation drill down setting:

```
If api.Pov.Cube.Name.XFEqualsIgnoreCase("TheDestCube") Then
    Dim result As New DrillDownFormulaResult()
    result.Explanation = "Pseudo-formula:
Cb#TheDestCube:S#TheDestScenario=Cb#AnotherCube:S#AnotherScenario”
    result.SourceDataCells.Add("Cb#AnotherCube:S#AnotherScenario")
    Return result
End If
Return Nothing
```

Out Of Balance

The following formula would be implemented as a Member Formula on the balance account. It stores the difference of two other accounts.

```
If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyForEntity())) Then
    api.Data.Calculate("A#Balance = A#2899 - A#5999")
End If
```

CTA Account Formula Example

This is essentially the same formula as Out Of Balance, but only runs on translated Data Units.

```
If ((Not api.Entity.HasChildren()) And (api.Cons.IsForeignCurrencyForEntity())) Then
    api.Data.Calculate("A#CTA = A#2899 - A#5999")
End if
```

Reading a Specific Data Cell in a Stored Formula

Stored formulas are run for an entire Data Unit, so unlike Dynamic Calc formulas, there is no context about a specific View, Account, Flow, Origin, IC, or User Defined member within the Data Unit. Therefore, to read the value for a Data Cell using `api.Data.GetDataCell`, a member needs to be explicitly specified for all non-Data Unit dimensions. This is different than Dynamic Calc Member Formulas where the default setting for every dimension member comes from the Data Cell currently being displayed.

Therefore, the following example is incorrect when trying to read a specific data cell inside a stored formula:

```
Dim objDataCell As DataCell = api.Data.GetDataCell("A#Cash")
```

Instead, all non-Data Unit dimensions need to be specified. If one or more of those dimensions needs to be based on the other data stored in the Data Unit, then use Eval. Otherwise, the formula will look like this example:

```
Dim objDataCell As DataCell = api.Data.GetDataCell("V#YTD:A#Cash:F#None:O#Import:I#None
    :U1#None:U2#None:U3#None:U4#None:U5#None:U6#None:U7#None:U8#None")
```

The previous syntax is accurate but creating that long string for many dimensions is tedious and error prone especially when using functions to determine what the member names should be and then concatenating multiple strings. Instead, it is recommended to use the MemberscriptBuilder class when creating member scripts.

```
Dim MemberscriptBldr = New MemberscriptBuilder("V#YTD:A#Cash")
MemberscriptBldr.SetFlow("None").SetOrigin("Import").SetIC("None").SetAllUDsToNone()
Dim Memberscript As String = MemberscriptBldr.GetMemberscript()
Dim objDataCell As DataCell = api.Data.GetDataCell(Memberscript)
```

Advanced Stored Formulas using Eval

As described in the previous examples, OneStream data buffer math is extremely powerful and can process hundreds or thousands of numbers with just a simple equation. Without data buffer math or an equivalent scripting capability, a large multi-dimensional financial application would not be feasible because every intersection would need to be considered separately. There is a consequence when processing data using data buffers instead of individual data cells. Additional capabilities are needed when wanting to perform math differently based on the individual data cell amounts.

Fortunately, OneStream accommodates that pattern using Eval. When implementing `api.Data.Calculate` functions, Eval has an advanced capability that provides the ability to get at the individual data cells in any Data Unit created while processing an `api.Data.Calculate` script. It even allows Eval() to be wrapped around a subset of the formula's math to evaluate the data buffer that was just created by running that math.

As an example for Eval, start with this formula:

```
api.Data.Calculate("A#Sales1 = A#Sales2”)
```

The Sales2 numbers need to be copied to Sales1 for green products. In this fictitious example, there is a special tax situation for green products and the sales numbers for those products need to be isolated into the special Sales1 account. The application uses the UD1 Member Text1 property to track which products are green.

The first thought might be to do something like the following example, which is incorrect:

```
Dim ud1Id As Integer = api.Pov.UD1.MemberId
Dim text1 As String = api.UD1.Text(ud1Id, 1)
If (text1.Equals("green", StringComparison.InvariantCultureIgnoreCase)) Then
    api.Data.Calculate("A#Sales1 = A#Sales2")
End If
```

The previous example would not work because stored formulas are run for an entire Data Unit. A Data Unit represents all data for a Cube, Scenario, Entity, Parent, Cons, and Time member. Since there is no single product (that is , UD1 member) for the Data Unit currently being calculated, the first line in the previous example does not make sense. A Data Unit cannot be asked what the UD1 MemberId is because a Data Unit has data for multiple UD1 members (200 different products in the Sales2 example).

The solution is using Eval to evaluate the individual data cells in a data buffer. Put the Eval keyword around any portion of the `api.Data.Calculate` function including math statements. After OneStream reads or calculates the data buffer defined within the Eval statement, it runs the Eval function to give the opportunity to filter the list of data cells in the data buffer or to completely change the list of data cells in the data buffer. After the Eval function is completed, OneStream uses the modified data buffer to perform the remaining part of the `api.Data.Calculate` function.

The example formula needs to be modified by adding the Eval keyword around the A#Sales2 data buffer. A helper function, typically the name OnEvalDataBuffer, needs to be implemented allowing an inspection, filter, or change to the data cells in the data buffer. The helper function loops over each of the Sales2 data cells (200 in this example). If the data cell's UD1 Text1 setting says green, add that data cell to a new list of result cells. Otherwise, ignore the data cell causing it to be skipped. The result is a new modified data buffer containing only the data cells for green products (that is , fewer than 200 data cells).

```
api.Data.Calculate("A#Sales1 = Eval(A#Sales2)", AddressOf OnEvalDataBuffer)

Private Sub OnEvalDataBuffer(ByVal api As FinanceRulesApi, ByVal evalName As String, _ByVal
eventArgs As EvalDataBufferEventArgs)
    'Filter to list of Sales2 source numbers to only include numbers
    'for "green" Products using each dataCell's UD1 Text1 setting.
    'The final list of resultCells is what will be assigned to Sales1 by api.Data.Calculate.
    Dim resultCells As New Dictionary(Of DataBufferCellPk, DataBufferCell)
    For Each sourceCell As DataBufferCell In
eventArgs.DataBuffer1.DataBufferCells.Values
        If (Not sourceCell.CellStatus.IsNoData) Then
            Dim ud1Id As Integer = sourceCell.DataBufferCellPk.UD1Id
            Dim text1 As String = api.UD1.Text(ud1Id, 1)
            If (text1.Equals("green", StringComparison.InvariantCultureIgnoreCase)) Then
            'Add this dataCell to the new list.
            resultCells(sourceCell.DataBufferCellPk) = sourceCell
            End If
        End If
    Next
    'Assign the new list of DataCells to the result.
    eventArgs.DataBufferResult.DataBufferCells = resultCells
End Sub
```

NOTE: If using Eval for NoData and ZeroCells, refer to the Remove Functions in Formulas section for alternative performance enhancing solutions.

Advanced Stored Formulas Using GetDataBuffer and SetDataBuffer

Most of the advanced stored formulas that need to process multiple data cells should use Eval. Eval allows the user to get at the individual data cells in any Data Unit while processing the member script in an `api.Data.Calculate` function. However, in some rare occasions, an appropriate member script may not be defined for the `api.Data.Calculate` function because multiple data cells that seem completely unrelated to each other are being processed and none of the dimension members are constant.

For those rare situations, use the GetDataBuffer and SetDataBuffer functions directly. GetDataBuffer and SetDataBuffer are more fundamental then Eval. They are part of the internal implementation of the Eval functionality. They allow the user to read some numbers using a member script, process or modify each cell in the result, and then save the changes.

In the following example, the UD2 and UD3 dimensions are being used to analyze data based on each UD1 member default settings for UD2 and UD3. All data is initially loaded to the U2#Input:U3#Input members, but that loaded data needs to be copied to the U2#DefaultUD1:U3#DefaultUD1 members. GetDataBuffer needs to be used in order to read the loaded data because the destination UD2 and UD3 members can be different for every data cell based on its UD1 member settings. Loop over each data cell and use the UD1 member to get its default UD2 and default UD3 settings. Then, change the UD2 and UD3 member IDs for the data cell in the data buffer. Finally, after looping, call SetDataBuffer to save the new numbers.

When using `api.Data.Calculate` functions with or without Eval, it is important to know to which member a formula is being attached. For example, if the formula starts with `api.Data.Calculate("A#Sales1 = …")`, put the formula in the Sales1 account member's Formula setting. However, the formula in this example is not writing to a specific member. Every data cell being saved is possibly written to a different UD2 and UD3 member.

Technically speaking, the formula can be put in any member's Formula property even a seemingly unrelated member. If the formula pass is set correctly, the formula runs before any other dependent formulas. Assigning stored formulas to members is for organizational purposes only. The member does not restrict what the formula can do. However, if a formula is attached to an unrelated member, it will make the application difficult to maintain and understand. Therefore, decide to attach a formula like this to the scenario's member, which means this formula needs to be processed before most other formula passes. If there are numerous scenarios, this formula should be put in a business rule file, and the business rule file should be added to the Cubes. This can be done under Application tab > Cube > Cubes.
See the completed formula using GetDataBuffer and SetDataBuffer in the following example:

```
'Copy all "U2#Input:U3#Input" numbers for this dataUnit to the corresponding UD1 default
Members for UD2 and UD3.
Dim destinationInfo As ExpressionDestinationInfo = api.Data.GetExpressionDestinationInfo("")
Dim sourceDataBuffer As DataBuffer = api.Data.GetDataBuffer
(DataApiScriptMethodType.Calculate, _"U2#Input:U3#Input", destinationInfo)
If Not sourceDataBuffer Is Nothing Then
    Dim resultDataBuffer As DataBuffer = New DataBuffer()
    For Each cell As DataBufferCell In sourceDataBuffer.DataBufferCells.Values
        If (Not cell.CellStatus.IsNoData) Then
            Dim ud1Id As Integer = cell.DataBufferCellPk.UD1Id
            cell.DataBufferCellPk.UD2Id = api.UD1.GetDefaultUDMemberId(ud1Id,
DimType.UD2.Id)
            cell.DataBufferCellPk.UD3Id = api.UD1.GetDefaultUDMemberId(ud1Id,
DimType.UD3.Id)
            resultDataBuffer.SetCell(api.DbConnApp.SI, cell)
        End If
    Next
    api.Data.SetDataBuffer(resultDataBuffer, destinationInfo)
End If
```

Comparing Two DataBuffers Using Eval2

Eval2 is the same as Eval except two members scripts are specified to define two data buffers. When finished, the OnEvalDataBuffer function can compare all the numbers in the two data buffers. The following example provided evaluates two separate Flow members to see if they contain the same value.

```
'Use Eval2 to compare the numbers in 2 DataBuffers and store a value in F#USDOverride_Check
for each pair of numbers that don't match.
If ((api.Cons.IsLocalCurrencyForEntity()) And (Not api.Entity.HasChildren())) Then
    api.Data.Calculate("V#YTD:F#USDOverride_Check:O#Forms = Eval2(V#YTD:F#Local_Change_
Validation:O#Top,V#YTD:F#USDOverride_Change_Validation:O#Top)", AddressOf OnEvalDataBuffer)
End If

Private Sub OnEvalDataBuffer(ByVal api As FinanceRulesApi, ByVal evalName As String, ByVal
eventArgs As EvalDataBufferEventArgs)

'This function compares the numbers in 2 DataBuffers and returns a new DataBuffer that has a
value for each pair of numbers that don't match.
    eventArgs.DataBufferResult.DataBufferCells.Clear()
        If Not eventArgs.DataBuffer1 Is Nothing And Not eventArgs.DataBuffer2
        Is Nothing Then
'For each cell in DataBuffer1, try to find a number for the same intersection in
DataBuffer2.
 For Each cell1 As DataBufferCell In eventArgs.DataBuffer1.DataBufferCells.Values
            If (Not cell1.CellStatus.IsNoData) Then
            Dim cell2 As DataBufferCell = eventArgs.DataBuffer2.GetCell
(api.SI,cell1.DataBufferCellPk)
            If Not cell2 Is Nothing Then
                If (cell1.CellAmount <> cell2.CellAmount) Then
                'Since the numbers don't match, add a cell to the result DataBuffer.
                 Dim resultCell As New DataBufferCell(cell1)
                resultCell.CellAmount = 1.0
                eventArgs.DataBufferResult.SetCell(api.SI, resultCell, False)
                End If
            Else
            'A number exists in the 1st DataBuffer, but it doesn't exist in the
2ndvDataBuffer.
            'Therefore, add a cell to the result DataBuffer.
                Dim resultCell As New DataBufferCell(cell1)
                resultCell.CellAmount = 1.0
                eventArgs.DataBufferResult.SetCell(api.SI, resultCell, False)
            End If
        End If
Next
'Now, for each cell in DataBuffer2, try to find a number for the same intersection in
DataBuffer1 (the opposite of the loop above).
'Create a cell in the result DataBuffer for each DataBuffer2 cell that doesn't exist in
DataBuffer1.
For Each cell2 As DataBufferCell In eventArgs.DataBuffer2.DataBufferCells.Values
        If (Not cell2.CellStatus.IsNoData) Then
            Dim cell1 As DataBufferCell = eventArgs.DataBuffer1.GetCell
(api.SI,cell2.DataBufferCellPk)
                If cell1 is Nothing Then
                'A number exists in the 2nd DataBuffer, but it doesn't exist in the 1st
DataBuffer.
                'Therefore, add a cell to the result DataBuffer.
                    Dim resultCell As New DataBufferCell(cell2)
                    resultCell.CellAmount = 1.0
                    eventArgs.DataBufferResult.SetCell(api.SI, resultCell, False)
                End If
            End If
        Next
    End If
```

```
End Sub
```

Referencing a Business Rule from a Member Formula or Business Rule

Finance business rules from Member Formulas or other business rules can be called. This is helpful when the same code must be copied to multiple Member Formulas and instead of using the same complicated code, a public function with two lines of code written in a business rule can be called.

First, create a new Finance business rule (in this example, a Finance business rule called SharedFinanceFunctions was created) and then set the Contains Global Functions for Formulas property to True. If the business rule is only being used to hold shared functions, delete most of the content in the main function. However, a main function is still needed even if it is empty.

Next, create a public function or sub in the business rule. See the following example. If any edits to the business rule impact calculation status, it is recommended to assign the Shared business rule to the cube under Application tab > Cube> > Cubes.

Use the business rule in a Member Formula by creating an instance of the business rule and assigning it to a variable. Then, any of the business rule's public functions or sub can be called.

Sample Member Formula Code

```
Dim sharedFinanceBR As New OneStream.BusinessRule.Finance.SharedFinanceFunctions.MainClass
Dim myResult As String = sharedFinanceBR.Test(si, api, args)
```

Sample Business Rule

```
Imports SystemImports System.Data
Imports System.Data.Common
```

```
Imports System.IO
Imports System.Collections.Generic
Imports System.Globalization
Imports System.Linq
Imports Microsoft.VisualBasic
Imports System.Windows.Forms
Imports OneStream.Shared.Common
Imports OneStream.Shared.Wcf
Imports OneStream.Shared.Engine
Imports OneStream.Shared.Database
Imports OneStream.Stage.Engine
Imports OneStream.Stage.Database
Imports OneStream.Finance.Engine
Imports OneStream.Finance.Database

Namespace OneStream.BusinessRule.Finance.SharedFinanceFunctions
    Public Class MainClass

    Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As
        FinanceRulesApi, ByVal args As FinanceRulesArgs) As Object
            Try
                Return Nothing
            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function

        Public Function Test(ByVal si As SessionInfo, ByVal api As FinanceRulesApi, ByVal
args As FinanceRulesArgs) As String
            Try
                Return "This is the result of my Test function!"
            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function
    End Class
End Namespace
```

Conditionally Apply Formulas to Entities and Consolidation Members

Conditionally apply formulas to run only when needed. For example, calculate headcount only at base entities and local currency and the consolidation engine will do the rest.

```
If (api.cons.IsLocalCurrencyForEntity And (Not api.entity.HasChildren)) Then
```

```
    api.data.calculate("A#10999=A#25999-A#25986")
End If
```

Focus Consolidation to Calculate Only When Needed

There are up to seven calculation operations per entity in the consolidation process. Add conditional statements to formulas to limit which consolidation calculation processes will run for a particular formula.

Base Entities Only

```
If Not api.entity.HasChildren Then
```

Local Currency Only

```
If Api.Cons.IsLocalCurrencyforEntity Then
```

Translated Currency Only

```
If api.Cons.IsForeignCurrencyForEntity Then
```

Parent-Child Relationships

`If api.Cons.IsForeignCurrencyForEntity Then` returns True if the current calculated consolidation member also depends on the parent entity (that is, OwnerPreAdj, Share, Elimination, OwnerPostAdj, Top). If there are two different parent entities for the same entity, then there are two different sets of numbers stored for those consolidation members.

At Specific Level of Consolidation

`If (Not api.Entity.HasChildren()) And (api.Pov.Cons.Name.XFEqualsIgnoreCase("Elimination")) Then` would run if the entity is a parent and also if the member of the Consolidation dimension being processed in the Data Unit is Elimination. Note that this is the preferred function to use rather than the formerly supported `api.Pov.Cons.ScriptName`.

Formulas for Calculation Drill Down

To drill down on calculated members, a formula must be entered in the Formula for Calculation Drill Down property. This allows drilling to occur on calculated Account, Flow, User Defined, or Scenario members.

Drill down can occur on data cells copied from one scenario to another using formula or data management sequence. Before displaying the drill results, every cell's Formula for Calculation DrillDown Scenario Property is run. The result determines whether the scenario member will appear as drillable or not. Therefore, use If statements in the formula to narrow in on the cell's storage type or the POV members associated with the data copy so cells do not appear to be drillable when they are not. The following example copies data from the Actual scenario type to Budget:

```
Dim result As New DrillDownFormulaResult()
If args.DrillDownArgs.RequestedDataCell.CellStatus.StorageType =
DataCellStorageType.Calculation Then
    'Use this to drill down to data that was copied using a Scenario Formula.
    result.Explanation = "Formula Definition: Actual = Budget"
    result.SourceDataCells.Add("Cb#Houston:E#Houston:S#Actual")
Else If args.DrillDownArgs.RequestedDataCell.CellStatus.StorageType =
DataCellStorageType.Input Then
    'Use this to drill down to data that was copied using Data Management.
    'result.Explanation = "Data Management Defintion: Actual = Budget"
    'result.SourceDataCells.Add("Cb#Houston:E#Houston:S#Actual")
End If
Return Result
```

Global Entity for Driver Storage

A global entity can be created to store information that is unrelated to the company's data, for example, switches, drivers, or values. To do this within the application, create the entity and set the Is Consolidated property to False. The consolidation process will then skip this entity allowing companies to use this for grouping purposes only.

Avoiding Data Explosion

In some cases, many data intersections unintentionally explode in the cube with unintended values. This can occur in multi-dimensional data structures.

> **Example:** An application has one entity, four fccounts, 100 products (UD1) and 100 customers (UD2).

The user has entered three numbers in the system.

```
(A#Sales:U1#Prod1:U2#Cust1=22.0,   A#Sales:U1#Prod5:U2#Cust1=33.0,
A#Sales:U1#Prod7:U2#Cust100=55.0).
```

The following cases explain what happens when writing the following formulas.

No Explosion

Use the following statement to set the value of a data intersection equal to another or a constant value.

```
api.Data.Calculate("A#Cash1 = A#RestrictedCash")
```

The Cash1 account equals the value in the Restricted Cash account. The only populated UD1 and UD2 members are the None members, because Cash is not typical to delineate by product or customer.

```
api.Data.Calculate("A#Profit:U2#Cust1 = A#Sales * 1.05")
```

Two of the three Sales values are multiplied by 1.05 and copied to the Profit account. Only the Sales values for Customer 1 are used as the source.

Some Explosion

```
api.Data.Calculate("A#Profit = A#Sales:U2#Cust1")
```

The user specified a customer for source data but did not specify which customer to use when writing to the destination. This means answers are written to every base-level member of the customer dimension, resulting in some data explosion. The Profit account has a number for every customer (UD2) using the same product (UD1) member as the source numbers. Profit account will have 300 numbers, but because there were three different source products, there will be some data explosion.

Not specifying a member for a dimension is equivalent to specifying All for a member. If All is specified on the left only, data explosion occurs. Some level of data explosion occurs in the following examples:

- The right side of an equation identifies a specific member or amount.
- The left side of an equation specifies All for one or more dimensions.

Large-scale Explosion when Setting to a Constant

```
api.Data.Calculate("A#Profit = 2.0")
```

Profit will have 1000 numbers stored that are all products by all customers, causing large scale explosion. However, this example is overly simple. If all dimensions are used, trillions of numbers may be stored. A constant is the same as not specifying All for every dimension on the right side of the equation.

Writing Formulas to Avoid Data Explosion

If a member is specified for a dimension on the right side of the equation, explicitly specify a value for that dimension on the left side. The following examples are the same as the previous examples with some explosion and large-scale explosion when setting to a constant, but this time the formulas are written to avoid data explosion.

Specify a Target on the Left Side of the Equation

`api.Data.Calculate("A#Profit = A#Sales:U2#Cust1")` would be better written as `api.Data.Calculate("A#Profit:U2#Cust1 = A#Sales:U2#Cust1" )` if that is what is intended. If the user intended on copying Cust1's sales figures to every customer intersection under the Profit account, the formula would need to be written as the following example:

```
api.Data.Calculate("A#Profit:U2#All = A#Sales:U2#Cust1"
```

`api.Data.Calculate("A#Profit = 2.0")` will fail since the constant of 2.0 implies all members from each dimension. In order for this formula to run, it needs to be `api.Data.Calculate ("A#Profit:F#All:O#All:I#All:U1#All:U2#All:U3#All:U4#All:U5#All:U6#All :U7#All:U8#All = 2.0")`. Although, the user may choose to do otherwise.

Avoiding Data Explosion in Stored Formulas

When writing stored calculations, the member script equations in api.Data.Calculate read or calculate one or more data buffers for the purpose of saving a resulting data buffer. As described earlier, thousands of data cells could be processed using a seemingly simple one-line formula.

Data explosion can occur when a formula is inadvertently written to read or calculate a data buffer and then copy all data cells in that data buffer multiple times to every base-level member of a dimension using the resulting data buffer (often causing hundreds of thousands of new numbers to be saved). We only apply calculation to intersections where data exists. Fortunately, OneStream protects the user from writing formulas that could result in data explosion, but the concepts are important to understand because it is possible to circumvent those protections.

Consider the formula from an earlier example:

```
api.Data.Calculate("A#Sales1 = A#Sales2 + A#Sales3")
```

Now, the formula is changed, so it only copies the sales data for specific customers with UD2 being the customer dimension.

```
api.Data.Calculate("A#Sales1:U2#None = A#Sales2:U2#CustomerX + A#Sales3:U2#CustomerY")
```

This formula will not result in data explosion because there is the same level of detail (that is, the same dimensions) specified in the destination as in every source operand. The example now reads the Sales2 data for CustomerX, adds it to the Sales3 data for CustomerY, and saves the results in the Sales1 account and the UD2 None member.

The following formula is written to cause data explosion:

```
api.Data.Calculate("A#Sales1 = A#Sales2:U2#CustomerX + A#Sales3:U2#CustomerY")
```

The UD2 dimension is specified for the source operands, which is the right side of the equal sign, but UD2 is not specified for the destination, which is the left side of the equal sign.

When processing this formula, OneStream will read the two data buffers and add them together correctly as before. However, it then needs to assign the combined data buffer to the Sales1 account. However, the system cannot use U2#CustomerX or U2#CustomerY because data cells were created by adding those together, and it cannot arbitrarily choose one over the other. The system could also have defaulted to use the U2#None member, but history suggests that this type of rule is more often written in error, and the customer did not intend the results to be stored in the U2#None member. From a maintenance perspective, it is recommended to explicitly specify U2#None if that is the intended destination member.

If something like this were to happen, OneStream will provide an error message notifying the user that data explosion will occur when trying to run the previous formula. Otherwise, the formula will copy the source data buffer to every base-level UD2 member because #All is the default setting for each unspecified member.

To override the error message and force data explosion, explicitly specify U2#All in the destination as shown in the following example. Do not do this. #All should never be used in member scripts for stored formulas. However, the capability is provided for extremely rare circumstances where that functionality relied on using an older product. In this case, the consultant carefully analyzed the quantity of data and metadata settings to ensure the data explosion resulted in a manageable number of data cells.

> **CAUTION:** This causes Data Explosion. Never use #All explicitly in stored Member Formulas.

```
api.Data.Calculate("A#Sales1:U2#All = A#Sales2:U2#CustomerX + A#Sales3:U2#CustomerY")
```

Key Functions

The following list contains the most commonly used functions; however, this is not the complete list of all available functions.See the API Overview Guide..

Account

Name	Function	Description
Account Type	Public Function GetAccountType (MemberId As Integer) As AccountType myAccountType = api.Account.GetAccountType (MemberId)	Retrieves the account type for the member.
Get Cell Account Type	GetCellLevelAccountType	Retrieves the account type of the data cell based on its account and flow settings.

Name	Function	Description
Get Formula Type	Public Function GetFormulaType (MemberId As Integer) As FormulaType myFormulaType = api.Account.GetFormulaType (MemberId)	Returns the formula type if the account is calculated.
Get Plug Account	Public Function GetPlugAccount (MemberId As Integer) As Member myMember = api.Account.GetPlugAccount (MemberId)	Retrieves the plug account.

Consolidation

Name	Function	Description
Calculate	Public Sub Calculate(Formula As String, Optional onBeforeSetDataBuffer As BeforeSetDataBufferDelegate, Optional userState As Object, Optional arg0 As String, Optional arg1 As String, Optional arg2 As String, Optional arg3 As String, Optional arg4 As String, Optional arg5 As String, Optional arg6 As String, Optional arg7 As String) api.Data.Calculate(Formula, onBeforeSetDataBuffer, userState, arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7)	Runs a calculation for a specifically qualified Point of View.
Execute Default Elimination	Public Sub ExecuteDefaultElimination() api.ExecuteDefaultElimination()	Puts data into the Elimination member of the Consolidation dimension.

Name	Function	Description
Second Pass Eliminations	Boolean argument for use in FinanceFunctionType.Calculate rules Dim bValue As Boolean = args.CalculateArgs.IsSecondPassEliminationCalc	Used as a calculate argument to retrieve eliminations data. The argument is designed to ensure the immediate sibling entities within the node are calculated at eliminations. After the immediate sibling entities complete a first pass to consolidation at elimination, the IsSecondPassEliminationCalc argument will allow a sibling entity to retrieve elimination values from any other sibling within the node. For this calculation to work, the scenario setting Use Two Pass Elimination must be set to True.
Execute Default Share	Public Sub ExecuteDefaultShare() api.ExecuteDefaultShare()	Puts data into the Share member of the Consolidation dimension.

Name	Function	Description
Execute Default Translation	Public Sub ExecuteDefaultTranslation (fxRuleTypeRevenueExp As FxRuleType, rateForRevenueExp As Decimal, fxRuleTypeAssetLiab As FxRuleType, rateForAssetLiab As Decimal) api.ExecuteDefaultTranslation (fxRuleTypeRevenueExp, rateForRevenueExp, fxRuleTypeAssetLiab, rateForAssetLiab) or api.ExecuteDefaultTranslation()	Calculates translation by bypassing consolidation and business rules.
Is Consolidated	Public Function IsConsolidated(MemberId As Integer) As TriStateBool myTriStateBool = api.Account.IsConsolidated(MemberId)	Reveals if the account is consolidated.

Name	Function	Description
Percent Consolidated	Public Function PercentConsolidation (Optional EntityId As Integer, Optional ParentId As Integer, Optional varyByScenarioTypeId As Integer, Optional varyByTimeId As Integer) As Decimal myDecimal = api.Entity.PercentConsolidation(EntityId, ParentId, varyByScenarioTypeId, varyByTimeId)	The percent that an entity contributes to a relationship.
Translated Currency Member	Public Function IsForeignCurrencyForEntity (Optional EntityId As Integer, Optional consId As Integer) As Boolean myBoolean = api.Cons.IsForeignCurrencyForEntity (EntityId, consId)	Returns if the Consolidation dimension member is a translated currency member. Used in translation rules.

Data

Name	Function	Description
Allocation	Use journals for allocations	Allocates data across dimensions (for example, Entities, User Defined dimensions, Accounts.) with configurable weighting, all through journals that are generated and posted and can be previewed and unposted.
Convert Data Buffer	api.Data.ConvertDataBuffer	Modifies dimension members for the cells in a data buffer using mapping.

Name	Function	Description
Convert Data Buffer Extended Members	api.Data.ConvertDataBufferExtendedMembers	Automatically aggregates the data for extended members to create data cells for parent members that are base-level members in the destination dimensions. This is used when copying data from a source data buffer created in another cube or scenario where one or more dimensions have been extended.

Name	Function	Description
Get Data Buffer (working with Data Units)	Public Function GetDataBuffer (scriptMethodType As DataApiScriptMethodType, sourceDataBufferScript As String, expressionDestinationInfo As ExpressionDestinationInfo) As DataBuffer myDataBuffer = api.Data.GetDataBuffer (scriptMethodType, sourceDataBufferScript, expressionDestinationInfo)	Retrieves Data Unit values during a consolidation, calculation, or translation.
Get Data Buffer Using Formula	Dim myDataBufer As DataBuffer = api.Data.GetDataBufferUsingFormula ("A#Sales-A#Costs)	Use an entire math expression to calculate a final data buffer.

Name	Function	Description
Get Stored and Dynamically Calculated Values	Public Function GetDataCell(Formula As String, Optional arg0 As String, Optional arg1 As String, Optional arg2 As String, Optional arg3 As String, Optional arg4 As String, Optional arg5 As String, Optional arg6 As String, Optional arg7 As String) As DataCell myDataCell = api.Data.GetDataCell(Formula, arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7)	Retrieves the data contained in an intersection of the cube. If this is a dynamically calculated value, the calculation will be run on the fly before returning the value.

Name	Function	Description
Has Cell Notes or Data Attachments	api.Data.GetDataCell(Formula, Parameters).CellStatus.HasAttachment	Indicates whether the data cell has notes or file attachments. (Boolean) To retrieve the data attachment, query the appropriate member of the View dimension (Annotation, AuditComment, Footnote, VarianceExplanation, Assumptions).
Has No Data	api.Data.GetDataCell(Formula, Parameters).CellStatus.IsNoData	Indicates if the cell has no data. (Boolean)
Has Valid Data	api.Data.GetDataCell(Formula,Parameters). CellStatus.DataIsTimeLogicDerived	Indicates if the cell contains valid data that was derived from time logic. (Boolean)

Name	Function	Description
Get Data Buffer for Custom Share Calculation	Public Function GetDataBufferForCustomShareCalculation (Optional cubeId As Integer, Optional entityId As Integer, Optional ParentId As Integer, Optional scenarioId As Integer, Optional timeId As Integer, Optional viewId As Integer) As DataBuffer myDataBuffer = api.Data. GetDataBufferForCustomShareCalculation (cubeId, entityId, ParentId, scenarioId, timeId, viewId)	Use this function to assist in custom consolidations.

Name	Function	Description
Get Data Buffer for Custom Elim Calculation	Public Function GetDataBufferForForCustomElimCalculation (Optional includeICNone As Boolean, Optional includeICPartners As Boolean, Optional combineImportFormsAndAdjConsolidatedInto Elim As Boolean, Optional cubeId As Integer, Optional entityId As Integer, Optional ParentId As Integer,Optional scenarioId As Integer, Optional timeId As Integer, Optional viewId As Integer) s DataBuffer myDataBuffer = api.Data. GetDataBufferForForCustomElimCalculation (includeICNone, includeICPartners, combineImportFormsAndAdjConsolidatedInto Elim, cubeId, entityId, ParentId, scenarioId, timeId, viewId)	Use this function to assist in custom calculations.

Name	Function	Description
Rounding	Use the VB.NET function for rounding: Round(expression[, numdecimalplaces])	Controls the level of rounding that is used.
Set and Clear Data	Public Sub SetDataCell(Memberscript As String, amount As Decimal, isNoData As Boolean) api.Data.SetDataCell(Memberscript, amount, isNoData)	Sets data to a certain value.
Show Cell Text	Data Attachment Members in the View Dimension: Annotation, Assumptions, AuditComment, Footnote and VarianceExplanation	Use these view members to display data attachment text within a Cube View, which can be used to edit text or display on reports.
Variance	((A-B)/Abs(B))	Returns a variance, but does not consider account type
BetterWorse Difference	Revenue Accounts:((A-B/Abs(B))) Expense Accounts: ((-1*A)-(-1*B))	Returns a variance based on the account type.

Name	Function	Description
VariancePercent	((A-B) / Abs(B)) * 100	Returns a variance percent but does not consider account type.
BetterWorse Percent	Revenue Accounts: ((A-B) / Abs(B)) * 100 Expense Accounts: (((-1 * A) – (-1 * B))/Abs(B)) * 100	Returns a variance percent based on the account type.
Unbalanced Math: Add, Subtract, Multiply, and Divide	Api.Data.Calculate(“A#TargetAccount = AddUnbalanced(A#60000,A#41000:O#Top, O#Top)”)	Required to perform math using two data buffers where additional dimensionality must be specified for the second data buffer.

Entity

Name	Function	Description
Default Currency	Public Function GetLocalCurrency(Optional EntityId As Integer) As Currency myCurrency = api.Entity.GetLocalCurrency (EntityId)	Retrieves the assigned currency for the entity or parent.
Is Descendant	Public Function IsDescendent(dimPk As DimPk, ancestorMemberId As Integer, descendentMemberId As Integer, dimDisplayOptions As DimDisplayOptions) As Boolean myBoolean = api.Members.IsDescendent(dimPk, ancestorMemberId, descendentMemberId, dimDisplayOptions)	Returns if the member is a descendant of another member. (Boolean)
Is Intercompany	Public Function IsIC(Optional EntityId As Integer) As Boolean myBoolean = api.Entity.IsIC(EntityId)	Returns if the entity or account member is an intercompany member.

Flow

Name	Function	Description
Switch Sign	Public Function SwitchSign(MemberId As Integer) As Boolean myBoolean = api.Flow.SwitchSign (MemberId)	Flow dimension only. Responds as to whether credits are switched to debits for the specified member for revenue and expense accounts. (Boolean)
Switch Type	Public Function SwitchType(MemberId As Integer) As Boolean myBoolean = api.Flow.SwitchType (MemberId)	Flow dimension only. Responds as to whether account types are switched for the current or specified member. This can drive translating this member by a different FX rate type. (Boolean)

FX

Name	Function	Description
Currency Type	Public Function GetDefaultCurrencyId(Optional CubeId As Integer) As Integer myInteger = api.Cubes.GetDefaultCurrencyId (CubeId) or Public Function GetCurrency(currencyName As String) As Currency myCurrency = api.Cons.GetCurrency (currencyName)	Retrieves the currency type for the cube or the Consolidation dimension member.
Current Exchange Rate	Public Function GetStoredFxRate(fxRateType As FxRateType, Optional timeId As Integer, Optional sourceCurrencyId As Integer, Optional destCurrencyId As Integer) As FxRate myFxRate = api.FxRates.GetStoredFxRate (fxRateType, timeId, sourceCurrencyId, destCurrencyId)	Retrieves the current exchange rate for the specified entity.

Name	Function	Description
Exchange Rate Calculated	Public Function GetCalculatedFxRate (fxRateType As FxRateType, timeId As Integer) As Decimal myDecimal = api.FxRates.GetCalculatedFxRate (fxRateType, timeId)	Calculates the exchange rate from the default currency to another.
Get FX Rate Type for Asset / Liability Accounts	Public Function GetFxRateTypeForAssetLiability (Optional CubeId As Integer, Optional ScenarioId As Integer) As FxRateType myFxRateType = api.FxRates.GetFxRateTypeForAssetLiability (CubeId, ScenarioId)	Retrieves the default rate type for asset and liability accounts in this cube or scenario (overrides cube value).
Get FX Rate Type for Revenue/ Expense Accounts	Public Function GetFxRateTypeForRevenueExp (Optional CubeId As Integer, Optional ScenarioId As Integer) As FxRateType myFxRateType = api.FxRates.GetFxRateTypeForRevenueExp (CubeId, ScenarioId)	Retrieves the default rate type for revenue and expense accounts in this cube or scenario (overrides cube value).

Name	Function	Description
Get FX Rule Type for Asset / Liability Accounts	Public Function GetFxRuleTypeForAssetLiability (Optional CubeId As Integer, Optional ScenarioId As Integer) As FxRuleType myFxRuleType = api.FxRates.GetFxRuleTypeForAssetLiability (CubeId, ScenarioId)	Retrieves the default translation rule type for asset and liability accounts in this cube or scenario (overrides cube value).
Get FX Rule Type for Revenue/ Expense Accounts	Public Function GetFxRuleTypeForRevenueExp (Optional CubeId As Integer, Optional ScenarioId As Integer) As FxRuleType myFxRuleType = api.FxRates.GetFxRuleTypeForRevenueExp (CubeId, ScenarioId)	Retrieves the default translation rule type for revenue and expense accounts in this cube or scenario (overrides cube value).
Translate	Public Sub Translate(sourceDataBufferScript As String, destDataBufferScript As String, fxRuleType As FxRuleType, rate As Decimal) api.Data.Translate(sourceDataBufferScript, destDataBufferScript, fxRuleType, rate)	Performs the translation method assigned to the cube or scenario (periodic or direct).

Journals

Name	Function	Description
Journal Postings Allowed	Public Function AllowAdjustments(Optional EntityId As Integer, Optional varyByScenarioTypeId As Integer, Optional varyByTimeId As Integer) As Boolean myBoolean = api.Entity.AllowAdjustments(EntityId, varyByScenarioTypeId, varyByTimeId)	Results determine if journal postings are allowed for the member.
Journal Postings from Children Allowed	Public Function AllowAdjustmentsFromChildren (Optional EntityId As Integer, Optional varyByScenarioTypeId As Integer, Optional varyByTimeId As Integer) As Boolean myBoolean = api.Entity.AllowAdjustmentsFromChildren(EntityId, varyByScenarioTypeId, varyByTimeId)	Results determine if journal postings from children are allowed for this member.

Member

Name	Function	Description
Base Members	HasChildren = false, for example, A#Root.Children (HasChildren=False)	Determines if the member is a base member.
Get Member ID	api.Members.GetMember(dimTypeId, MemberName).MemberPk.MemberID or api.POV.Dimension.Memberpk.Memberid	Retrieves the name for the selected member.
Get Member Name	api.Members.GetMember(dimTypeId, MemberName).Name	Retrieves the name for the selected member.
Member Lists	See section of documentation on Specify Member List Settings.	Retrieves the members from a named list stored in a business rule.
Member Name	api.POV.AccountDim, api.POV.EntityDim, etc	Retrieves the member name.

Name	Function	Description
Member Name or ID	Get member name from member ID: Public Function GetMember(dimTypeId As Integer, MemberId As Integer) As Member myMember = api.Members.GetMember(dimTypeId, MemberId) Get member ID from member name: Public Function GetMember(dimTypeId As Integer, MemberName As String) As Member myMember = api.Members.GetMember(dimTypeId, MemberName)	Retrieves the member for the specified ID number or name.
Top Member	api.Account.GetTopMemberForDimType (AccountMemberId, dimTypeForTopMember)	Retrieves the top member of the selected dimension.

Name	Function	Description
Is Base	Public Function IsBase(dimPk As DimPk, ancestorMemberId As Integer, baseMemberId As Integer, Optional dimDisplayOptions As DimDisplayOptions) As Boolean myBoolean = api.Members.IsBase(dimPk, ancestorMemberId, baseMemberId, dimDisplayOptions)	Determines whether the POV member is the base of a defined member.
Get First Common Parent	Public Function GetFirstCommonParent(dimPk As DimPk, topMostMemberId As Integer, MemberIdA As Integer, MemberIdB As Integer, Optional dimDisplayOptions As DimDisplayOptions) As Member myMember = api.Members.GetFirstCommonParent (dimPk, topMostMemberId, MemberIdA, MemberIdB, dimDisplayOptions)	Returns the first common parent between multiple members.

Scenario

Name	Function	Description
Scenario Consolidation View	Public Function GetConsolidationView (Optional ScenarioId As Integer) As ViewMember myViewMember = api.Scenario.GetConsolidationView (ScenarioId)	Determines if the scenario's consolidation view is set to YTD or periodic.
Scenario Default View	Public Function GetDefaultView (Optional ScenarioId As Integer) As ViewMember myViewMember = api.Scenario.GetDefaultView (ScenarioId)	Retrieves the scenario's default view.
Scenario Input Frequency	Public Function GetInputFrequency (Optional ScenarioId As Integer) As Frequency myFrequency = api.Scenario.GetInputFrequency (ScenarioId)	Retrieves the scenario's input frequency.

Status

Name	Function	Description
Calc Status	Public Function GetCalcStatus(Optional CubeId As Integer, Optional EntityId As Integer, Optional ParentId As Integer, Optional consId As Integer, Optional ScenarioId As Integer, Optional timeId As Integer) As CalcStatus myCalcStatus = api.CalcStatus.GetCalcStatus(CubeId, EntityId, ParentId, consId, ScenarioId, timeId)	Retrieves the calculation status for the data intersection.
Impact Calc Status	Public Sub ImpactCalcStatus(Optional CubeId As Integer, Optional EntityId As Integer, Optional ParentId As Integer, Optional consId As Integer, Optional ScenarioId As Integer, Optional timeId As Integer) api.CalcStatus.ImpactCalcStatus(CubeId, EntityId, ParentId, consId, ScenarioId, timeId)	Changes the status of the specified Data Unit to impact.

Time

Name	Function	Description
MemberId for the Month to which a Week belongs	Dim timeIdForMonth As Integer = BRApi.Finance.Time.ConvertIdToClosestIdUsingAnotherFrequency(si, timeIdForWeek, Frequency.Monthly)	Weekly applications only: This determines to which month a specific week belongs. Used within a Finance business rule.

Functions

Name	Function	Description
Days Sales Outstanding	Dim cell As DataCell = api.Functions.GetDSODataCell (acctsReceivableMember, salesMember)	Calculates days sales outstanding.

Name	Function	Description
Dynamic Simulation of Consolidation	api.Functions.GetEntityAggregationDataCell (memberScript as string, Optional useStoredAmountsWhereCalcStatusIsOK as Boolean, Optional fxRateTypeRevenueExpOverride as String, Optional fxRuleTypeRevenueExpOverride as String, Optional fxRateTypeAssetLiabOverride as String, Optional fxRuleTypeAssetLiabOverride as String)	Pseudo or approximate consolidation of a data cell.
GetStage or GetCustom	See the API Overview Guide for examples: GetStageBlendTextUsingCurrentPOV GetStageBlendText GetStageBlendNumberUsingCurrentPOV GetStageBlendNumber GetStageBlendDataTableUsingCurrentPOV GetStageBlendDataTable GetCustomBlendDataTableUsingCurrentPOV GetCustomBlendDataTable	Several functions that retrieve text and calculate values by reading values from the OneStream stage or a custom relational table.

Examples of Key Functions in Use

Here are examples of Key Functions in use.

Calculate Data

The most common function used is `api.data.calculate`, which sets the value of one or more values (left side of formula) equal to another (right side). A final argument (optional) can be added as true or false as to whether to use a data cell storage type of durable. Durable data will not be cleared automatically when a Data Unit is recalculated. It can only be cleared by calling `api.Data.ClearCalculatedData` with the clearDurableCalculatedData boolean property set to True.

```
api.Data.Calculate(formula, isDurableCalculatedData)
```

For example, the following stat account is used to calculate total cost of sales for three months and is not set to be a durable storage method (optional argument):

```
api.data.calculate("A#TOT_COS_LAST3:V#YTD = A#TOT_COS:V#Periodic + A#TOT_
COS:T#POVPrior1:V#Periodic +
    A#TOT_COS:
    T#POVPrior2:V#Periodic", False)
```

An alternative to this overloaded function is to provide Member Filters (all optional) that can be used to filter the results before saving them to the target to affect fewer intersections, for example, only to be applied to certain flow members:

```
api.Data.Calculate(formula, accountFilter, flowFilter, originFilter, icFilter, ud1Filter,
ud2Filter,
    ud3Filter, ud4Filter,
    ud5Filter, ud6Filter, ud7Filter, ud8Filter, onEvalDataBuffer, userState,
isDurableCalculatedData)
```

Another alternative allows the use of arguments to be applied:

```
api.Data.Calculate(Formula, onBeforeSetDataBuffer, userState, arg0, arg1, arg2, arg3, arg4,
arg5, arg6, arg7)
```

Clear Calculated Data

Clearing calculated data is performed when calculated members need to be reset to NoData. This will result in the Data Units requiring a calculation. Note that the final argument must be true to clear any data with a storage type of durable. See Calculate Data section above for how to set a calculation to store data as durable.

```
api.Data.ClearCalculatedData(dataBufferScript, clearCalculateData, clearTranslationData,
    clearConsolidationData,
    clearDurableCalculatedData)

api.Data.ClearCalculatedData("A#[cash deposits]:ud1#Production", true, true, true, false)
```

DataBuffer

The DataBuffer log helps debugging and troubleshooting. It allows datastring results to be written to any target and multiple string results can be merged. The output is formatted as tab delimited for copy and paste functionality to Excel. Results include the DataBuffer formula, the size and number of records, and the XFCommon member PK. It also contains a default record limit to guard against log explosion.

Examples of DataBuffer outputs:

- Error Log
- Send to SQL
- Send to a File
- Email
- FTP

When setting a value equal to another value, the item on the left side of the expression is the value being set, and the item on the right side is the value being queried or calculated to set the left side. Example: F#BeginBalance = F#EndingBalance.T#POVPrior1 would set the beginning balance in the Flow dimension to the prior period's ending balance. In a business rule, the DestinationInfo is the left side of the equation while a GetDataBuffer is the right side of the equation.

```
Dim destinationInfo As ExpressionDestinationInfo = api.Data.GetExpressionDestinationInfo
    ("A#EBITDA:UD1#Tires")
Dim sales As DataBuffer = api.Data.GetDataBuffer("A#Sales:UD1#Tires", destinationInfo)

Dim operatingExpenses As DataBuffer = api.Data.GetDataBuffer
("A#OperatingExpenses:UD1#Tires",
    destinationInfo)
Dim ebitda As DataBuffer = (sales - operatingExpenses)
api.Data.SetDataBuffer(ebitda, destinationInfo)
```

This translates to the following equation:

```
A#EBITDA:UD1#Tires=A#Sales:UD1#Tires - A#OperatingExpenses:UD1#Tires
```

GetDataBufferUsingFormula

Use an entire math expression to calculate a final data buffer. `api.Data.GetDataBufferUsingFormula` can perform the same data buffer math as `api.Data.Calculate`, but the result is assigned to a variable where `api.Data.Calculate` saves the calculated data.

Example

```
Dim myDataBuffer As DataBuffer = api.Data.GetDataBufferUsingFormula ("A#Sales-A#Costs")
```

Loop over the contents of `myDataBuffer` to conditionally change each data cell.

Formula Variables

There is additional capability to using formula variables to achieve the same level of flexibility and integration as using Evals. After creating a data buffer variable, name it as a formula variable and reference it inside `api.Data.Calculate` or other calls to `api.Data.GetDataBufferUsingFormula`. This provides flexibility and can improve performance because the data buffer is calculated once and the variable is reused multiple times.

Example

```
Dim myDataBuffer As DataBuffer = api.Data.GetDataBufferUsingFormula ("A#Sales-A#Costs")
api.Data.FormulaVariables.SetDataBufferVariable("myDataBuffer",myDataBuffer,False)
api.Data.Calculate("A#Profit=A#5000 + $myDataBuffer
```

Use `api.Data.FormulaVariables.SetDataBufferVariable` to name the data buffer. Pass in any name followed by the data buffer variable. Enter a True or False value for Uses Indexes to Optimize Repeat Filtering. Using True will reuse the same data buffer using FilterMembers and improve performance. After naming the data buffer, use a dollar sign and the name when referencing it in a script.

FilterMembers

Use FilterMembers inside of an `api.Data.Calculate` or `api.Data.GetDataBufferUsingFormula` script.

Example

```
Dim myDataBuffer As DataBuffer
Api.Data.GetDataBufferUsingFormula("FilterMembers(A#All,A#6000,[A#[3600].Base])")
```

Change a data buffer and only include numbers for the specified dimensions. The first parameter is the starting data buffer. This can be a variable name or an entire math equation in parentheses. There can be as many parameters as needed to specify Member Filters, and different Member Filters can be used for multiple dimension types. The resulting filtered data buffer will only contain numbers that match the members in the filters.

RemoveMembers

This uses the same syntax as FilterMembers, but it takes the data cells away for the specified members instead of keeping them.

Retrieving Member Names from IDs

This retrieves the Member names from the IDs when looping over the cells in a data buffer.

Example

```
For Each sourceCell As DataBufferCell In myDataBuffer.DataBufferCells.Values
     Dim accountName As String = sourceCell.DataBufferCellPk.GetAccountName(api)
     Dim ud1Name As String = sourceCell.DataBufferCellPk.GetUD1Name(api)
     If ud1Name = "None" Then BRApi.ErrorLog.LogMessage(si, "UD1"& ud1Name, Nothing)
Next
```

Logging Contents of a Data Buffer

Log the contents of a data buffer to the error log when writing business rules to make corrections and troubleshoot along the way.

Example

```
myDataBuffer.LogDataBuffer(api,”MyDataBufferOutput”,1000)
```

The third parameters (1000) indicates the maximum number of cells to include in the log and displays what is in the data buffer.

Example api

```
api.LogMessage(XFErrorLevel.Information, "MyDataBuffer As a CSV String For Excel",
myDataBuffer.GetCSVString(api, False, 1000))
```

The False parameter specifies whether to include nember IDs in the output. Member names are always included.

Remove Functions in Formulas

Remove Zeros

The `RemoveZeros` function evaluates a source data buffer and removes all cells where the amount is zero whether it is flagged as a NoData cell or not. Therefore, it is not necessary to use both `RemoveZeros` and `RemoveNoData` together since `RemoveZeros` handles both situations. This can be identified on individual cells as the Cell Amount under Cell Status.

This function is important to use for performance purposes. Use the `RemoveZeros` function in calculations where there is a substantial amount of No Data or 0.00 cells in Data Units. This can be determined by looking at the Data Unit Statistics when right-clicking on a cell in a Cube View.

⊟ General	
Total Number of Stored Records	1637
⊟ NODATA Status	
Number of NODATA Cells	4
Number of Zero Cells	13
Number of Real Cells	1617
Number of Derived Cells	3

Example Using api.Data.Calculate

```
'Run for base Entities and local currency only
    If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyforEntity())) Then
    api.Data.Calculate("A#CashCalc=RemoveZeros(A#10000)")
End If
```

Example Using GetDataBufferUsingFormula

```
If ((Not api.Entity.HasChildrenQ) And (api.Cons.IsLocalCurrencyforEntityQ)) Then

    ' Get Data Buffer by using GetDataBufferUsingFormula
    Dim dataBufferExample As DataBuffer = api.Data.GetDataBufferUsingFormula("RemoveZeros
(A#100000)")

    ' Set Data Buffer Variable to pass into api.Data.Calculate.
    ' Can be used for multiple instances of api.Data.Calculate.
    ' Create a unique name to store the Data Buffer as a Formula Variable.
    api.Data.FormulaVariables.SetDataBufferVariable("dataBufferExample", dataBufferExample,
False)

    ' Pass Formula Variable into api.Data.Calculate using a $
    api.Data.Calculate("A#CashCalc = $dataBufferExample")

End If
```

Remove NoData

The RemoveNoData function evaluates a source data buffer and removes data cells that have a cell amount of NoData.

This function is important to use for performance purposes. Use the RemoveNoData function in calculations where there is a substantial number of cells with a Cell Status of NoData in Data Units. This can be determined by looking at the Data Unit Statistics when right-clicking on a cell in a Cube View.

Example Using api.Data.Calculate

```
'Run for base Entities and local currency only
    If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyforEntity())) Then
    api.Data.Calculate("A#CashCalc=RemoveNoData(A#10000)")
End If
```

Example Using GetDataBufferUsingFormula

```
If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyforEntity())) Then

    ' Get Data Buffer by using GetDataBufferUsingFormula to do the math
    Dim dataBufferExample As DataBuffer = api.Data.GetDataBufferUsingFormula("RemoveNoData
(A#10000)")

    ' Set Data Buffer Variable to pass into api.Data.Calculate.
    ' Can be used for multiple instances of api.Data.Calculate.
    ' Create a unique name to store the Data Buffer as a formula variable.
    api.Data.FormulaVariables.SetDataBufferVariable("dataBufferExample", dataBufferExample,
False)

    ' Pass variable into api.Data.Calculate using a $
    api.Data.Calculate("A#CashCalc = $dataBufferExample")

End If
```

In the calculation performance testing for a single Data Unit, the use of RemoveZeros and RemoveNoData using `api.Data.Calculate` or `GetDataBufferUsingFormula` rendered a significant performance advantage. This is testing the before and after calculation time of a single formula change for a Data Unit. Times may vary from Data Unit to Data Unit and application to application. However, the use of RemoveZeros and RemoveNoData is highly recommended in formulas where cube and dimensionality designs lend to sparse data models. This is not limited to sparse data models as this can be identified in dense data models as well.

Dynamic Calc Using GetDataCell

A Dynamic Calc returns an account value on demand without storing the data in the cube. These members cannot be referenced by other members in calculating their values. Dynamic Calcs are best used in members required for reporting only. One ideal use is the GetDataCell function in ratio-style statistical accounts, for example, Current Ratio.

```
Return api.Data.GetDataCell("Divide(A#[CurrentAssets], A#[CurrentLiabilities]")
```

Dynamic Filters and Aliases

Create dynamic Member Filters to assign to rows or columns within Cube Views. The following example is a common example of a variance column that subtracts one member from another. Adding the :Name("Membername") option at the end gives the ability to assign a display name to the header of the Cube View. The double quotes in the Name() function are optional.

```
Cube View Filter: GetDataCell("Variance(T#POV,T#Prior12)"):Name("Variance")
```

Error Traps

```
Try
  if api.POV.Cons.name =api.POV.GetEntityCurrency() then
     api.Data.Calculate("A#Cash1=A#[Restricted Cash] + 70000")

  End If
catch ex as exception
api.LogError(ex)
End Try
```

Math Functions

Use the math functions built into VB.NET.

Example: Absolute Value: math.abs()

Example: Rounding: math.round()

Several other supported math functions and examples are listed here: Math Functions

Member List Functions

When a cube filter is run and the filter value does not match a member, the system looks in all business rules for the MemberListProperty property to find a matching member list name. Users should create MemberLists in business rules that are not assigned to cubes because the entire business rule will be run if the system finds a matching member list name. Reference MemberLists format: E#Root.PartnerList.

The following example first declares the two MemberListHeader names of Sample Member List and PartnerList. It then addresses each MemberList. Sample Member List shows a way to get the children of an entity. E#Texas.[Sample Member List] is just another way of saying E#Texas.Children. In the PartnerList example, E#Root.PartnerList will generate a list that includes Paris and Nice.

```
Case Is = FinanceFunctionType.MemberListHeaders
                ' Additional logic that defines the names of the two custom Member ListsDim
myMemberListHeaders = New List(Of MemberListHeader)myMemberListHeaders.Add(new
   MemberListHeader("Sample Member List"))myMemberListHeaders.Add(new
   MemberListHeader("PartnerList"))
Return myMemberListHeaders

Case Is = FinanceFunctionType.MemberList
   If args.MemberListArgs.MemberListName = "Sample Member List" Then
           Dim myMemberListHeader = new MemberListHeader(args.MemberListArgs.MemberListName)
           Dim myMembers = new List(Of Member)()
           Dim myMemberList = New MemberList(myMemberListHeader,
           myMembers)myMembers.AddRange(api.Members.GetChildren(args.MemberListArgs.DimPk,

args.MemberListArgs.TopMember.MemberPk.MemberId,args.MemberListArgs.DimDisplayOptions))

          Return myMemberList
   Else If args.MemberListArgs.MemberListName = "PartnerList" Then
          Dim myMemberListHeader = new MemberListHeader(args.MemberListArgs.MemberListName)
```

```
        Dim myMembers = new List(Of Member)()
          Dim myMemberList = New MemberList(myMemberListHeader, myMembers)

         myMembers.AddRange(api.Members.GetBaseMembers(args.MemberListArgs.DimPk,

         api.Members.GetMember
(args.MemberListArgs.DimPk.dimtypeid,"Paris").Memberpk.Memberid,
         args.MemberListArgs.DimDisplayOptions))
         myMembers.AddRange(api.Members.GetBaseMembers(args.MemberListArgs.DimPk,
         api.Members.GetMember
(args.MemberListArgs.DimPk.dimtypeid,"Nice").Memberpk.Memberid,
         args.MemberListArgs.DimDisplayOptions))
         Return myMemberList
End If
```

POV Object

In business rules, only Data Unit dimensions are valid in the POV object, not Account or User Defined members, so the following code only works in business rules:

```
Dim AcctID as Integer=api.POV.Account.MemberPk.MemberID
```

```
Dim AcctID as Integer=api.POV.GetDataCellPk.AccountID
```

Retrieve Time Varying Property of Relationship's Percent Consolidated Entity Attribute

```
' Get Entity ID, Parent Entity ID, Scenario ID, Scenario Type ID, and Time ID
Dim EntityId As Integer = api.Pov.Entity.MemberPk.MemberId
Dim ParentId As Integer = api.Pov.Parent.MemberPk.MemberId
Dim ScenarioId As Integer = api.Pov.Scenario.MemberPk.MemberId
Dim ScenarioTypeId As Integer = api.Scenario.GetScenarioType(ScenarioId).Id
Dim TimeId As Integer = api.Time.GetIdFromName("2013M1")

' Get time-varying node percent consolidation value
' for the current Scenario POV and 2013M1
Dim myDecimal As Decimal = api.Entity.PercentConsolidation(EntityId, ParentId,
ScenarioTypeId, TimeId)

' Log the time-varying property value
api.LogError(
    XFErrorLevel.Information,
    myDecimal & " " & api.Pov.Entity.Name & "." & api.Pov.Parent.Name
)
```

> **TIP:** `API.POV.Parent` object is only relevant for relationship-level consolidation members.

SetDataCell

Use SetDataCell to set a value equal to another value. Any dimensions listed on the right side must have a member specified. You do not need to have all dimensions on the right side, but if they are not specified, the current POV is used.

```
' Set a data cell using explicit Memberscript, amount, and isNoData flag
api.Data.SetDataCell(Memberscript, amount, isNoData)

' Example: Set a data cell on Restricted Cash for a specific intersection
api.Data.SetDataCell(
    "A#[Restricted Cash]:O#Forms:F#None:IC#None:" &
    "U1#None:U2#None:U3#None:U4#None:" &
    "U5#None:U6#None:U7#None:U8#None",
    50,
    False
```

```
)
```

Translate

The `api.data.translate` function is the same as `api.data.calculate`, but aggregates AdjInput data into the AdjConsolidated member.

```
'GetParentCurrency only returns a value when running a translate.
If api.Parameters.FunctionType = FinanceFunctionType.Translate Then if api.POV.Cons.Name =
    api.POV.GetParentCurrency.Name Then
    api.Data.Translate("A#[Restricted Cash]=A#[Restricted Cash]:C#[USD]*10")end if
End If
```

Unbalanced Math

The unbalanced math functions are required when performing math with two data buffers where the second data buffer needs to specify additional dimensionality. The term unbalanced is used because the script for the second data buffer can represent a different set of dimensions from the other data buffer in the `api.Data.Calculate` text. These functions prevent data explosion.

In the following examples, the first two parameters represent the first and second data buffers on which to perform the function. The third parameter represents the members to use from the second data buffer when performing math with every intersection in the first data buffer. The math favors the intersections in the first data buffer without creating additional intersections.

It is key that the dimensionality of the target (left side of the equation) matches the dimensionality of the first data buffer on the right side of the equation (argument 1).

AddUnbalanced

```
api.Data.Calculate("A#TargetAccount = AddUnbalanced(A#OperatingSales,
A#DriverAccount:U2#Global,
    U2#Global)")
```

SubtractUnbalanced

```
api.Data.Calculate("A#TargetAccount = SubtractUnbalanced(A#OperatingSales,
A#DriverAccount:U2#Global,
    U2#Global)")
```

DivideUnbalanced

```
api.Data.Calculate("A#TargetAccount =DivideUnbalanced (A#OperatingSales,
A#DriverAccount:U2#Global,
    U2#Global)")
```

MultiplyUnbalanced

```
api.Data.Calculate("A#TargetAccount =MultiplyUnbalanced (A#OperatingSales,
A#DriverAccount:U2#Global,
    U2#Global)")
```

Consider this example. A#OperatingSales has 100 stored records in January for a single entity.

Because A#OperatingSales has a total of 100 stored values, A#TargetAccount will end up with 100 stored numbers and the amounts would be the values from A#OperatingSales plus, minus, multiplied by, or divided by A#DriverAccount:U2#Global for each of those 100 intersections.

This means that if there was no data in A#OperatingSales:U2#Widgets, then even though the UD2 dimension is unspecified in the target and in the first data buffer expression, no record would be created, hence avoiding data explosion. The most common use case would be applying a driver for some of the dimensions.

ConvertUnbalanced

This function is related to the unbalanced math functions. See Unbalanced Math. It is used to convert a data buffer so that it is balanced with an `api.Data.Calculate` script where unbalanced math does not apply. This is necessary when using an `api.Data.GetDataBufferUsingFormula` to calculate a data buffer where the script was not balanced to match a formula in another script where a data buffer variable needs to be used.

In the following example, a myDataBuffer was created to have data for all stored accounts, but the subsequent `api.Data.Calculate` scripts expects each operand to use a specific account. The ConvertUnbalanced function filters the data buffer to only include the specified account name and it also converts the data buffer to make it balanced and consistent with the destination. The same data buffer can be reused multiple times.

Example

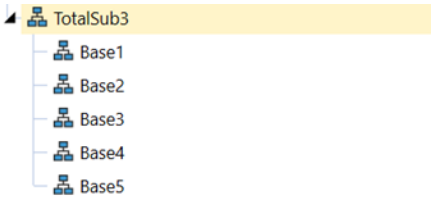
```
Dim myDataBuffer As DataBuffer = api.Data.GetDataBufferUsingFormula("A#All")
api.Data.FormulaVariables.SetDataBufferVariable("myDataBuffer ", myDataBuffer, True)
api.Data.Calculate("A#6050 = ConvertUnbalanced($myDataBuffer, A#6000) +
ConvertUnbalanced($myDataBuffer, A#3000)")
```

Finance Function Types

Use Finance function types, except for Custom Calculate, to run logic in certain calculation sequences while a function type is in process. There are thirteen function types:

Calculate
This function type features additional logic during calculation of Entity, Consolidation, Scenario, and Time. This sets the value of one or more values (left side of formula) equal to another (right side). It then runs a calculation for a specifically qualified Point of View. This is the most common function used.

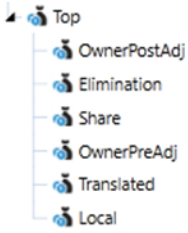
There are situations where the entity being processed must access another entity's data. In situations involving pulling Consolidation dimension elimination results from other entities as siblings, the multi-thread processing of the calculations requires an additional function to ensure the calculations are complete. The following example shows this sibling relationship. The entities Base1, Base2, Base3, Base4, and Base5 are siblings under TotalSub3 that would be calculated simultaneously during a consolidation:



For this purpose, the Calculate Finance functiontType supports the argument IsSecondPassEliminationCalc.

```
Case Is = FinanceFunctionType.Calculate
    If (api.Pov.Cons.MemberID = DimConstants.ConsElimination) And
(args.CalculateArgs.IsSecondPassEliminationCalc) Then
```

During the Calculate process, this function allows business rules to run after the sibling entities have calculated results to the Consolidation Elimination member. The following reference is for the Consolidation dimension:



After all the sibling Entity members are calculated to elimination, the business rules within the IsSecondPassEliminationCalc will be run.

Translate

This function type features additional logic that uses custom translation.

FXRate

This function type features custom logic used to determine foreign exchange rates for any intersection.

Consolidate Share

This function type features additional logic used during the custom calculation of the Share member.

Consolidate Elimination

This function type features additional logic used during the custom calculation of the Elimination member.

Custom Calculate

A CustomCalculate Finance function type can be used i to run a single year custom calculation using a Dashboard Parameter Component Server Task Action. This is considered a partial calculation and does not store the calculated data or run the calculation during a consolidation. Running a custom calculation from a dashboard will impact calculation status for the affected Data Unit even if the data does not change.

See "Data Management" in *Application Tools Guide* for details on creating this type of Data Management Step.

Example

```
Select Case api.FunctionType
              Case Is = FinanceFunctionType.CustomCalculate
            If args.CustomCalculateArgs.FunctionName.XFEqualsIgnoreCase("FunctionName") Then
            Me.CalcTest(si, globals, api, args)
            api.Data.Calculate("A#TFS2903 = A#10000 + A#69000)
            End If
         End Select
```

Calculation Drill Down Member Formula

This function type provides custom drill down results.

Conditional Input Rule

Conditional Input Rules make data cells read-only. While the settings for this can be done directly on the cube, using a Conditional Input business rule offers more flexibility and still allows the use of the cube settings. This rule can return the following options: ConditionalInputResultType.Default, ConditionalInputResultType.NoInput, ConditionalInputResultType.NoInputAllowCellDetail, and ConditionalInputResultType.NoCellDetailAllowInput.

The following business rule example will make all cells for the Account 6000 read-only. This should be added to a business rule attached to a cube.

```
Case Is = FinanceFunctionType.ConditionalInput
                If api.Pov.Account.Name.XFEqualsIgnoreCase("6000") Then
                    Return ConditionalInputResultType.NoInput
                End If
            Return ConditionalInputResultType.Default
```

Confirmation Rule

This function type features special logic that runs with confirmation rules.

Data Cell

This function type features named GetDataCell calculations that can be reused such as a Better/Worse calculation in Cube Views.

Dynamic Calc Account

This function type features special logic to use in Dynamic Calc members.

Member List and Member List Headers

This function type features a custom list of members for use in Cube Views and other areas.

`Select Case api.FunctionType` is the expression used when a certain process needs to be isolated and run special logic. See the following examples:

```
Select Case api.FunctionType
```

```
    Case FinanceFunctionType.Calculate
        ' Additional logic to run with every calculation.
        api.Data.Calculate("A#DACash1 = A#Cash4 + 1")
        api.Data.Calculate("A#2150:F#Movement = V#Periodic:A#5750:F#None")

        If api.MetaData.Cons.Name = "USD" Then
            api.Data.Calculate("A#Cash1 = A#[Restricted Cash]")
        End If

    Case FinanceFunctionType.Translate
        ' Additional logic to run with every translation.
        ' If api.ExecuteDefaultTranslation is not included, the standard
        ' translation process will not run.
        api.ExecuteDefaultTranslation()

        Dim cubeId As Integer = api.Cubes.GetCubeInfo("GolfStream").Cube.CubeId
        Dim fxRule As FxRuleType = api.Cubes.GetFxRuleTypeForAssetLiability(cubeId)

        api.Data.Translate(
            "A#[Restricted Cash]:C#[USD]",
            "A#[Restricted Cash]",
            fxRule,
            0.5)

    Case FinanceFunctionType.ConsolidateShare
        ' Additional logic to run when the Share member of the
        ' Consolidation dimension is being calculated.
        api.ExecuteDefaultShare()
        api.Data.Calculate("A#[Restricted Cash] = A#[Restricted Cash] * 0.8")

    Case FinanceFunctionType.ConsolidateElimination
        ' Additional logic to run while the Elimination member of the
        ' Consolidation dimension is being calculated.
        api.ExecuteDefaultElimination()
        api.Data.Calculate("A#[IC AR] = A#[IC AR] * 1.1")

    Case FinanceFunctionType.MemberListHeaders
        ' Additional logic that defines the names of custom member lists.
        Dim myMemberListHeaders As New List(Of MemberListHeader)()
        myMemberListHeaders.Add(New MemberListHeader("Sample Member List"))

        Return myMemberListHeaders

    Case FinanceFunctionType.MemberList
        ' Additional logic that defines the members within custom member lists
        ' that are included in business rules.
        ' Example: "A#Root.[Sample MemberList]"

        If args.MemberListArgs.MemberListName = "Sample Member List" Then
            Dim myMemberListHeader As New MemberListHeader
(args.MemberListArgs.MemberListName)
            Dim myMembers As New List(Of Member)()
            Dim myMemberList As New MemberList(myMemberListHeader, myMembers)
```

```
            myMembers.AddRange(
                api.Members.GetChildren(
                    args.MemberListArgs.DimPk,
                    args.MemberListArgs.TopMember.MemberPk.MemberId,
                    args.MemberListArgs.DimDisplayOptions))

            Return myMemberList
        End If

End Select
```

Ultimate Ownership

This function calculates and stores ultimate ownership results for every ancestor-child entity relationship. The current entity being calculated is the ancestor entity, and when storing the results, the members in the Intercompany dimension are used to represent each descendant entity. This function assumes the source direct ownership numbers are weights and are typically entered by a user (for example, numbers between 0.0 and 1.0).

C#OwnerPreAdj:O#AdjInput is using an account that does not consolidate and accepts manual entry for adjustments in those parent-child relationships. It cascades the user-entered amounts up the entity tree and multiplies them when a parent owns part of a xhild and that child owns part of a grandchild.

sourceDirectOwnershipMemberscript represents the data cells where the user-entered direct ownership amounts are found for parent-child entity combinations.

destUltimateOwnershipMemberscript represents the intersection for the resulting calculations. All results are stored using the current Data Unit being calculated.

This function is intended to run only for the entity's local consolidation member, and it uses IC members to store the results for each descendant entity.

> **Example:** Use the following Member Formula in the UltimateOwnership account.

```
If api.Cons.IsLocalCurrencyForEntity() Then
            api.Data.CalculateUltimateOwnership
("C#OwnerPreAdj:A#DirectOwnership:O#AdjInput",
            "A#UltimateOwnership:O#Forms", 1.0)
        End If
```

Here are the steps to set up ownership entry and calculations in an application. This functionality is intended to be used in a separate control cube accessed using the main financial cube:

Create Two Accounts

DirectOwnership

This is for the user to enter Direct Ownership for each parent-child entity combination.

`AccountType = BalanceRecurring, IsConsolidated = False, Adjustment Type = Data Entry`

UltimateOwnership

This stores the calculated results in parent entities using IC members for every descendant entity.

`AccountType = BalanceRecurring, FormulaType = Formula Pass 1, Allow Input = False, IsConsolidated = False, Is IC Account = Conditional`

Formula

```
If api.Cons.IsLocalCurrencyForEntity() Then
            api.Data.CalculateUltimateOwnership
("C#OwnerPreAdj:A#DirectOwnership:O#AdjInput",
            "A#UltimateOwnership:O#Forms", 1.0)
        End If
```

Create a Cube View for Entering Direct Ownership Amounts

Cube View POV

Consolidation = OwnerPreAdj, Origin = AdjInput, Flow, IC and UDs = None

Column

A#DirectOwnership

Row

E#Root.TreeDescendants

Create a Cube View for Running a Consolidation and Viewing the Ultimate Ownership Results

Cube View POV

Consolidation = Local, Account = UltimateOwnership, Origin = Forms, Flow IC and UDs = None

Column

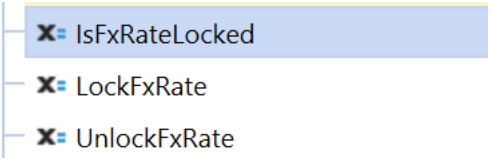
IC#ICEntities.Base

Row

E#Root.TreeDescendants

BRAPI Rule Functions

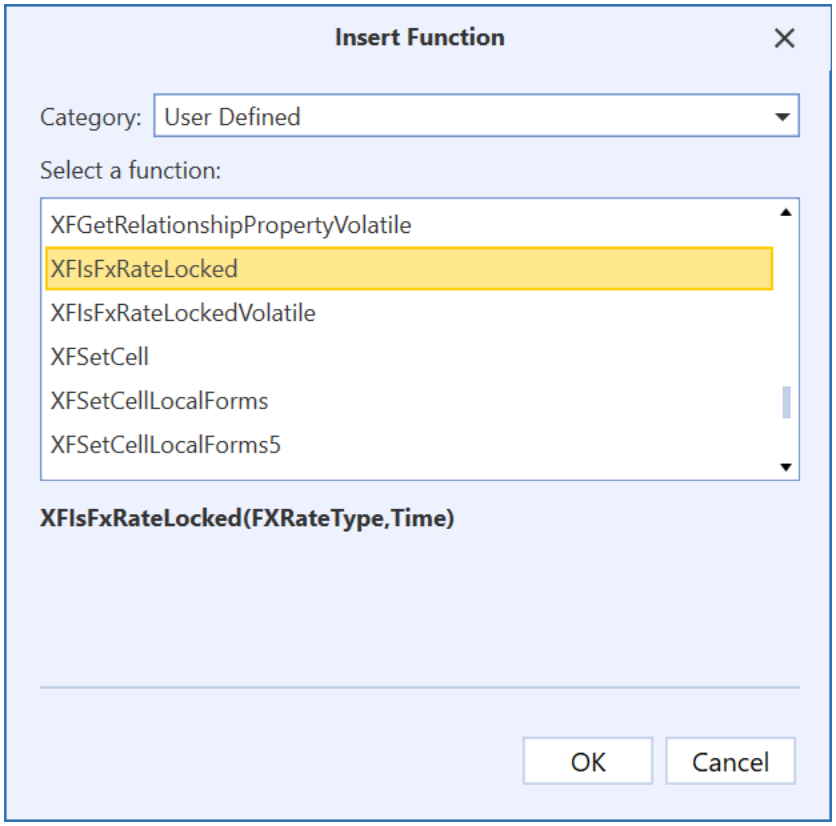
The following BRApis allow for automation and dashboard development: ISFxRateLocked, LockFxRate, and UnlockFxRate.



Spreadsheet or Excel Functions

Retrieve the status in reports. In spreadsheet or Excel, you can go to functions and select XFisFXRateLocked for your rate sheets.

1. From the **Application** tab, go to **Tools** > **Spreadsheet**.
2. Select the **OneStream** tab.
3. Select the **Insert Function** button.
4. From the **Category** drop-down menu, select **User Defined**.
5. Under **Select a function:**, select **XFIsFxRateLocked**.



6. Select the **OK** button. The Function Arguments window opens.
7. Enter a value in **FxRateType**.
8. Enter a value in Time.
9. Select the **OK** button. The spreadsheet will populate with the new data.

> **NOTE:** If you change the lock or unlock rate types in the spreadsheet, when you refresh the application and return to the Rates grid, the data will be updated to reflect your changes in the spreadsheet.

Member Filters

A Member Filter can be used to filter the data by creating a list of restricted members. Member Filters contain multiple member scripts separated by commas.

A member script is a brief statement typed to query one defined set of dimension members. Members can be specified for any or all dimensions, and the primary dimension can also specify a Member Expansion formula, for example,, .Descendants..

Here is an example of a simple member script that returns the year 2012:

```
T#2012
```

Here is an example of a member script with a Member Expansion that returns all the Income Statement accounts.

```
A#[Income Statement].Descendants
```

If one or a few dimensions in the member script are qualified, the remaining dimensions are pulled either from the Cube View POV, the Global POV, or workflow, time, and scenario. Separate each dimension qualified in the member script with a colon. Here is an example of a fully qualified member script:

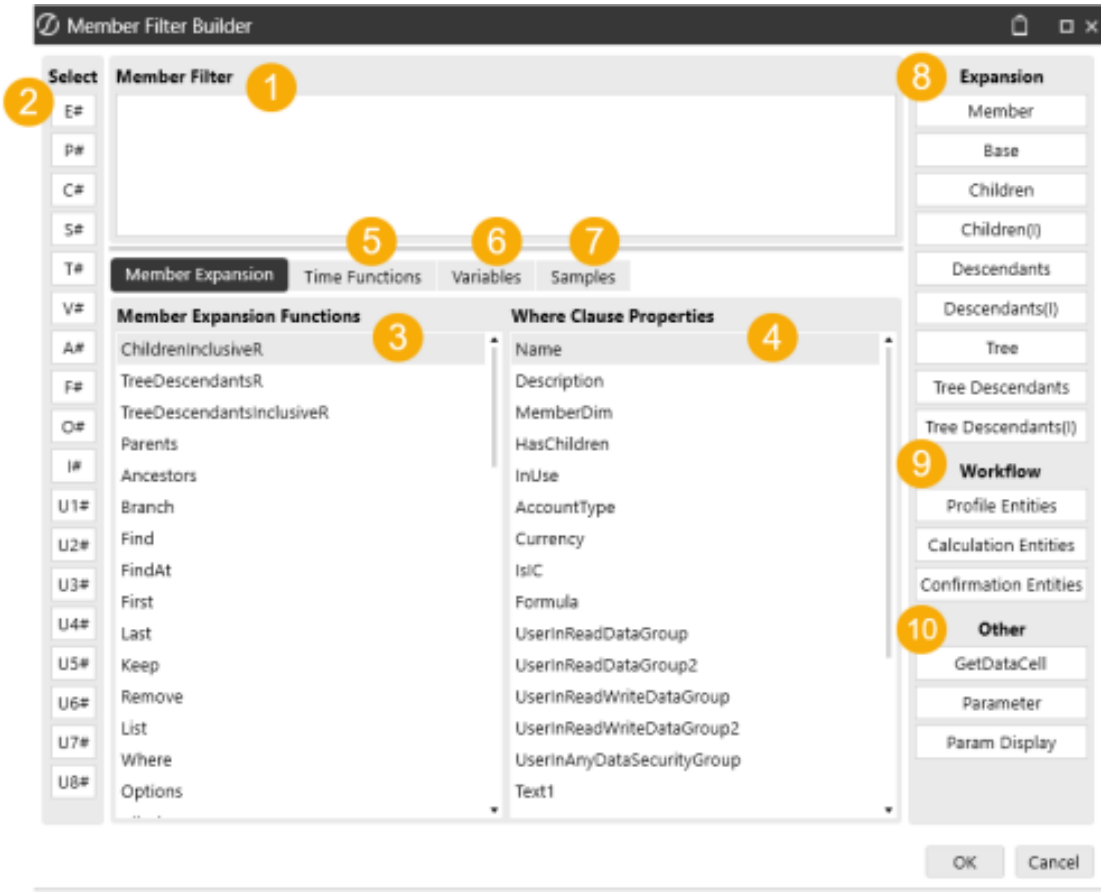
```
Cb#GolfStream:E#Houston:P#Texas:C#Local:S#Budget:T#2012M3:V#YTD:A#60000:F#None:O#Forms:IC#No
ne: U1#Sales: U2#HybridXL: U3#Northeast: U4#SportsCo: U5#None: U6#None: U7#None: U8#None
```

Here is an example of a Member Filter made up of three different member scripts that returns the actual, budget and forecast scenarios.

```
S#Actual, S#Budget, S#Forecast
```

Member Filter Builder

The dialog box makes it simple to build complex Member Filters without having to remember or look up the proper syntax. This dialog box is embedded in areas such as Cube Views and the Dimension Library.



1. **Member Filter**: Location where the Member Filter is built. Enter text in this section or use the other items in the dialog box to enter information into the field.
2. Dimension Tokens: There is a button for each dimension that launches the appropriate selection dialog box.

   > **NOTE:** When searching for a member in the Select Member window, you can search by its name, alias, or description.

3. **Member Expansion Functions**: Double-click a Member Expansion to add it to the Member Filter.
4. **Member Expansion Where Clause**: If you select the Member Expansion Where, use the Where Clause properties to complete the expression.

   > **Example:** UD2#AllProducts.Children.Where(Name Contains Clubs)

5. **Time Functions**: Only applies to the Time dimension, such as T#POVPrior1. Double-click a Time function to add it.
6. **Substitution Variables**: Double-click on a system-wide substitution variable to add it.
7. **Samples**: Refer to this tab for example syntax for complex queries and calculations, including how to build Member Expansions, Where Clause expressions, time expressions, calculations (GetDataCell, and column and row expressions), custom member list expressions, and XFBR, XFCell, and XFGetMemberProperty.
8. **Expansion**: Commonly used Member Expansions added to the end of filters. Double-click the expansion to add it.
9. **Workflow**: These buttons are commonly used Workflow Member Expansions used in Cube Views that point to a report, form, or dashboard and are affiliated with a specific Workflow Profile.

10. **Other**: Commonly used Member Filter functions that enable you to create calculated columns and rows or use a custom parameter to store member lists.

Member Script Abbreviations and Examples

There are 18 dimensions that can be used to filter data. This section briefly explains each dimension and gives an example.

See "Cube Dimension"s in the *Understanding the Financial Model Guide* for a full description of each Dimension.

Brackets are optional around member names, but mandatory if the member name contains a space, period, comma, or semicolon.

Cube
Cb#GolfStream
Cube being referenced.

Entity
E#Houston
Entity being referenced.

Parent
P#Texas
Parent of the entity being referenced. This is important because entities can be rolled up into multiple hierarchies.

Consolidation
C#Local
Member of the Consolidation dimension being referenced.

Scenario

S#Actual

Scenario being referenced.

Time

T#2012Q1

Specific time period in focus. There are several selections available relative in nature to certain boundaries like the time period set in the POV, the selected workflow time period, or the global time period.

More of these constants are explained in the Member Script Constants section later in this document.

T#POV

Returns the current period of the selected year in the cube POV.

T#W1

Returns the first week of the selected year in the cube POV. This only applies to weekly applications.

T#M1

Returns the first month of the selected year in the cube POV.

T#Q1

Returns the first quarter of the selected year in the cube POV.

T#H1

Returns the first half of the selected year in the cube POV.

View

V#YTD

View of the requested information.

Account
A#60000
Account being referenced.

Flow
F#BegBal
Flow member being referenced. There is a dedicated Flow dimension.

Origin
O#Import
Origin of the data being referenced and if it was imported, entered as a journal, or entered into a form.

UD1 to UD8
U2#[Fairway Woods] or U3#TotalRegions
Members from the User Defined dimensions being referenced. If none of these dimensions are being used, they still need to be qualified in the POV, in a Cube View, or in the member script to get a proper intersection. Each User Defined dimension has a member called None or the reserved word All that can be used,for example, U5#All, in some areas of the product such as Intercompany Reporting settings.

Member Script Constants

Member script constants are used as a point of reference when querying data for members and data. These are the constant types:

- **Point of View (POV)**: Select members based on their literal or relative position to the selected main Point of View (POV) or the Cube View POV. The main POV settings are available under the Application tab > Tools > Application Properties. Call out a dimension to pull its member value from what is selected in the POV in this way: E#POV, C#POV, T#POV, V#POV, A#POV,... (all Dimension types).
- **Workflow (WF)**: Refer to the time period or scenario currently selected in the workflow module. Examples are T#WF and S#WF.
- **Global** : The application has a Global Time and Global Scenario setting found under the Application tab > Tools > Application Properties. These settingsare optional and can be referenced, for example,, T#Global and S#Global.. An example of using this as a reference is to build a Cube View and have global time and everything before it display in columns and the actual scenario and everything after it display as the forecast scenario.

Member Expansion Functions

Additional commands can be added at the end of a member script to help expand the presented results. For example, E#US will return just the US entity, while E#US.Descendants will return all of the child entities that fall under the US entity, for example, California, Texas, or New York.

Member Expansions

To illustrate these expansion concepts, the following examples will use a portion of the entity structure from GolfStream Corporation, which is a golf supply company. This is the management organization, but there is also a geographical organization where Houston rolls up to Texas; Carlsbad rolls up to California; and Frankfurt rolls up to Europe.

- Total GolfStream
    - Clubs
        - NA Clubs
            - Canada Clubs
                - Montreal
                - Quebec City
            - US Clubs
                - Augusta
                - Carlsbad
                - Houston
        - Frankfurt
    - Golf Balls
        - Europe Golf Balls
        - NA Golf Balls
    - Accessories & Apparel
    - Course Management
    - Corporate

Add these expansions after a member name to return the desired members like this: E#Houston.Ancestors.

> **TIP:** Use brackets, [ ], to reference any name with a space. For example, E#[NA Clubs].

Member

This lists the member requested. E#Frankfurt.Member is the same as stating E#Frankfurt. This expansion can be combined with another Member Expansion to create an inclusive list. This is an example of using member for a children inclusive list:

E#[Total GolfStream].Member.Children returns the same as E#[Total GolfStream].ChildrenInclusive:

- Total GolfStream
    - Clubs
    - Golf Balls
    - Accessories & Apparel
    - Course Management
    - Corporate

This is an example of using member for a parent inclusive list:

E#Clubs.Member.Parents returns Clubs and its parent, Total GolfStream.

This is an example of using member for an ancestors inclusive list:

E#[NA Clubs].Member.Ancestors returns NA Clubs and its ancestors, Clubs and Total GolfStream.

Base

E#[NA Clubs].Base queries only the base-level entities and excludes any other aggregate members:

- Montreal
- Quebec City
- Augusta

- Carlsbad
- Houston

Children
E#Clubs.Children returns the first-level children in a flat list:

- NA Clubs
- Frankfurt

ChildrenInclusive
E#[NA Clubs].ChildrenInclusive returns both NA Clubs and its children:

- NA Clubs
    - Canada Clubs
    - US Clubs

This can be chained together with another .Children statement to see the next level as well. This may also be referred to as Children.Children elsewhere in the documentation.

E#[NA Clubs].ChildrenInclusive.Children returns:

- NA Clubs
    - Canada Clubs
        - Montreal
        - Quebec City
    - US Clubs

- Augusta
- Carlsbad
- Houston

Descendants

E#[NA Clubs].Descendants lists every member under NA Club in a flat, nonhierarchical list excluding NA Clubs:

- Canada Clubs
- Montreal
- Quebec City
- US Clubs
- Augusta
- Carlsbad
- Houston

DescendantsInclusive

E#[US Clubs].DescendantsInclusive lists every member under US Clubs in a flat, nonhierarchical list including US Clubs:

- Augusta
- Carlsbad
- Houston Heights
- South Houston
- Houston
- US Clubs

Tree

E#[US Clubs].Tree returns the specified member and all members below it in a hierarchy:

This expansion automatically includes expandable rows.

- US Clubs
    - Augusta
    - Carlsbad
    - Houston

TreeDescendants

E#Clubs.TreeDescendants lists every member under Clubs in a hierarchical tree excluding Clubs:

- NA Clubs
    - Canada Clubs
        - Montreal
        - Quebec City
    - US Clubs
        - Augusta
        - Carlsbad
        - Houston
- Frankfurt

TreeDescendantsInclusive

E#[US Clubs].TreeDescendantsInclusive lists every member under US Clubs in a hierarchical tree including US Clubs:

- US Clubs
    - Augusta
    - Carlsbad
    - Houston

Parents

E#Houston.Parents returns the direct parents of the given member regardless of how many hierarchies to which the member belongs:

- US Clubs (from the management organization)
- Texas (from the geographical organization)

The parent is derived from the Cube View POV setting by default. If a member is used in multiple hierarchies, specify a specific parent using the following syntax to override it: E#Houston.Base:P#USClubs or E#Houston.TreeDescendants:P#Texas

> **NOTE:** P# works differently when using it with an expandable tree filter because that filter processes the children as individual queries, so the P# will only apply to the top-level members. Use E#Houston.TreeDescendants:P#Texas instead, or specify a parent on the Cube View POV settings.

Ancestors

E#[NA Clubs].Ancestors returns all members above NA Clubs:

- Total GolfStream
- Clubs

Branch

Expand multiple members of a given expansion by finding specific items and then performing additional expansions on those items.

E#[Total GolfStream].Children.Branch(Find([Clubs]).Children, Find(Golf Balls).Children) returns the children of Total GolfStream, it then finds the children of Clubs and Golf Balls and returns them in the hierarchy as well.

- Total GolfStream
    - Clubs
        - NA Clubs
        - Frankfurt
    - Golf Balls
        - Europe Golf Balls
        - NA Golf Balls
    - Accessories & Apparel
    - Course Management
    - Corporate

This is also used with Quick Views to expand several hierarchies at one time.

Find

Find will apply Member Expansions to a nested subset of results:

A#[Income Statement].Descendants.Find(64000).Children will return all of the descendants of the Income Statement as a flat list, but will find account 64000 (that is, Earnings Before Taxes in this example) and then indent and present that account's children.

FindAt

This returns a specific member of a given expansion using a zero-based position index and performing an additional expansion on the specific item. Refer to the Samples tab in the Member Filter Builder for an example of the syntax.

E#[Total GolfStream].Children.FindAt(1).Children returns the children of Total GolfStream, locates the member in the first position, and returns its children as well.

- Total GolfStream
    - Clubs
    - Golf Balls
        - Europe Golf Balls
        - NA Golf Balls
    - Accessories & Apparel
    - Course Management
    - Corporate

First

First will find the first items in the list of results and allow additional Member Filters to be applied:

E#[Total GolfStream].Children.First.Children will list all of the children of Total GolfStream, go to the first entity (Clubs), then show its children:

- Clubs
    - NA Clubs
    - Frankfurt
- Golf Balls
- Accessories & Apparel
- Course Management
- Corporate

Last
Similar to First, Last will find the last items in the list of results and allow additional Member Filters to be applied.

Keep
Keep will search the results from a Member Filter and only keep certain values:

E#[Total GolfStream].Children.Keep(Clubs, [Golf Balls]).Children will list all of the first-level children of Total GolfStream, only keep Clubs and Golf Balls, then show their children:

- Clubs
    - NA Clubs
    - Frankfurt
- Golf Balls
    - Europe Golf Balls
    - NA Golf Balls

Remove
This will remove some of the members from the results:

E#[Total GolfStream].Children.Remove(Corporate).Find(Clubs).Children.Find([NA Clubs]).Children will list several entities and then remove the Corporate entity from the results:

- Clubs
    - NA Clubs
        - Canada Clubs
        - US Clubs
    - Frankfurt
- Golf Balls

- Accessories & Apparel
- Course Management

List

This will create a list of specific members:

A#Root.List(63000, 64000) will return these two accounts.

An indent level can also be specified when defining a list of members.

E#[Total GolfStream].List(Clubs.IndentLevel(0), [Golf Balls].IndentLevel(2), [Course Management].IndentLevel(1))

- Clubs
    - Golf Balls
        - Course Management

Where

The Where clause in a Member Filter allows further qualification of the results. It adds an additional level of filtering after the list of members has been retrieved. See the following use cases.

You can filter members where a property is a certain value:

- X#MyMember.Where(PropertyName = 'Value')
- E#[Total GolfStream].Descendants.Where(IsIC = True)

You can add other Member Expansions, for example, .Children, before and after the Where clause:

- X#MyMember.MemberExpansion.Where(PropertyName = 'Value').MemberExpansion
- A#Sales.Descendants.Where(Name Contains '60000').Children

You can filter for multiple values using detailed conditions within .Where() using AND, OR with additional parentheses:

- X#MyMember.Where((PropertyName = 'Value') OR (PropertyName = 'Value'))
- A#60999.Descendants.Where((Name Contains '60000') OR (Name Contains '41000'))
- E#Root.Children.Find(US).Children.Find(Michigan, Texas)Children.Where((Name = Flint) OR (Name = Rochester) OR (Name = Dallas)) returns Michigan and Texas and only specific child city entities.

You can use the IN or NotIN qualifiers to return members in or not in a list:

- X#MyMember.Where(PropertyName In '60000')
- A#60999.Descendants.Where(Name In '60000', '41000') returns 60000 as it is a descendant of 60999. 41000 is not returned as it is not a descendant of 60999.
- E#Root.Children.Find(US).Children.Find(Michigan, Texas)Children.Where(Name In Flint, Rochester, Dallas) returns Michigan and Texas and only a few child entities.

You can also add a filter to another Member Expansion, for example, .descendants, with or without .Where:

- A#MyMember.MemberExpansion(MyPropertyName='MyValue')
- A#Sales.Descendants.Where(Name = '60000')
- A#Sales.Descendants(Name = '60000')

You can combine criteria within the Where clause, such as filtering for different properties:

A#Sales.Descendants.Where((HasChildren = True) AND (Name Contains '60000'))

You can use operators other than = within the Where clause. The following list includes supported operators:

- >=
- <=
- <>
- =

- >
- <
- StartsWith
- Contains
- DoesNotContain
- EndsWith

Operators with greater than or less than are not applicable to the text-based fields:

- >=
- <=
- >
- <

Options

Use options to reference sub-cubes and specify how dimensions should be processed. Options must immediately follow the expansion function it is being used for.

For example, A#MyAccount.Base might return a different list when looking at an extended dimension associated with a different cube. Therefore, the cube and scenario type can be specified using options:

A#MyAccount.Base.Options(Cube=CubeName, ScenarioType=ScenarioTypeName, MergeMembersFromReferencedCubes=False)

The following example returns the children for account 19999 from the Total GolfStream cube and Actual scenario type:

A#19999.Children.Options(Cube=[Total GolfStream], ScenarioType=Actual, MergeMembersFromReferencedCubes=False)

If you drill to base members in a Cube View or Quick View, you can use options to ensure you are returning the desired members. For example, A#60000.Base could return different members due to extended dimensions. Using options ensures you are returning the desired members:

- A#60000.Base.Options(Cube=Houston, ScenarioType=Actual, MergeMembersFromReferencedCubes=False)
- A#60000.Base.Options(Cube=NewYork, ScenarioType=Actual, MergeMembersFromReferencedCubes=False)

The following example returns tree descendants inclusive of the account based on the workflow cube and Scenario Type property of the workflow:
A#MyAccount.TreeDescendantsInclusive.Options(Cube=|WFCube|, ScenarioType=XFMemberProperty(DimType=Scenario,Member=|WFScenario|, Property=ScenarioType),MergeMembersFromReferencedCubes=False)

Combined Expressions

You can stack multiple Member Expansions to display additional members, add conditions, remove members, and more.

This example shows Children of Children, which display the children members and their children members.

E#[Total GolfStream].Children.Children returns:

- Clubs
    - NA Clubs
    - Frankfurt
- Golf Balls
    - Europe Golf Balls
    - NA Golf Balls
- Accessories & Apparel
- Course Management
- Corporate

This example uses the List and Find expressions to display base and parent members:

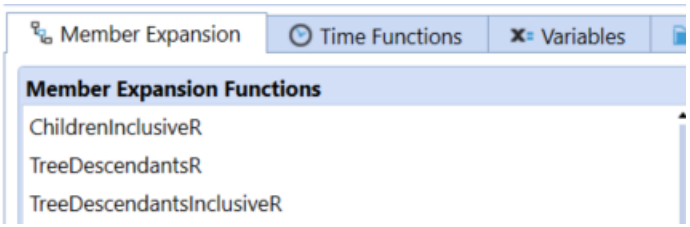
E#[Total GolfStream].List(Clubs,Corporate,Frankfurt).Find(Clubs).Base.Find(Carlsbad).Parents returns:

- Clubs
- Montreal
- Quebec City
- Augusta
- Carlsbad
- California
- US Clubs
- Total Product Rollup
- Houston Heights
- South Houston
- Frankfurt
- Corporate
- Frankfurt

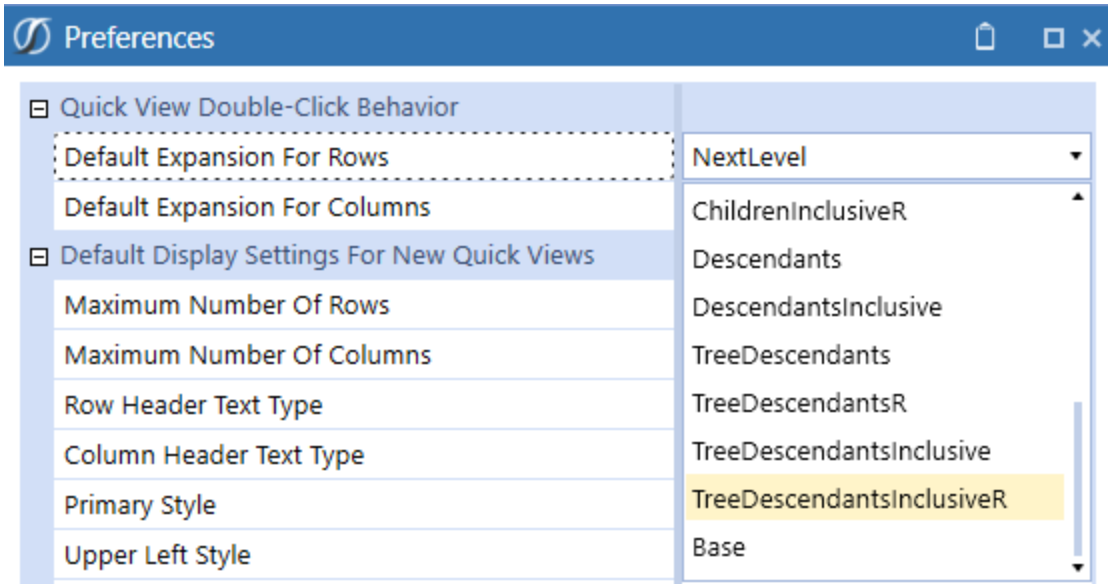
Reverse Order Member Expansions

Functionality to support reversing the direction of the order of results from certain member filters are supported in Cube Views and Quick Views. These reverse order selections include ChildrenInclusiveR, TreeDescendantsR, and TreeDescendantsInclusiveR.

In Cube Views, this functionality can be set in the Member Filter Builder using the Member Expansion Functions tab for Rows (and/or) Columns.



In Quick Views, this can be set in Preferences under the Quick View Double-Click Behavior section in the Default Expansion For Rows or Columns.



This example will demonstrate the TreeDescendantsR being used in a Quick View. The following Gross Income view has been created using Next Level expansion, which is available in a Quick View:

	Jan 2018	Feb 2018
61000 - Gross Income	27,462,816	28,236,051
60999 - Net Sales	56,155,448	56,119,871
60000 - Operating Sales	57,982,342	57,944,272
2000_100 - Third Party Sales	50,945,236	50,985,236
2000_200 - OEM Sales	4,606,157	4,536,127
2000_300 - Subassembly Sales	382,840	382,800
2000_400 - Parts Sales	2,048,108	2,040,108
60100 - IC Sales	15,000	15,990
60200 - Returns & Allowances	1,841,894	1,840,391
2200_100 - Third Party Returns & Allowances	728,117	727,974
2200_200 - OEM Returns & Allowances	37,372	37,312
2200_300 - Subassembly Returns & Allowances	95,952	95,052
2200_400 - Parts Returns & Allowances	980,454	980,054
60250 - Other Outside Sales		
43000 - Cost of Goods Sold	28,692,632	27,883,820
41000 - Operating Cost of Goods Sold	28,677,632	27,868,820
42000 - IC Cost of Goods Sold	15,000	15,000

To change the double-click behavior, under the Administration menu,select Preferences . In the Quick View Double-Click Behavior section, select the drop-down list for Default Expansion For Rows and select TreeDescendantsR. The result is the reverse direction of the originating account on expansion for rows when using the double-click:

	Jan 2018	Feb 2018
2000_100 - Third Party Sales	50,945,236	50,985,236
2000_200 - OEM Sales	4,606,157	4,536,127
2000_300 - Subassembly Sales	382,840	382,800
2000_400 - Parts Sales	2,048,108	2,040,108
60000 - Operating Sales	57,982,342	57,944,272
60100 - IC Sales	15,000	15,990
2200_100 - Third Party Returns & Allowances	728,117	727,974
2200_200 - OEM Returns & Allowances	37,372	37,312
2200_300 - Subassembly Returns & Allowances	95,952	95,052
2200_400 - Parts Returns & Allowances	980,454	980,054
60200 - Returns & Allowances	1,841,894	1,840,391
60250 - Other Outside Sales		
60999 - Net Sales	56,155,448	56,119,871
41000 - Operating Cost of Goods Sold	28,677,632	27,868,820
42000 - IC Cost of Goods Sold	15,000	15,000
43000 - Cost of Goods Sold	28,692,632	27,883,820
61000 - Gross Income	27,462,816	28,236,051

Time Member Expansions

These Member Expansions can be added onto a regular time-based member script. They can be used in a Cube View or in the Form Template Time Filter for the Complete Form option, so only the time periods needed will be displayed.

AllPriorInYear

This returns all the time periods before the specified time period excluding the specified time for its frequency. T#2012M6.AllPriorInYear, T#WF.AllPriorInYear return the previous periods in that year but not the specified period.

AllPriorInYearInclusive

This returns all the time periods before the specified time period including the specified time for its frequency. T#2012M4.AllPriorInYearInclusive returns periods 2012M1 through 2012M4.

AllNextInYear

This returns all the time periods after the specified time period not including the specified time for its frequency. T#2012M8.AllNextInYear returns periods 2012M9 through 2012M12.

AllNextInYearInclusive

This returns all the time periods before the specified time period including the specified time for its frequency. T#2012M8.AllNextInYearInclusive returns periods 2012M8 through 2012M12.

Weeks

This returns all the weeks associated with the specified time filter. T#2017M2.Weeks returns all the weeks in M2. T#2017.Weeks returns all the weeks in 2017. If the POV Time is 2017M7, T#POV.Weeks returns all the weeks in M7.

Months

This returns all the months associated with the specified time filter. T#2017.Months returns all the months in 2017. If the POV Time is set to 2017Q2, T#POV.Months returns all the months in Q2.

MonthsInQuarter

This returns the months in the specified quarter. If the Workflow period is 2012M2, T#WF.MonthsInQuarter will return 2012M1, 2012M2, and 2012M3.

MonthsInHalfYear

This returns the months in the half year of the specified period. If the Global period is 2012M2, T#Global.MonthsInHalfYear will return 2012M1 through 2012 M6.

MonthsInYear

This returns all of the months in the year of the specified period. If the POV period is 2012M7, T#POV.MonthsInYear will return 2012M1 through 2012M12.

Quarters

This returns the quarters associated with the specified year. T#2017.Quarters will return 2017Q1, 2017Q2, 2017Q3, and 2017Q4.

QuartersInHalfYear
This returns the quarters in the half year of the specified period. If the Workflow period is 2012M3, T#WF.QuartersInHalfYear will return 2012Q1 and 2012Q2.

QuartersInYear
This returns all of the quarters in the year of the specified period. T#2012M7.QuartersInYear will return 2012Q1, 2012Q2, 2012Q3, and 2012Q4.

HalfYears
This returns the half years associated with the specified year. T#2017.HalfYears returns 2017H1 and 2017H2.

HalfYearsInYear
This returns the half years in the year of the specified period. If the Global period is 2012M5, T#Global.HalfYearsInYear will return 2012H1 and 2012H2.

Prior 1-Prior 12
This returns the prior periods in relation to the specified period. T#2010M12.Prior12 will return the 12 months prior to 2010M12 not including 2010M12. T#2017W40.Prior12 will return the 12 weeks prior to 2017W40 not including 2017W40.

Next 1-Next 12
This returns the next periods in relation to the specified period. T#2010M12.Next12 will return the 12 months after 2010M12 not including 2010M12. T#2017W40.Next12 will return the 12 weeks after 2017W40 not including 2017W40.

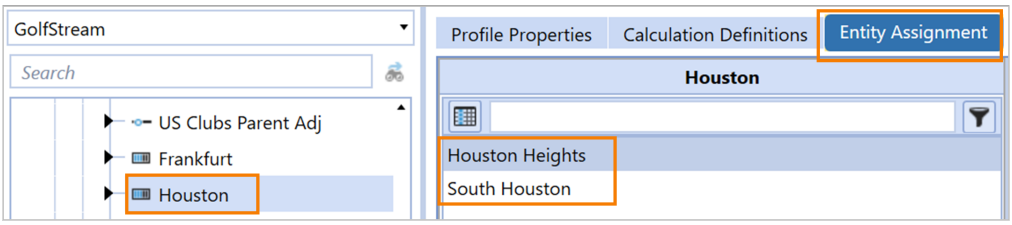
Workflow Member Expansions

These are used in Cube Views that are used in reports, forms, or dashboards and presented during the workflow process. These method queries can also be used when setting pop-up parameters when dashboards and reports are run.

WFProfileEntities (Entity Dimensions Only)

This returns all entities associated with this Workflow Profile.

In the following example, the Houston Workflow Profile is assigned the Houston Heights and South Houston entity members. If the Workflow POV is set to Houston, then E#Root.WFProfileEntities returns Houston Heights and South Houston.



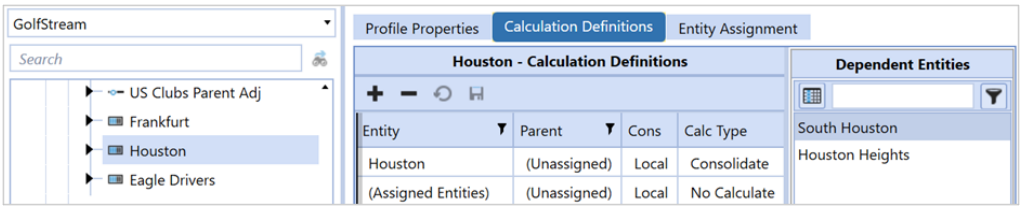
WFCalculationEntities (Entity Dimension Only)

This filters to the entities encompassed within the Workflow Profile Calculation Filter, which are calculated, translated, and consolidated when the user selects Process during workflow.

When used as a Member Expansion, this expression returns all entities defined as part of the calculation definitions for the selected Workflow Unit.

In the following example, the Houston Workflow Profile defines Houston and the assigned entities, South Houston and Houston Heights, within calculation definitions.

E#Root.WFCalculationEntities returns Houston, South Houston, and Houston Heights.

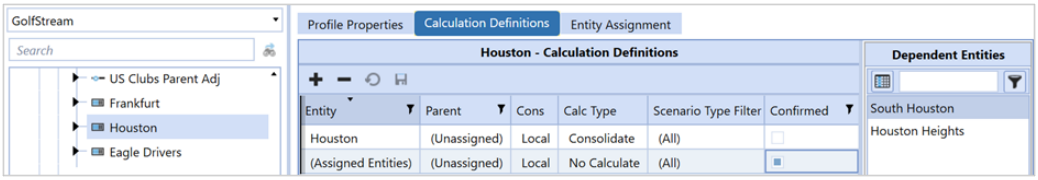


WFConfirmationEntities (Entity Dimensions Only)

This filters to the entities encompassed within the Workflow Profile Confirmation Filter, which are checked when the user selects Confirm during workflow.

In the following example, the Houston Workflow Profile has the Confirmed Filter selected for the assigned entities, South Houston and Houston Heights, within the calculation definitions.

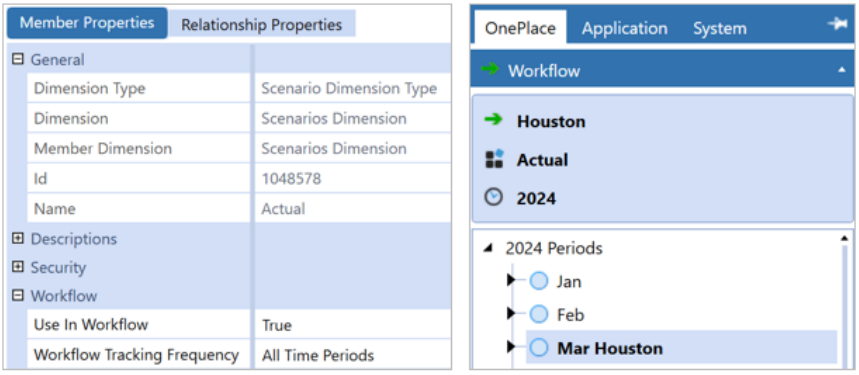
E#Root.WFConfirmationEntities returns South Houston and Houston Heights.



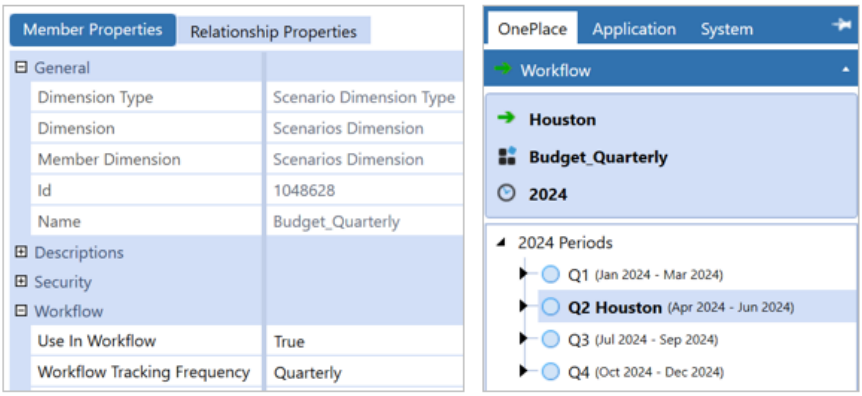
WFTimePeriods

This returns the time periods associated with the Workflow Profile. When using the standard workflow tracking frequency, it returns the single time period associated with the selected Workflow Unit. When using other workflow tracking frequencies, it returns the range of time periods for the selected Workflow Unit between WFStartTime and WFEndTime.

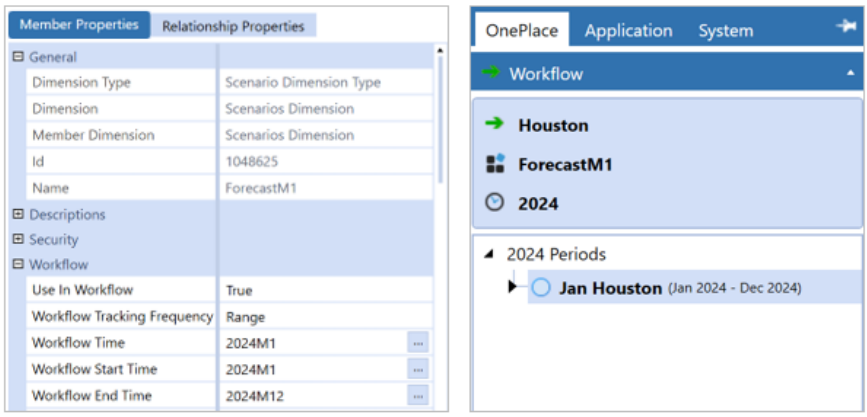
In the following example, the Actual scenario member has the Workflow Tracking Frequency property set to All Time Periods in a monthly application. If the Workflow POV is Actual and 2024M3, then T#Root.WFTimePeriods returns 2024M3.



In the following example, the Budget_Quarterly scenario member has the Workflow Tracking Frequency property set to Quarterly. If the Workflow POV is Actual and 2024Q2, then T#Root.WFTimePeriods returns 2024M4, 2024M5, and 2024M6.



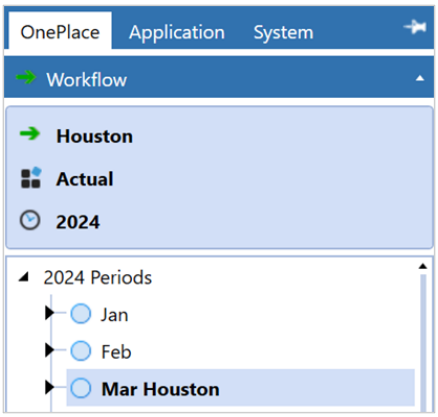
In the following example, the ForecastM1 scenario member has the Workflow Tracking Frequency property set to Range, the Workflow Start Time property set to 2024M1, and the Workflow End Time property set to 2024M12. If the Workflow POV is ForecastM1 and 2024M1, then T#Root.WFTimePeriods returns 12 time members, 2024M1 through 2024M12.



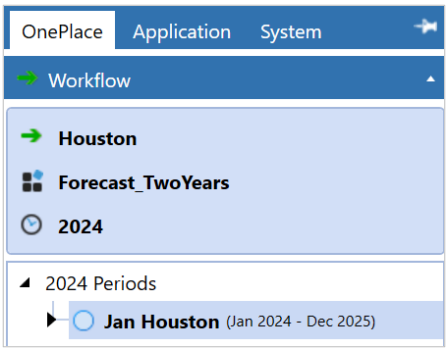
WFCalculationTimePeriods

This returns the time periods needed to be calculated for a Workflow Unit. For example, if a range-based Workflow Unit spans two years, it would return the last period of the first year and the last period of the second year. Running a calculation using those two time periods would cause all 24 time periods to be calculated because the calculation engine automatically calculates all prior periods in a year. WFCalculationTimePeriods is intended to be used in data management steps and business rules.

In the following example, the Actual scenario member has calculations monthly. If the Workflow POV is Actual and 2024M3, then T#Root.WFCalculationTimePeriods returns 2024M3.



In the following example, the Forecast_TwoYears scenario member has a range Workflow Unit that spans two years from 2024M1 to 2025M12. If the Workflow POV is Forecast_TwoYears, then T#Root.WFTimePeriods returns 2024M12 and 2025M12 as calculations are needed for each year.



WFChannelMembers (User Defined Dimensions Only)
This returns the UD dimension members associated with the workflow channel that is associated with this Workflow Profile. This can be a one-to-many relationship. For example, UD1 represents cost centers for this application and three UD1 members are associated with the workflow channel called Engineering. If the Workflow Profile in the workflow view is associated with the workflow channel Engineering, UD1#Root.WFChannelMembers will return those three UD1 Members.

Member Expansion Where Clause

The Where clause adds an additional level of filtering after the list of members has been retrieved. See Member Expansions Where.

The following examples are Where clauses that filter for common member properties. You can filter on additional member properties that are not in these examples.

Name (Example: (Name StartsWith ‘Sales’))
Filter for members based on their Name property.
A#Sales.Descendants.Where(Name Contains '6000') filters the member list to include only member names containing ‘6000.’ This could also be written as:

A#Sales.Descendants(Name StartsWith 6000)

The following examples are based on this entity hierarchy:

- Total GolfStream
    - Clubs
        - NA Clubs
        - Frankfurt
    - Golf Balls
        - Europe Golf Balls
        - NA Golf Balls
    - Accessories & Apparel
    - Course Management
    - Corporate

E#[Total GolfStream].Children.Where((Name Contains 'Balls') OR (Name = 'Clubs'))

This returns the children of Total GolfStream if the member name contains Balls or equals Clubs:

- Clubs
- Golf Balls

Other Member Expansions can also be added after Where clause expansions.

E#[Total GolfStream].Member.Children.Where((Name Contains Balls) OR (Name = Clubs)).Children

This returns Total GolfStream due to E#[Total GolfStream].Member. It also returns its children where Name contains Balls or equals Clubs and their children:

- Total GolfStream
    - Clubs
        - NA Clubs
        - Frankfurt
    - Golf Balls
        - Europe Golf Balls
        - NA Golf Balls

Description (Example: (Description Contains 'Cost'))
Filter for members based on their Description property.
A#Root.Descendants.Where(Description Contains 'Cost') filters the member list to include only accounts with Descriptions containing 'Cost.'

MemberDim
Filter for members within certain dimensions.
A#MyAccount.Base.Where(MemberDim = DimensionName)
A#[Income Statement].Where(MemberDim = |WFAccountDim|) filters the member list to the accounts within the workflow's Account dimension.
U2#Top.Descendants.Where(MemberDim = 'HoustonProducts') filters the member list to the products in the HoustonProducts UD2 dimension.

HasChildren (Example: (HasChildren = True))
Filter for members with or without children.
A#Root.Children.Where(HasChildren = False) filters the member list to include only the members without children.

A#Root.Children.Where(HasChildren = True) filters the member list to include only the members with children.

InUse
Filter for Entity, Account, Flow, and UD1-UD8 members in use or not in use.
A#Root.Children(InUse = True) filters the results to only include the members with the In Use member property set to True.

AccountType
Filter Account members based on their account type.
A#Root.Base(AccountType = Revenue) filters the results to only the members with the account type of Revenue.

Currency
Filter for entity members of a specific currency.

E#[Total GolfStream].Descendants.Where(Currency = EUR) filters the results to only include descendants of Total GolfStream with the currency of EUR.

IsIC
Filter for intercompany members based on the Is IC Entity member property and the Is IC Account member property.

E#[Total GolfStream].Descendants.Where(IsIC = True) filters the results to only include descendants of the Total GolfStream with the Is IC Entity member property set to True.

A#Sales.Descendants.Where(IsIC = True) filters the results to only include descendants of Sales with the Is IC Account member property set to True.

Formula
Filter for Scenario, Account, Flow, and UD1-UD8 members with Member Formulas. F#Root.TreeDescendants(Formula <> '') filters out members with an empty value for the Formula property so that only members with Member Formulas display.

User In Security Group Where Clauses
These Where clauses will restrict the list to only members the user is allowed to read and write to based on the user security settings or expressions can be used to refer to multiple properties:

E#[Total GolfStream].Base.where((UserInReadDataGroup = True) or (UserInReadWriteDataGroup = True))

Choices:

- UserInReadDataGroup
- UserInReadDataGroup2
- UserInReadWriteDataGroup
- UserInReadWriteDataGroup2
- UserInAnyDataSecurityGroup

TextN (Text1 through Text8)
This uses the TextN property in the Scenario, Entity, Account, Flow, and UD1 to UD8 dimensions. There are eight TextN properties in these types of Members that can be used for any need. These Where clauses will query those fields for their values:

- A#Root.Children(Text1 = 'Blue')
- A#Root.Children(Text2 StartsWith 'Strategic')
- A#Root.Children(Text3 Contains 'Tax')
- A#Root.Children(Text4 EndsWith 'Old')

- A#Root.Children(Text5 <> '') filters out the members that have an empty value for the Text5 property.

Time Functions

Time Functions and Pov Member References

When using the POV as a point of reference, use the following constants in the member scripts. In these constants, N refers to the number of periods desired.

T#POV (All Dimension Types)
Select members based on their literal or relative position to the selected main Point of View (POV) or the Cube View POV. The main POV settings are available under the OnePlace. Add a dimension to pull its member value from what is selected in the POV in this way: E#POV, C#POV, T#POV, V#POV, A#POV, and all other dimension types.

If a Cube POV is 2024M1, then a Cube View with T#POV returns 2024M1.

If the POV is 2024, T#POV returns 2024.

T#PovYear (Uses Time Member Suffix, Example: T#PovYearM1)
This returns a specific time period within a current POV year defined by the currently selected POV period.

- If POV is 2024M1, T#PovYear returns 2024.
- If POV is 2024M1, T#PovYearM3 returns 2024M3.

You can combine T#PovYear with expansions.

- If the POV is 2024M3, T#PovYear.HalfYears returns the 2024H1 and 2024H2.
- If the POV is 2024M3, T#PovYear.Quarters returns all quarters for 2024.

- If the POV is 2024M3, T#PovYear.Months returns all months for 2024.
- If the POV is 2024M3, T#PovYear.Weeks returns all weeks for 2024 in weekly applications.

T#PovPriorYear (Uses Time Member Suffix, Example: T#PovPriorYearM12)
This returns a specific time period within the prior year defined by the currently selected POV period.

- If POV is 2024M1, T#PovPriorYear returns 2023.
- If POV is 2024M1, T#PovPriorYearm12 returns 2023M12.

T#PovPriorinYear1 (Uses Integer Suffix, Example: T#PovPriorInYear1)
This returns the relative prior period from the currently selected POV period, but within the same year.

- If the current period is 2024M8, T#PovPriorInYear4 returns 2024M4.
- If the current period is 2024M8, T#PovPriorInYear10 returns the first period in the year, 2024M1, because there are not 10 periods prior to 2024M8 within the same year.

T#PovPrior1 (Uses Integer Suffix, Example: T#PovPrior1)
This returns the relative prior period from the currently selected POV period. If the POV is 2024M1, T#PovPrior3 returns 2023M10.

T#PovNextInYear1 (Uses Integer Suffix, Example: T#PovNextInYear2)
This returns the relative next period in the future from the currently selected POV period, but within the same year.

- If the current period is 2024M4, T#PovNextInYear5 returns 2024M9.
- If the current period is 2024M4, T#PovNextInYear10 returns the last period in the year 2024M12 becasue there are not 10 periods after 2024M4 in within the same year.

T#PovNext1 (Uses Integer Suffix, Example: T#PovNext1)

This returns the relative next period from the currently selected POV period. If the POV is 2024M8, T#PovNext5 returns 2025M1.

T#PovFirstInYear

This returns the first period in the year portion of the POV period. It is the same as T#POVYearM1, T#POVYearQ1, or, T#POVYearH1 based on the Input Frequency setting of the given scenario. T#M1, T#Q1, or, T#H1 work the same way as well.

- If the POV is 2024M2, T#PovFirstInYear returns 2024M1.
- If the POV is 2024Q2, T#PovFirstInYear returns 2024Q1.

T#PovLastInYear

This returns the last period in the year portion of the POV period. If the current period is 2024M1, T#PovLastInYear returns 2024M12.

T#POVFirstInMonthPrior1

POV Month Example

This refers to the POV month and returns N months prior. If the POV is 2024M10 and T#POVFirstInMonthPrior5 is requested, 2024M5 is returned.

POV Week Example

This refers to the POV week and goes back N weeks from the first week in that month. This varies by calendar (445, 454, 544).

If the POV is 2024W38 (M9) in a 52 week 445 calendar, T#PovFirstInMonthPrior5 returns 2024W30. The result in this example is driven by the first week in M9, which is 2024W35, and then returns five periods prior to 2024W30.

T#PovFirstInQuarterPrior1 (Uses Integer Suffix, Example: T#PovFirstInQuarterPrior1)

This refers to the POV sub-period's quarter and goes back N periods from the first period in that quarter. If the period is 2024M5, T#PovFirstInQuarterPrior2 returns 2024M2. This example's result is driven by the first period in the requested quarter, which is 2024M4, and then returns two periods prior to 2024M2.

T#PovFirstInHalfYearPrior1 (Uses Integer Suffix, Example: T#Pov FirstInHalfYearPrior2)
This refers to the POV subperiod half year and goes back N periods from the first period in that half year.

If the period is 2024M5, T#PovFirstInHalfYearPrior2 returns 2023M11. The result in this example is driven by the first period in the half year requested, which is 2024M1, and then returns two periods prior to 2023M11.

T#PovFirstInYearPrior1 (Uses Integer Suffix, Example: T#Pov FirstInYearPrior3)
This refers to the POV year and goes back N periods from the first period.

If the period is 2024M5, T#PovFirstInYearPrior1 returns 2023M12. The result in this example is driven by the first period in the year requested, which is 2024M1, and then returns one period prior to 2023M12.

T#PovLastInYearPrior1 (Uses Integer Suffix, Example: T#PovLastInYearPrior4)
This refers to the POV year and goes back N periods from the last period.

If the period is 2024M5, T#PovLastInYearPrior1 returns 2024M11. The result in this example is driven by the last period in the year requested, which is 2024M12, and then returns one period prior from this period to 2024M11.

Time Functions and Workflow References

T#WF (Example: T#WF or S#WF)

This refers to the time period or scenario currently selected in the workflow. Examples are T#WF and S#WF.

- If the Workflow POV is 2024M1, T#WF returns 2024M1.
- If the Workflow POV is Actual, S#WF returns Actual.

T#WFYear (Uses Time Member Suffix, Example: T#WFYearM1)

This returns a specific time period within the current Workflow year.

- If the Workflow period is 2024M1, T#WFYear returns 2024.
- If the Workflow period is 2024M1, T#WFYearM3 returns 2024M3.

T#WFPriorInYear1 (Uses Integer Suffix, Example: T#WFPriorInYear1)

This returns the relative prior period from the currently selected Workflow period, but within the same year.

- If the current Workflow period is 2024M8, T#WFPriorInYear4 returns 2024M4.
- If the current Workflow period is 2024M8, T#WFPriorInYear10 returns the first period in the year, 2024M1, because there are not 10 periods prior to 2024M8 within the same year.

T#WFPrior1 (Uses Integer Suffix, Example: T#WFPrior2)

This returns the relative prior period to the current Workflow period. If the current Workflow period is 2024M1, T#WFPrior3 returns 2023M10.

T#WFNextInYear1 (Uses Integer Suffix, Example: T#WFNextInYear1)

This returns the relative next period in the future from the currently selected Workflow period, but within the same year.

- If the current Workflow period is 2024M4, T#WFNextInYear5 returns 2024M9.
- If the current Workflow period is 2024M4, T#WFNextInYear10 returns the last period in the year, 2024M12, because there are not 10 periods after 2024M4 within the same year.

T#WFNext1 (Uses Integer Suffix, Example: T#WFNext2)
This returns the relative next period to the current Workflow period. If the current Workflow period is 2024M8, T#WFNext5 returns 2025M1.

T#WFFirstInYear
This returns the first period in the year portion of the Workflow period. This is the same as T#WFYearM1, T#WFYearQ1, or T#WFYearH1 based on the Input Frequency settings of the given scenario.

- If the current Workflow period is 2024M2, T#WFFirstInYear returns 2024M1.
- If the current Workflow period is 2024Q2, T#WFFirstInYear returns 2024Q1.

T#WFLastInYear
This returns the last period in the year portion of the Workflow period. If the current Workflow period is 2024M1, T#WFLastInYear returns 2024M12.

Time Functions and Global References

T#Global (Example: T#Global or S#Global)
The application has a Global Time and Global Scenario setting found in the Application tab > Tools > Application Properties. These settings can be referenced, for example, T#Global and S#Global. For example, T#Global can be referenced in Cube View columns where everything before T#Global appears as the Actual scenario, and everything after T#Global appears as the Forecast scenario.

- If the Global POV is 2024M1, T#Global returns 2024M1.
- If the Global POV is Actual, S#Global returns Actual.

T#GlobalYear (Uses Time Member Suffix, Example: T#GlobalYearM1)
This returns a specific time period within the current Global year.

- If the Global period is equal to 2024M1, T#GlobalYear returns 2024.
- If the Global period is equal to 2024M1, T#GlobalYearM3 returns 2024M3.

T#GlobalPriorInYear1 (Uses Integer Suffix, Example: T#GlobalPriorInYear1)

This returns the relative prior period from the currently selected Global period, but within the same year.

- If the current Global POV period is 2024M8, T#GlobalPriorInYear4 returns 2024M4.
- If the current Global POV period is 2024M8, T#GlobalPriorInYear10 returns the first period in the year, 2024M1, because there are not 10 periods prior to 2024M8 within the same year.

T#GlobalPrior1 (Uses Integer Suffix, Example: T#GlobalPrior3)

This returns the relative prior period to the current Global period. If the current Global period is 2024M1, T#GlobalPrior3 returns 2023M10.

T#GlobalNextInYear1 (Uses Integer Suffix, Example: T#GlobalNextInYear5)

This returns the relative next period in the future from the currently selected Global POV period but within the same year.

- If the current Global POV period is 2024M4, T#GlobalNextInYear5 returns 2024M9.
- If the current Global POV period is 2024M4, T#GlobalNextInYear10 returns the last period in the year, 2024M12, because there are not 10 periods after 2024M4 within the same year.

T#GlobalNext1 (Uses Integer Suffix, Example: T#GlobalNext5)

This returns the relative next period to the current Global period. If the current Global period is 2024M8, T#GlobalNext5 returns 2025M1.

T#GlobalFirstInYear

This returns the first period in the year portion of the Global period. This is the same as T#GlobalYearM1, T#GlobalYearQ1, or T#GlobalYearH1 based on the Input Frequency setting of the given scenario.

- If the current Global period is 2024M2, T#GlobalFirstInYear returns 2024M1.
- If the current Global period is 2024Q1, T#GlobalFirstInYear returns 2024Q1.

T#GlobalLastInYear

This returns the last period in the year portion of the Global period. If the current Global period is 2024M1, T#GlobalLastInYear returns 2024M12.

Time Functions and General References

Time functions, for example, T#Year(|!MyTimeParam!|), return values based on what is within the parentheses, such as a parameters, substitution variables, and time members. For example, a user can select a time member from a parameter prompt, such as 2024M6.

With these time functions, reference the year separate from the subperiod wherever T# is used. The following examples use parameters and substitution variables.

This example uses a parameter named MyTimeParam. The Cube View can use the selection: T#Year(|!MyTimeParam!|)Period(|!MyTimeParam!|)

The Year function returns the specified year, and the Period function returns the period without the year. The result is a valid time member because they are combined.

If the user selected 2024M6, after the parameter substitution, this is the format: T#Year (2024M6)Period(2024M6)

The result used in the Cube View is T#2024M6.

This example shows the values for the same month last year . If the user selects 2024M6, this is the format: T#YearPrior1(|!MyTimeParam!|)Period(|!MyTimeParam!|) returns T#2023M6.

This example shows the values for the last month in the same year. If the user selects 2024M6, this is the format: T#Year(|!MyTimeParam!|)PeriodPrior1(|!MyTimeParam!|) returns T#2024M5.

This example shows all base members for the year using a Member Expansion: T#Year (|!MyTimeParam!|).Base

These examples look at the user POV using a substitution variable. If the user POV is 2024M1, these are the results:

- T#Year(|POVTime|)PeriodNext1(|POVTime|) returns 2024M2.
- T#YearPrior2(|POVTime|)Period(|POVTime|) returns 2022M1.
- T#YearNext2(|POVTime|)Period(|POVTime|) returns 2026M1.
- T#Year(|POVTime|)Quarter(|POVTime|).MonthsInQuarter returns 2024M1, 2024M2, and 2024M3.

This example looks at the Workflow POV time member and then counts back two years prior, such as when viewing the YTD Net Income Values of a prior year. If the Workflow time is 2024M1, T#YearPrior2(|WFTime|) returns 2022.

This example looks at the Global POV time member and then counts forward 3 years. If the Global time is 2024M3, T#YearNext3(|GlobalTime|) returns 2027.

Here are additional examples for each General Time Function, which return values based on the parameters, substitution variables, and time members within the parentheses.

T#Year():

This returns the year. If a period is specified after, it returns the year and then the specified period. If the Year section is skipped, the year comes from the Period section. These are examples:
T#Year(|!TimeParam!|)Period(|!TimeParam!|):

- T#Year(2024M6) returns 2024.
- T#Year(2024M6)Period(2024M6) returns 2024M6.
- T#Year(2024M6)M2 returns 2024M2.
- T#Year(2024M6)PeriodPrior3(2024M6) returns 2024M3.
- T#Year(2024M6)PeriodPrior8(2024M6) returns 2023M10. PeriodPrior8 caused the year to change.

- T#Year(2024M6)PeriodPriorInYear8(2024M6) returns 2024M1. PeriodPriorInYear8 does not change the year and returns M1 because there are not 8 periods prior to M6 in the year.

T#YearPrior1():

This returns a prior year based on the number in the function. If a period is specified after, it returns the prior year and then the specified period.

- T#YearPrior2(2024M6) returns 2022.
- T#YearPrior1(2024M6)Period(2024M6) returns 2023M6.
- T#YearPrior2(2024M6)Period(2024M6) returns 2022M6.

T#YearNext1():

This returns a future year based on the number in the function. If a period is specified after, it returns the future year and then the specified period.

- T#YearNext2(2024M6) returns 2026.
- T#YearNext1(2024M6)Period(2024M6) returns 2025M6.

T#Period():

This returns the period and year. The year comes from the period section if the Year() section is skipped.

- T#Period(2024M6) returns 2024M6.

T#PeriodPrior1():

This returns the relative prior period based on the number in the function. The year comes from the relative prior period.
T#PeriodPrior3(|WFTime|)

- T#PeriodPrior3(2024M6) returns 2024M3.
- T#PeriodPrior8(2024M6) returns 2023M10.

T#PeriodNext1():

This returns the relative future period based on the number in the function. The year comes from the relative future period.

T#PeriodNext2(|GlobalTime|)

- T#PeriodNext2(2024M6) returns 2024M8.
- T#PeriodNext8(2024M6) returns 2025M2.

T#PeriodPriorInYear1():

This returns the relative prior period within the same year based on the number in the function. The year comes from the period section.

- T#PeriodPriorInYear2(2024M6) returns 2024M4.
- T#PeriodPriorInYear8(2024M6) returns 2024M1. This returns the first period in the year, M1, because there are not 8 periods prior to M6 in the same year.

T#PeriodNextInYear1():

This returns the relative future period within the same year based on the number in the function. The year comes from the period section.

T#PeriodNextInYear1(|GlobalTime|)

- T#PeriodNextInYear1(2024M6) returns 2024M7.
- T#PeriodNextInYear8(2024M6) returns 2024M12. This returns the last period in the year, M12, because there are not 8 periods after M6 in the same year.

T#FirstPeriodInQuarter():

This returns the first period in the quarter. The year comes from the period section.

T#FirstPeriodInQuarter(|WFTime|)

- T#FirstPeriodInQuarter(2024M6) returns 2024M4.
- T#FirstPeriodInQuarter(2024M2) returns 2024M1.

T#FirstPeriodInQuarterPrior1():

This looks at the first period in the quarter and then it returns the relative prior period based on the number in the function. The year comes from the relative prior period.

- T#FirstPeriodInQuarterPrior1(2024M6) returns 2024M3. The first period in the quarter is 2024M4, then 1 period prior is M3.
- T#FirstPeriodInQuarterPrior4(2024M6) returns 2023M12. The first period in the quarter is 2024M4, then 4 periods prior is 2023M12.

T#FirstPeriodInQuarterNext1():

This looks at the first period in the quarter, then it returns the relative future period based on the number in the function. The year comes from the relative future period.

T#FirstPeriodInQuarterNext2(|POVTime|)

- T#FirstPeriodInQuarterNext2(2024M6) returns 2024M6. The first period in the quarter is 2024M4, then 2 periods in the future is M6.
- T#FirstPeriodInQuarterNext9(2024M6) returns 2025M1. The first period in the quarter is 2024M4, then 9 periods in the future is 2025M1.

T#FirstPeriodInHalfYear():

This returns the first period in the half year. The year comes from the period section.

T#FirstPeriodInHalfYear(|WFTime|)

- T#FirstPeriodInHalfYear(2024M6) returns 2024M1.

T#FirstPeriodInHalfYearPrior1():

This looks at the first period in the half year. Then it returns the relative prior period based on the number in the function. The year comes from the relative prior period.

- T#FirstPeriodInHalfYearPrior1(2024M6) returns 2023M12. The first period in the half year is 2024M1, then 1 period prior is 2023M12.

T#FirstPeriodInHalfYearNext1():

This looks at the first period in the half year, then it returns the relative future period based on the number in the function. The year comes from the relative future period.

T#FirstPeriodInHalfYearNext2(|WFTime|)

- T#FirstPeriodInHalfYearNext2(2024M6) returns 2024M3. The first period in the half year is 2024M1, then 2 periods in the future is M3.

T#FirstPeriodInYear():

This returns the first period in the year. The year comes from the period section.

T#FirstPeriodInYear(|WFTime|)

- T#FirstPeriodInYear(2024M6) returns 2024M1.

T#FirstPeriodInYearPrior1():

This looks at the first period in the year, then it returns the relative prior period based on the number in the function. The year comes from the relative prior period.

T#FirstPeriodInYearPrior1(|!TimeParam!|)

- T#FirstPeriodInYearPrior1(2024M6) returns 2023M12. The first period in the year is 2024M1, then 1 period prior is 2023M12.

T#FirstPeriodInYearNext1():

This looks at the first period in the year, then it returns the relative future period based on the number in the function. The year comes from the relative future period.

T#FirstPeriodInYearNext2(|PovTime|)

- T#FirstPeriodInYearNext2(2024M6) returns 2024M3. The first period in the half year is 2024M1, then 2 periods in the future is M3.

T#Quarter():

This returns the quarter. The year comes from the period section.

T#Quarter(|WFTime|)

- T#Quarter(2024M6) returns 2024Q2.
- T#Quarter(2024M1) returns 2024Q1.

T#QuarterPrior1():

This returns the relative prior quarter based on the number in the function. The year comes from the relative prior quarter.

- T#QuarterPrior1(2024M6) returns 2024Q1.
- T#QuarterPrior3(2024M6) returns 2023Q3.

T#QuarterNext1():

This returns the relative future quarter based on the number in the function. The year comes from the relative future quarter.

T#QuarterNext2(|POVTime|)

- T#QuarterNext2(2024M6) returns 2024Q4.
- T#QuarterNext3(2024M6) returns 2025Q1.

Weekly Functions

The following Time Functions apply to weekly applications. The following examples are based on a 52 week 445 calendar.

T#FirstPeriodInMonth

If a weekly period is used in the function, this returns the first week in the month. The year comes from the period section.

- T#FirstPeriodInMonth(2024W43) returns 2024W40.

T#FirstPeriodInMonthPrior1()

If a weekly period is used in the function, this looks at the first weekly period in the month. It then returns the relative prior weekly period based on the number in the function. The year comes from the relative prior period.

T#FirstPeriodInMonthPrior3(|WFTime|)

- T#FirstPeriodInMonthPrior3(2024W5) returns 2024W2.
- T#FirstPeriodInMonthPrior3(2024W1) returns 2023W50.

T#FirstPeriodInMonthNext1()

If a weekly period is used in the function, this looks at the first weekly period in the month. It then returns the relative future weekly period based on the number in the function. The year comes from the relative future period.

- T#FirstPeriodInMonthNext2(2024W49) returns 2024W50. The first weekly period in the month is 2024W48, then 2 periods after is 2024W50.
- T#FirstPeriodInMonthNext2(2024W52) returns 2024W50. The first weekly period in the month is 2024W48, then 2 periods after is 2024W50.

T#Month()

This returns the month. The year comes from the period section.

- T#Month(2024W40) returns 2024M10

T#MonthPrior1()

This returns the relative prior month based on the number in the function. The year comes from the relative prior month.

- T#MonthPrior1(2024W5) returns 2024M1.
- T#MonthPrior2(2024W5) returns 2023M12.

T#MonthNext1()

This returns the relative future month based on the number in the function. The year comes from the relative future month.

T#MonthNext5(|POVTime|)

- T#MonthNext5(2024W9) returns 2024M8.
- T#MonthNext2(2024W52) returns 2025M2.

Changing the Time Label in Headers when Using These Time Functions
When using the Time function to change a header label in a Cube View, use the :Name("") function and substitute the label.

The following example shows what a Time function returns in the header of a Cube View without and with a :Name("") function. The Time function that follows returns the last period of the prior year using the Global Time period. For this example, the Global Time is 2024M6, and the |MFYear| substitution variable is the Member Filter year:

- T#YearPrior1(|GlobalTime|)M12 returns 2023M12 - Dec 2023 (dependent on the Cube View Header Text properties).
- T#YearPrior1(|GlobalTime|)M12:Name("Year End |MFYear|") returns Year End 2023.

The following examples show what similar substitution variables return if Global Time is 2024M6:

- T#YearPrior1(|GlobalTime|)M12:Name("Year End |MFTime|") returns Year End 2023M12.
- T#YearPrior1(|GlobalTime|)M12:Name("Year End |MFTimeDesc|") returns Year End Dec 2023.
- T#YearPrior1(|GlobalTime|)M12:Name("Year End |MFTimeShortDesc|") returns Year End Dec.
- T#YearPrior1(|GlobalTime|)M12:Name("Year End |MFSubPeriod|") returns Year End M12.

Substitution Variables

Substitution variables can be used to exchange values at runtime for parameters presented to the user before a dashboard runs, for members in the POV, for global variables for scenario or time, or for workflow variables for time, scenario, or other Workflow Profile attributes.

Substitution Variables from Current POV

These settings will return the values from the user's current POV for each dimension.

By adding Desc to the end of many of these substitution variables, the description for the member will be returned instead of the member name:

|PovEntityDesc| will return Akron instead of the entity name 123.

- POVCube and POVCubeDesc
- POVEntity and POVEntityDesc
- POVParent and POVParentDesc
- POVConsolidation and POVConsolidationDesc
- POVScenario and POVScenarioDesc
- POVTime, POVTimeDesc and POVTimeShortDesc
- POVView and POVViewDesc
- POVAccount and POVAccountDesc
- POVFlow and POVFlowDesc
- POVOrigin and POVOriginDesc
- POVIC and POVICDesc
- POVUD1-UD8 and POVUD1-8Desc

> **NOTE:** If there is an index out of range, the Cube View will display an error showing the data POV as not valid. When the Cube View is passed to the pivot grid control, it attempts to resolve the error column. It cannot, so for the Time dimension, it will display the default time member as "Name - Description". To resolve the issue in the pivot grid, you must first correct the issue in the source Cube View.

Substitution periods are defined within a given year. The number of subperiods are limited to the number of months or weeks within the given application year. For example, if you use prior or next within a given year, it will stop within the year. PovSubPeriodPrior14 in a monthly application for a year will not go past M1. PovSubPeriodNext will not go past M12. In a weekly application, PovSubPeriodPrior will not go past W1, and PovSubPeriodNext will not go past W53.

POVTimeDimProfile

This returns the time dimension profile name associated with the current cube POV.

POVCurrency

This returns the actual currency for an entity based on the Consolidation dimension setting in the POV. For example, if Consolidation is set to Local for the Manchester Entity, GBP will return, which is its currency. If Consolidation is set to Translated and the Parent is set to US, USD will return.

POVTimePriorInYearN

This returns the prior (or more) time period if it is still in the current year.

POVTimePriorN

This returns the prior (or more) time period.

POVTimeNextInYearN

This returns the next (or more) time period if it is still in the current year.

POVTimeNextN

This returns the next (or more) time period.

POVYear

This returns the year portion of a POV time period.

POVYearPriorN

This returns the year portion of a time period for the previous year where N is the number of years.

POVYearNextN

This returns the year portion of a time period for the upcoming year where N is the number of years.

POVSubPeriod

This returns the week, month, quarter, or half year portion of a time period. If the POV is 2012M5, M5 is the Subperiod.

POVSubPeriodPriorN

This returns the previous week, month, quarter, or half year portion of a time period.

POVSubPeriodNextN
This returns the next week, month, quarter, or half year portion of a time period.

POVSubPeriodNum
This returns the subperiod number for the current POV, so if the POV is set to M3, the number 3 is returned.

POVTimeFirstInYear
This is the same as T#POVFirstInYear, T#W1, T#M1, T#Q1, or T#H1 based on the input grequency of the scenario.

POVTimeLastInYear
This returns the last time period in the year based on the input frequency of the scenario.

Workflow Substitution Variables

WF
This is used within the Point of View area for scenario and time: T#WF.

WFProfile
This provides the current Workflow Profile name.

WFProfileIndex
This provides a numeric value of the Workflow Profile in the hierarchy.

WFProfileLastDescendantIndex
This provides an index of bottom descendants in the Workflow Profile tree.

Workflow Profile Keys
These return a numeric internal ID for Workflow Profiles.

- WFProfileKey
- WFReviewProfileKeys
- WFInputParentProfileKeys
- WFImportProfileKeys
- WFFormProfileKeys
- WFJournalProfileKeys

WFScenario
This returns the scenario in the workflow view.

WFScenarioDesc
This returns a description of the scenario in the workflow view.

WFScenarioID
This returns the numeric ID of the scenario in the workflow view.

WFTime
This returns the time period associated with current Workflow Unit.

WFTimeDesc and WFTimeShortDesc
This returns the description (for example, Feb 2011) or short description (for example, Feb) for workflow time as defined in time profile.

WFTimeID
This returns the numeric ID of the Workflow Unit time period.

WFTimePriorInYearN
This returns the prior (or more) workflow time period if it is still in the current year.

WFTimePriorN
This returns the prior (or more) time period.

WFTimeNextInYearN

This returns the next (or more) time period if it is still in the current year.

WFTimeNextN

This returns the next (or more) time period.

WFYear

This returns the year portion of the Workflow Unit.

WFYearPriorN

This returns the year portion of a time period for the previous year where N is the number of years.

WFYearNextN

This returns the year portion of a time period for the upcoming year where N is the number of years.

WFSubPeriod

This returns the week, month, quarter, or half year portion of a time period

WFSubPeriodPriorN

This returns the previous week, month, quarter, or half year portion of a time period.

WFSubPeriodNextN

This returns the next week, month, quarter, or half year portion of a time period.

WFSubPeriodNum

This returns the number for the current workflow period, so if the workflow is set to M3, the number 3 is returned.

WFTimeFirstInYear

This is the same as T#WFFirstInYear, T#W1, T#M1, T#Q1, or T#H1 based on the input frequency of the scenario.

WFTimeLastInYear

This returns the last time period in the year based on the input frequency of the scenario.

WFStartTime

This is the workflow start time entry for this scenario.

WFStartTimeDesc

This provides a description of the workflow start time entry for this scenario.

WFStartTimeShortDesc

This provides a short description of the workflow start time entry for this scenario.

WFEndTime

This is the workflow end time entry for this scenario.

WFEndTimeDesc

This provides the description of the workflow start time entry for this scenario.

WFEndTimeShortDesc

This provides the short description of the workflow start time entry for this scenario.

WFCubeRoot

This returns the very top level of the Workflow Profile hierarchy.

WFCube

This returns the cube related to this Workflow Profile.

WFTimeDimProfile

This returns the time dimension profile name associated with the current Workflow POV.

WFEntityDim

The returns the dimension for the workflow entities assigned to this Workflow Profile.

WFScenarioDim

This returns the Scenario dimension for the cube assigned to this Workflow Profile.

WFAccountDim

This returns the Account dimension associated with the cube for this Workflow Profile.

WFFlowDim
This returns the Flow dimension for the cube assigned to this Workflow Profile.

WFUD1Dim-WFUD8Dim
This returns the UD1 dimension-UD8 dimension for the cube assigned to this Workflow Profile.

WFText1, WFText2, WFText3, and WFText4
This provides optional text fields that can be populated and referenced from a Workflow Profile.

Global Substitution Variables

GlobalScenario
This is the Global Point of View Scenario from Application Properties.

GlobalTime
This is the Global Point of View Time period from Application Properties.

GlobalScenarioDesc
This provides the descriptions for the global scenario.

GlobalTimeDesc and GlobalTimeShortDesc
This provides the description (for example, Feb 2011) or short description (for example, Feb) for global time, as defined in the time profile.

Cube View Substitution Variables

These settings will return the values from the current Cube View in the Point of View slider for each dimension. For the dimensions not specified in the Cube View Point of View slide out panel, the member will be represented by the POV settings.

By adding Desc to the end of many of these substitution variables, the description for the member will be returned instead of the member name:

- |CVAccountDesc| will return Net Sales instead of 61000.
- |CVTimeDesc| will return Dec 2011 instead of 2011M12.
- |CVTimeShortDesc| will return Dec instead of 2011M12.

The following substitution variables will return the Cube View name, cube name, or dimension name associated with the specific Cube View as well as a description.

- CVName and CVDesc
- CVCube and CVCubeDesc
- CVTimeDimProfile
- CVEntity and CVEntityDesc
- CVParent and CVParentDesc
- CVConsolidation and CVConsolidationDesc
- CVScenario and CVScenarioDesc
- CVTime, CVTimeDesc, and CVTimeShortDesc
- CVView and CVViewDesc
- CVAccount and CVAccountDesc
- CVFlow and CVFlowDesc
- CVOrigin and CVOriginDesc
- CVIC and CVICDesc
- CVUD1-8 and CVUD1-8Desc

CVCurrency

This returns the actual currency for an entity based on setting in the specific Cube View.

Additional Cube View Time Functions

See Substitution Variables from Current POV to see examples of these Time Functions.

- CVTimePriorInYear
- CVTimePrior
- CVTimeNextInYear
- CVTimeNext
- CVYear
- CVYearPrior
- CVYearNext
- CVSubPeriod
- CVSubPeriodPrior
- CVSubPeriodNext
- CVSubPeriodNum
- CVTimeFirstInYear
- CVTimeLastInYear

General Substitution Variables

The script or variable will use pipe characters to include a predefined substitution variable, for example, |UserName|. These can be used in business rules, Cube Views, and dashboard headers.

Null

|Null| is empty text. |Null| is mostly used in cube-level security and the Stage parser. I t can also be used within a comma-separated list of parameter values to set a value to be an empty string. However, |Null| is not a commonly used substitution variable.

Space

This is used to replace a string with nothing or use a space along with the Name function in a Member Filter. For example, to make a column header or row header blank, use T#POV:Name(“ ”) or T#POV:Name(|space|). Both will produce the same result.

UserName

This provides the current username.

UserText1-4

This allows the reference of the Text1 through Text4 properties related to a user account: |UserText3|.

AppName

This provides the application name.

DateTimeForFileName

This returns the current date and time: 20131208_102540.

DateForFileName

This returns the current date: 20131208.

DateLong

This returns the current date: Sunday, December 08, 2013.

DateMMDDYYYY

This returns the current date as month/day/year: 12/08/2013.

DateDDMMYYYY

This returns the current date as day/month/year: 08/12/2013.

DateYYYYMMDD
This returns the current date as year/month/day: 2013/12/08.

DateTimeHHMMSS
This returns the current date and time as hour:minutes:seconds: 11:00:19.

- DateTimeForFileNameUTC
- DateForFileNameUTC
- DateLongUTC
- DateMMDDYYYYUTC
- DateDDMMYYYYUTC
- DateYYYYMMDDUTC
- DateTimeHHMMSSUTC

The version of these functions with the UTC suffix returns the same result but in Coordinated Universal Time.

Member Filter Substitution Variables

To change how a member displays in results, use the following MF functions.

See the section on Changing the Time Label in Headers When Using These Time Functions under Time Functions for several examples of how these are used.

These are used in Name and GetDataCell functions only.

- MFTime
- MFTimeDesc
- MFTimeShortDesc
- MFYear
- MFSubPeriod
- MFSubPeriodNum

The following substitution variables work with the XFMemberProperty function to retrieve any dimension member name being used within a Member Filter. Add Desc to any of these substitution variables to display the member description instead of the member name.

See XFMemberProperty under Commonly Used Member Expansion Functions for more details on using these Substitution Variables.

- |MFCube|
- |MFEntity|
- |MFParent|
- |MFConsolidation|
- |MFScenario|
- |MFTime|
- |MFView|
- |MFAccount|
- |MFView|
- |MFFlow|
- |MFOrigin|

- |MFIC|
- |MFUD1|-|MFUD8|
- Loop1-4Variable
- Loop1-4DisplayVariable
- Loop1-4Index
- Variable1-10

Custom Substitution Variables

For user-defined (that is, custom) substitution variables, the user will enclose the variable name using both pipes and exclamation points: |!myVariable!|

This excludes the Member Filters substitution variables.

Custom subsitution variables are ideal for parameters added to a dashboard. A parameter called SalesDashboardEntity might be created and prompt the user with a list of all entities to choose from before filtering a report. In the Cube View that drives this report, limit the Cube View POV to the selected parameter entity by entering |!SalesDashboardEntity!|. When designing a dashboard, |!SalesDashboardEntity!| can also be entered in the title of the Dashboard component to be displayed at runtime.

Samples of Member Filter Functions

The Samples tab in the Member Filter Builder contains examples of Member Filter functions. In the Member Filter Builder, hover your cursor over the samples to view a description and an example.

Samples of Member Expansions

See "Member Expansions" , "Reverse Order Member Expansions" and the "Samples" tab in the Member Filter Builder section for examples.

Samples of Where Clause Expressions

See Member Expansions, Member Expansion Where Clause and the Samples tab in the Member Filter Builder for examples.

Samples of Time Expressions

Samples of GetaDataCell Expressions

Use the GetDataCell function to retrieve specific cells or perform math or business rule operations. The results are returned in a Cube View column or row. The column or row header comes from what is defined within :Name(), which can be changed.

Variance Using GetDataCell

This calculates the difference as a ratio between two members using: `((A-B)/Abs(B))`

```
GetDataCell(Variance(S#Actual,S#BudgetV1)):Name(Variance)
```

This example returns the value of `((S#Actual - S#BudgetV1) / Abs(S#BudgetV1))` in a Cube View column or row named Variance.

Variance Percent Using GetDataCell

This calculates the difference as a percentage between two members using: `((A-B)/Abs(B)) * 100`

```
GetDataCell(VariancePercent(S#Actual,S#BudgetV1)):Name(Var %)
```

This example returns the value of `((S#Actual - S#BudgetV1) / Abs(S#BudgetV1)) * 100` in a Cube View column or row named Var %.

Better or Worse Using GetDataCell

This calculates the difference between two members while considering the Account Type:

- Revenue accounts are calculated using (A-B).
- Expense accounts are calculated using ((-1 * A) - (-1 * B)).

```
GetDataCell(BWDiff(S#Actual, S#BudgetV1)):Name(BetterWorse Difference)
```

This example returns the value of `(S#Actual - S#BudgetV1)` for Revenue accounts and `((-1 *S#Actual) - (-1 * S#BudgetV1))` for Expense accounts in a Cube View column or row named BetterWorse Difference.

Better or Worse Percent Using GetDataCell

This calculates a variance percent while considering the account type:

- Revenue Accounts are calculated using ((A-B) / Abs(B)) * 100.
- Expense Accounts are calculated using (((-1 * A) – (-1 * B))/Abs(B)) * 100.

```
GetDataCell(BWPercent(S#Actual, S#BudgetV1)):Name("BetterWorse %")
```

Ratio Using GetDataCell

This calculates the ratio between two members using the Divide function. Use the Divide function instead of the division operator (/) to avoid dividing by zero or NoData.

```
GetDataCell(Divide(A#Cash, A#AcctsRec)):Name(Ratio)
```

If there is data and the denominator is not zero, this example returns A#Cash divided by A#AcctsRec in a Cube View column or row named Ratio.

Difference Using GetDataCell

This calculates the difference of two members using the Subtraction operator.

```
GetDataCell(S#Scenario1-S#Scenario2):Name(Difference)
```

Sum Using GetDataCell

This calculates the sum of two members using the Addition operator.

```
GetDataCell(S#Scenario1+S#Scenario2):Name(Total)
```

Custom Functions to Call Business Rules Using GetDataCell

There are custom functions found by navigating to Samples > GetDataCell Expressions that call a Finance business rule using the BR# token within a GetDataCell.

Custom Function

```
GetDataCell(BR#[BRName = MyRuleName, FunctionName = MyFunctionName]):Name(Custom Function)
```

Custom Function (Condensed Syntax)

```
GetDataCell(BR#[MyBusinessRuleName, MyFunctionName]):Name(Custom Function)
```

Here is an example of a business rule that dynamically calculates Gross Profit percentage and a Cube View that calls it. See the following business rule and Cube View Member Filter syntax:

- **Business Rule Function**

  The function name in the example is GrossProfitPercentage, which needs to be referenced in the Member Filter.

```
Case Is = FinanceFunctionType.DataCell
    If args.DataCellArgs.FunctionName.XFEqualsIgnoreCase("GrossProfitPercentage")
        Return api.Data.GetDataCell("Divide(A#GrossProfit,A#Sales)")
    End If
```

- **Cube View Member Filter Using Custom Function(Condensed Syntax)**

  In the GetDataCell string, define the business rule name, function name, and Cube View column or row header name. This example returns the values from the GrossProfitPercentage function in the CorporateCalcs business rule in a Cube View column or row named Gross Profit %.

```
GetDataCell(BR#[CorporateCalcs, GrossProfitPercentage]):Name(Gross Profit %)
```

- **Custom Function With Parameters**

```
GetDataCell(BR#[BRName=MyBusinessRuleName, FunctionName=MyFunctionName, Name1=Value1,
AnotherName=[Another Value]]):Name(Custom Function)
```

passes a business rule in a Cube View Member Filter to perform calculations on the members referenced in the Cube View. Configure the Finance business rule to indicate the name of the function and any name-value pairs to reference in the Member Filter. A different action can be performed based on the definition of the name-value pair. The function can be reused with different values passed in through the name-value pairs, making it versatile.

In the following example, the business rule calculates an account as a percentage of sales using accounts in the Cube View member script. See the following business rule and Cube View Member Filter syntax.

- **Business Rule Function and Name-Value Pair**

  The function name in the following example is PctOfSales, which needs to be referenced in the Member Filter. The name-value pair is Account, which needs to be referenced and defined in the Member Filter.

```
        Case Is = FinanceFunctionType.DataCell
                If
args.DataCellArgs.FunctionName.XFEqualsIgnoreCase("PctOfSales") Then
                        Dim cvAccount As String_
                        =
args.DataCellArgs.NameValuePairs.XFGetValue("Account")
                        'Calculate account percentage of sales
                        Return api.Data.GetDataCell("Divide(A#" & cvAccount
& " ,A#60999)")
                End If
```

- **Cube View Member Filter Using Custom Function with Parameters**

  In the GetDataCell string, define the business rule name, the function name, the name-value pair from the business rule, and the Cube View row or column header name. This example returns A#GrossProfit divided by A#Sales, based on the business rule and name-value pair, in a row or column named Gross Profit %.

```
GetDataCell(BR#[BRName = CorporateCalcs, FunctionName = PctOfSales, Account =
61000]):Name(Gross Profit %)
```

In the following example, the business rule reads the current time period from the rule and gets the value for the prior year based on the member script. See the following business rule and Cube View Member Filter syntax.

- **Business Rule Function and Name-Value Pair**

  The function name in the following example is PYMonthForCol, which needs to be referenced in the Member Filter. The name-value pair is Field1, which needs to be referenced and defined in the Member Filter. Based on the name-value pair, the rule can run different actions.

```
Case Is = FinanceFunctionType.DataCell
If args.DataCellArgs.FunctionName.XFEqualsIgnoreCase("PYMonthForCol") Then
            Dim ms1 As String = args.DataCellArgs.NameValuePairs("Field1")
            Dim priorYearTimeName As String = _
api.Time.GetNameFromId(api.Time.GetPriorYearPeriodId())
            Dim memberScript As New System.Text.StringBuilder
            memberScript.Append(ms1)
            memberScript.Append(":T#")
            memberScript.Append(priorYearTimeName)
            'Get the Data Cell For the Prior Year
            Return api.Data.GetDataCell(memberScript.ToString)
End If
```

- **Cube View Member Filter Using Custom Function with Parameters**

  In the GetDataCell string, define the business rule name, the function name, and the name-value pair from the business rule. The Cube View returns the sales, A#60999, from the prior year.

```
GetDataCell("BR#[BRName = XFR_CVDataCellHelperNew, FunctionName = PYMonthForCol,
Field1 =A#60999]"):Name("Sales Last Year")
```

Samples of Column and Row Expressions

These samples contain GetDataCell calculations that reference the Cube View column and row names to perform column and row math.

See Column and Row Expressions and the Samples tab in the Member Filter Builder for examples.

Samples of Custom Member List Expressions

There are custom member list expressions that call a member list from a Finance business rule. You can build a custom list in a business rule and pass that business rule in a Cube View. This is used to return a list of members or member information when running the rule in a Cube View. MemberInfo allows users to specify additional information, for example, a parent entity ID and an indent level. An unlimited number of name-value pairs can be passed to the Member List business rule. Some use cases are ranking members by the top ten values in a Cube View or listing members alphabetically.

Custom Member List

```
E#Root.CustomMemberList(BRName=MyBusinessRuleName, MemberListName=MyMemberListName)
```

Custom Member List (Condensed Syntax)

```
E#Root.[MyBusinessRuleName, MyMemberListName]
```

Custom Member List with Parameters

```
E#Root.CustomMemberList(BRName=MyBusinessRuleName, MemberListName=MyMemberListName,
Name1=Value1, AnotherName=[Another Value])
```

This example returns a member list in a Cube View. See the following business rule and Cube View Member Filter syntax.

Business Rule with Name Value Pair

In the business rule, define the list name and the name-value pairs.

```
Select Case api.FunctionType
```

```
Case Is = FinanceFunctionType.MemberList
        If args.MemberListArgs.MemberListName.XFEqualsIgnoreCase("EntityParentList") Then
            Dim entityList As String = args.MemberListArgs.NameValuePairs("EntityList")
            Dim objMemberListHeader As New MemberListHeader
(args.MemberListArgs.MemberListName)
            Dim objMemberInfos As List(Of MemberInfo) = _
api.Members.GetMembersUsingFilter(args.MemberListArgs.DimPk, entitylist, Nothing)
            Dim objMemberList As New MemberList(objMemberListHeader, objMemberInfos)
            Return objMemberList
        End If
```

Cube View Member Filter Using Custom Member List

In the Cube View Member Filter, define the business rule name, member list name, the name-value pairs from the business rule, and the members to which it applies. This example returns the base members of E#Houston, which are South Houston and Houston Heights. The name-value pairs can be changed to create additional custom member lists.

```
E#Root.CustomMemberList(BRName = XFR_MemberListBasicNew, MemberListName = EntityParentList,
EntityList = E#Houston.Base)
```

This example returns a member list in alphabetical order in a Cube View. See the following business rule and Cube View Member Filter syntax.

Business Rule to Sort Member List in Alphabetical Order

This business rule sorts the balance sheet member list alphabetically when it is called from a Cube View. It returns A#BalanceSheet.base members for the workflow profile cube and Actual scenario type.

```
'This will put a member list of a dimension in Alphabetical order.
'Use the following on the cube view A#Member.[Name of Business Rule, Name of List in
Business Rule]
'A#Root.[XFR_MemberListAlphabetical, Acct_Sort]
Dim Memberlistname As String = "BSAcct_Sort"
Dim wfProfileName As String = api.Workflow.GetWorkflowUnitInfo.ProfileName
Dim wfProfileCubeName As String = BRApi.Workflow.Metadata.GetProfile(si, _
wfProfileName).CubeName
Dim CurCube As String = api.Pov.Cube.Name
```

```
Select Case api.FunctionType
        Case Is = FinanceFunctionType.MemberList
            If args.MemberListArgs.MemberListName = "BSAcct_Sort" Then
                    Dim objMemberListHeader = New
MemberListHeader(args.MemberListArgs.MemberListName)
                    Dim MemberListstart As String =
"A#BalanceSheet.base.Options(Cube = " & _
                    wfProfileCubeName & _
                    ",ScenarioType=
Actual,MergeMembersFromReferencedCubes=False)"
                    'brapi.ErrorLog.LogMessage (si,"Cube Name: " &
wfProfileCubeName)
                    'Read the members
                    Dim objMemberInfos As List(Of MemberInfo) = _
api.Members.GetMembersUsingFilter(args.MemberListArgs.DimPk, _
                    MemberListstart, Nothing)

                    'Sort the members
                    Dim objMembers As List(Of Member) = Nothing
                    If Not objMemberInfos Is Nothing Then
                        objMembers = (From memberInfo In objMemberInfos
Order By _
                        memberInfo.Member.Name Ascending Select
memberInfo.Member _
                        Distinct).ToList()
                    End If

                    'Return
                    Return New MemberList(objMemberListHeader, objMembers)
            End If
End Select
    Return Nothing
```

- **Cube View Member Filter Syntax**

  In the Cube View Member Filter, define the business rule name and the member list name. The A#BalanceSheet.base members for the workflow profile cube and Actual scenario type will show in the Cube View alphabetically by the member names.

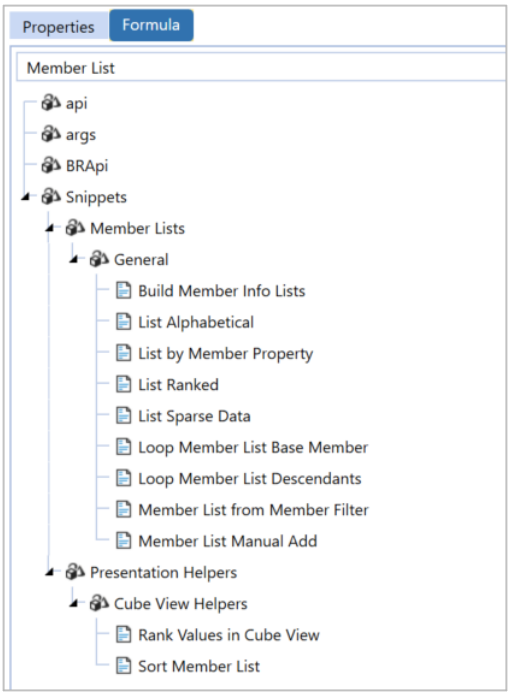
  ```
  A#Root.[XFR_MemberListAlphabetical, BSAcct_Sort]
  ```

This example shows how custom display names can be applied and displayed in a Cube View header when creating custom member lists.

Business Rule for Custom Display Names in Member Lists

```
Select Case api.FunctionType
Case Is = FinanceFunctionType.MemberList
Dim listName As String = args.MemberListArgs.MemberListName
        'Get the Passed in parameters
        Dim entityList As String = args.MemberListArgs.NameValuePairs("EntityList")
        If listName.Equals("AlphaSortList", _
StringComparison.InvariantCultureIgnoreCase) Then
Dim objMemberListHeader = New _
MemberListHeader(args.MemberListArgs.MemberListName)
            'Read the members
            Dim objMemberInfos As List(Of MemberInfo) = _
api.Members.GetMembersUsingFilter(args.MemberListArgs.DimPk, _
entityList, Nothing)
            'Sort the members
            Dim objMembers As List(Of MemberInfo) = Nothing
            If Not objMemberInfos Is Nothing Then
                objMembers = (From memberInfo In objMemberInfos Order By _
memberInfo.Member.Name Ascending Select memberInfo).ToList()
            End If
            'Apply a custom display name to the first item.
            If Not objMembers Is Nothing Then
                If objMembers.Count > 0 Then
                    objMembers(0).RowOrColNameForCalcScript = _
objMembers(0).NameAndDescription & "(Custom Name)"
                End If
            End If
            'Return list
            Return New MemberList(objMemberListHeader, objMembers)
        End If
End Select
```

Search snippets for member list in a Finance business rule for examples of business rules for custom member lists, for example, the List Alphabetical and List Ranked snippets. The Cube View Member Filter syntax displays within a comment in the business rule sample.



Samples of XFBR, XFCell, and XFMember Property

XFBR

You can use XFBR to call a business rule to create a custom string substitution.

```
XFBR(businessRuleName, brFunctionName, Name1=[Value 1], Name2=Value2)
```

This example uses XFBR to call a business rule to return an alternative view member based on the POV. See the following business rule and Cube View Member Filter syntax.

Business Rule for Custom Description Syntax

```
If (args.FunctionName.XFEqualsIgnoreCase("GetAltViewDescription")) Then
            Dim viewName As String = args.NameValuePairs.XFGetValue("ViewName")
            If viewName.XFEqualsIgnoreCase("Periodic") Then
```

```
                Return "MTD"
            Else
                Return viewName
            End If
        End If
```

- **Cube View Member Filter Syntax**

  The Cube View Member Filter contains V# and then defines the business rule name, the function name, and a name-value pair within XFBR(). If the POV contains V#Periodic, V#MTD is shown in the Cube View.

  ```
  V#XFBR(XFR_DescriptionHelper, GetAltViewDescription, ViewName=[|PovView|])
  ```

This example uses XFBR to call a business rule to return a certain time period based on the scenario selected when the Cube View is run. See the following business rule and Cube View Member Filter syntax.

Business Rule for Custom Description Syntax

If the scenario is Actual, then the twelve months prior is returned. Otherwise, the Cube View time is returned.

```
If (args.FunctionName.XFEqualsIgnoreCase("timeHelper1")) Then
            Dim currentScenario As String = args.NameValuePairs.XFGetValue
("currentScenario")
            If currentScenario.XFEqualsIgnoreCase("Actual") Then
                Return "MonthPrior12(|CVTime|)"
            Else
                Return "|CVTime|"
            End If
        End If
```

Cube View Member Filter Syntax

The Cube View Member Filter contains T# and then defines the business rule name, the function name, and a name-value pair within XFBR(). The scenario member in the Cube View POV is |!SelectScenario!|, which prompts the user to select a scenario member when the Cube View is run. If the user selects S#Actual, the time period 12 months prior is returned.

```
T#XFBR(XFR_CubeViewHelper, timeHelper1, currentScenario = |!SelectScenario!|)
```

XFCell

See Creating a Document in Microsoft Excel in the Presentation Guide and the Samples tab in the Member Filter Builder for examples.

XFMemberProperty

This function allows users to specify a dimension property and display the member property selection as a row or column header on a Cube View. This can be used if the header name for a row or column must be different than the member name or description. Use the XFMemberProperty within the Name() or GetDataCell() portion of a Member Filter.

This example returns the base members of A#60999 and shows the Account Type in the row or column instead of the member name or description.

```
A#60999.Base:Name(XFMemberProperty(DimType=Account,
Member=|MFAccount|,Property=AccountType))
```

The following name-value pair settings are needed in this function:

DimType: Define the dimension type, for example, Entity or Account.

Member: Define the dimension member name. In the previous example, the member name is defined with a Member Filter substitution variable.

> **NOTE:** To reference the dimension members specified in the Member Filter, use the dimension-specific Member Filter substitution variables. For example, use the Member Filter `A#NetSales.Children, use =|MFAccount|` to dynamically point to each child member of net sales while running this function. See Substitution Variables for more details.

Property: Define the exact dimension member property name, for example, Property=AccountType.

Add additional name-value pair settings in this function if the property varies:

- **VaryByCubeType**: Use this if the property varies by cube type.

```
A#NetSales.Children:Name(XFMemberProperty(DimType=Account, Member=|MFAccount|,
Property=Text1, VaryByCubeType=Standard)
```

- **VaryByScenario**: Use this if the property varies by scenario type, for example, Actual or Budget.

```
A#NetSales.Children:Name(XFMemberProperty(DimType=Account, Member=|MFAccount|,
Property=Text1, VaryByScenarioType=Actual))
```

- **VaryByTime**: Use this if the property varies by a specific time period, for example, 2024M5.

```
A#NetSales.Children:Name(XFMemberProperty(DimType=Account, Member=|MFAccount|,
Property=Text1, VaryByScenarioType=Actual, VaryByTime=2024M5))
```

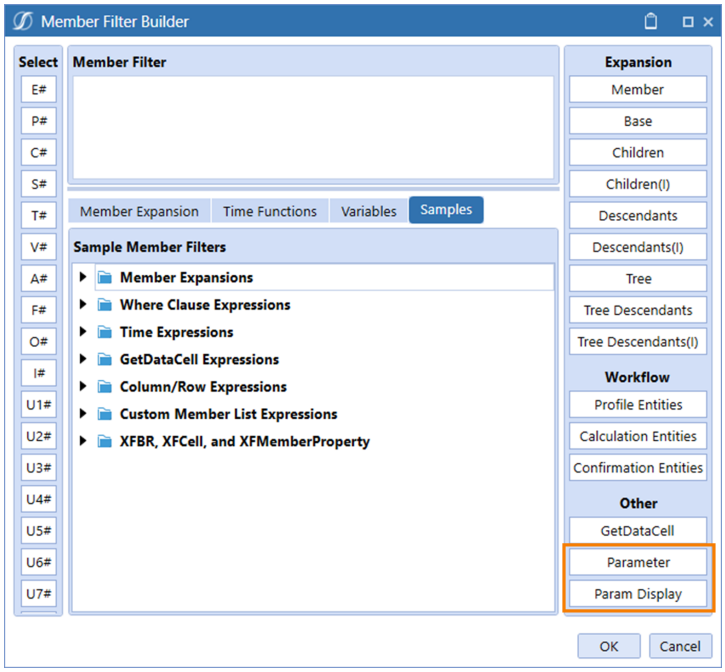
- **TimeDimProfile**: Use this to specify the time dimension profile. This can be set to CV, WF, any time dimension profile name, or a cube name using the CB# qualifier.

```
T#2024.Base:Name(XFMemberProperty(DimType=Time, Member=|MFTime|, Property=Description,
TimeDimProfile=|CVTimeDimProfile|))
```

The previous example uses the |MFTime| substitution variable to reference the members in the T#2024.Base Member Filter. It also uses the |CVTimDimProfile| substitution variable to point to the time dimension profile for the Cube View. See Substitution Variables more details.

Parameter and Param Display

In the Member Filter Builder, select the buttons for parameters to open the Object Lookup dialog box. Use these to enter a custom parameter reference that comes from either a form or dashboard in a Member Filter. Param Display is only used when working with a Delimited List parameter where the display item is displayed instead of the member name.

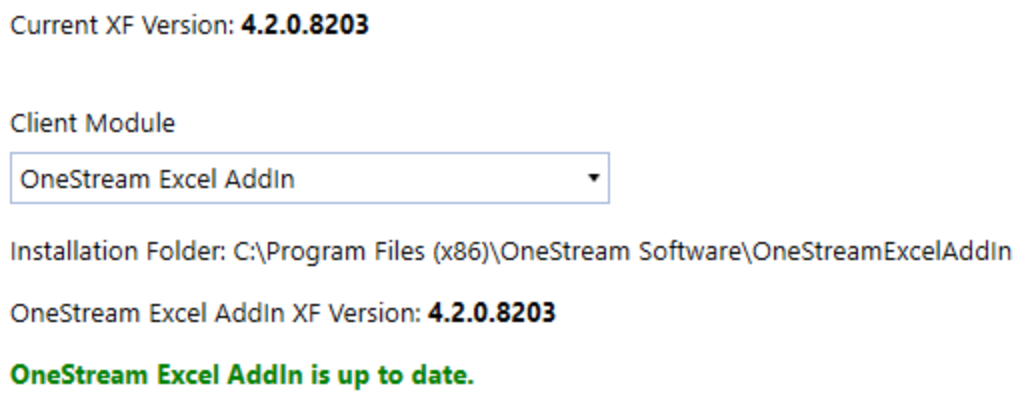


Client Updater

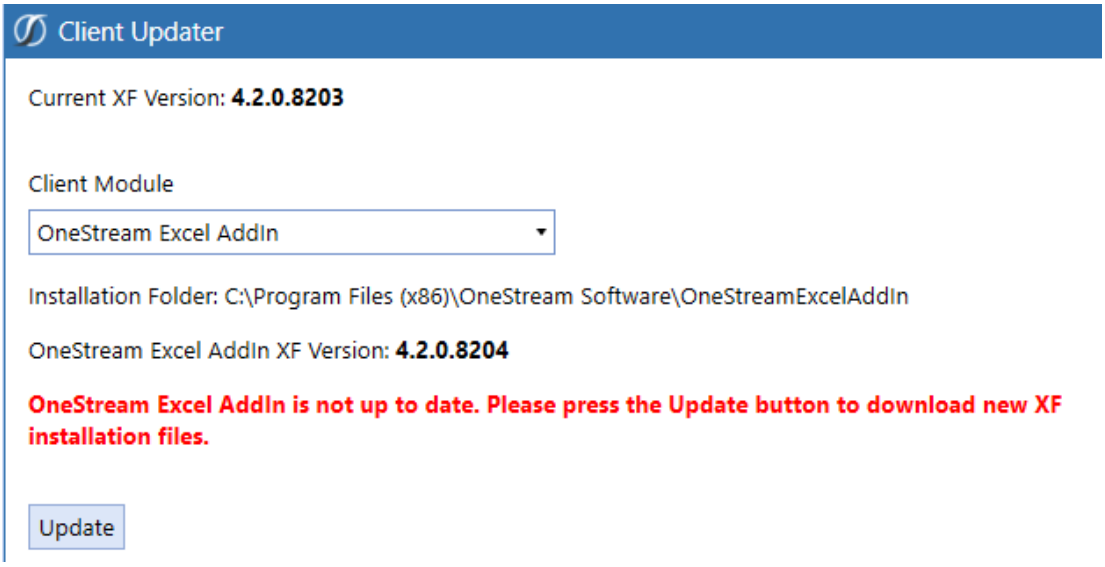
The client updater retrieves updated software from the OneStream server for the Excel Add-In client program when the version used does not match the version found on the server being connected. To update the client, the user must be able to write to the installation folder. From this page, the user can first save work and then select the Restart OneStream as Windows Administrator button using elevated Windows privileges.

Information about the current version displays at the top of the window as well as the module being reviewed, the location of the installation folder, and the version status.

The following example displays matching versions.



The following example displays versions that do not match.



Client Module

Use this field to select which application, for example, OneStream Excel Add-In, to review current and installed versions for updates.

1. From the **Client Module** drop-down menu, select the application.
2. If the versions do not match, select the **Update** button and then select the **OK** button.

> **NOTE:** Close any open versions of Excel before clicking the OK button to proceed.

> **NOTE:** A backup folder with files for outdated version is automatically created and saved as part of the update process. It can be found in the same location as the newly updated version folder.

Data Management

The Data Management page enables you to copy or clear data for a cube, scenario, entity, and time. To do this, you must use a data management group containing a step and a sequence. The steps are created and specifically defined by cube, scenario, entity, and time. Once the step is defined and saved, it can be assigned to a sequence. Once these groups are created, they are organized into data management profiles.

To create a data management group, select the **Create Group** icon. A data management group can be assigned to multiple data management profiles.

Property	Description
General	
Name	The name of the new data management group.
Description	A description of the new data management group.
Security	
Access Group	Manages the users that have access to the data management group.
Maintenance Group	Manages the users that have access to maintain and administer the data management group.

See Steps and Sequences.

Rename, Copy, and Paste

Existing sequences and steps can be renamed, copied, and pasted using the toolbar.

To rename a step or sequence, select it from the data management group and select the **Rename Selected Item**  icon. Rename the step or sequence and select the **OK** button.

To copy a sequence or step, right-click and select **Copy** or and select the **Copy**  icon. To paste a sequence or step, right-click the location in the Data Management Group pane and select **Paste** or select the **Paste**  icon.

> **NOTE:** Data management groups and profiles cannot be copy or pasted.

> **NOTE:** When you paste a sequence or step within the same hierarchy, _Copy is appended to the item's name.

Search

To search a data management object, complete the following steps:

1. Select the **Search**  icon to display the Select Data Management Object dialog box.
2. From the **Object Type** drop-down menu, select **Sequence**, **Step**, or **All Items**.
3. In the search bar, enter information on the object and select the **Search** button. To view the results by hierarchy, click the **View in Hierarchy** button.

Sequences

A sequence is an ordered series of one or more data management steps that run in the order defined in the Sequence Steps tab. To create a new data management sequence, select the

Create Sequence  icon.

General (Sequence)

Name
The name of the data management sequence.

Description (Optional)
A description of the data management sequence.

Data Management Group
The data management group the sequence belongs to.

Application Server (Optional)
This enables you to specify a particular application server in the event the sequence has an abnormally long run-time or one application server is preferred over another.

Notification

Use these properties to enable email notifications upon the completion, failure, or both, of a data management sequence.

Enable Email Notification
When set to False, the default value, email notifications are not sent. When set to True, notifications can be customized by user and group and are sent based on event type.

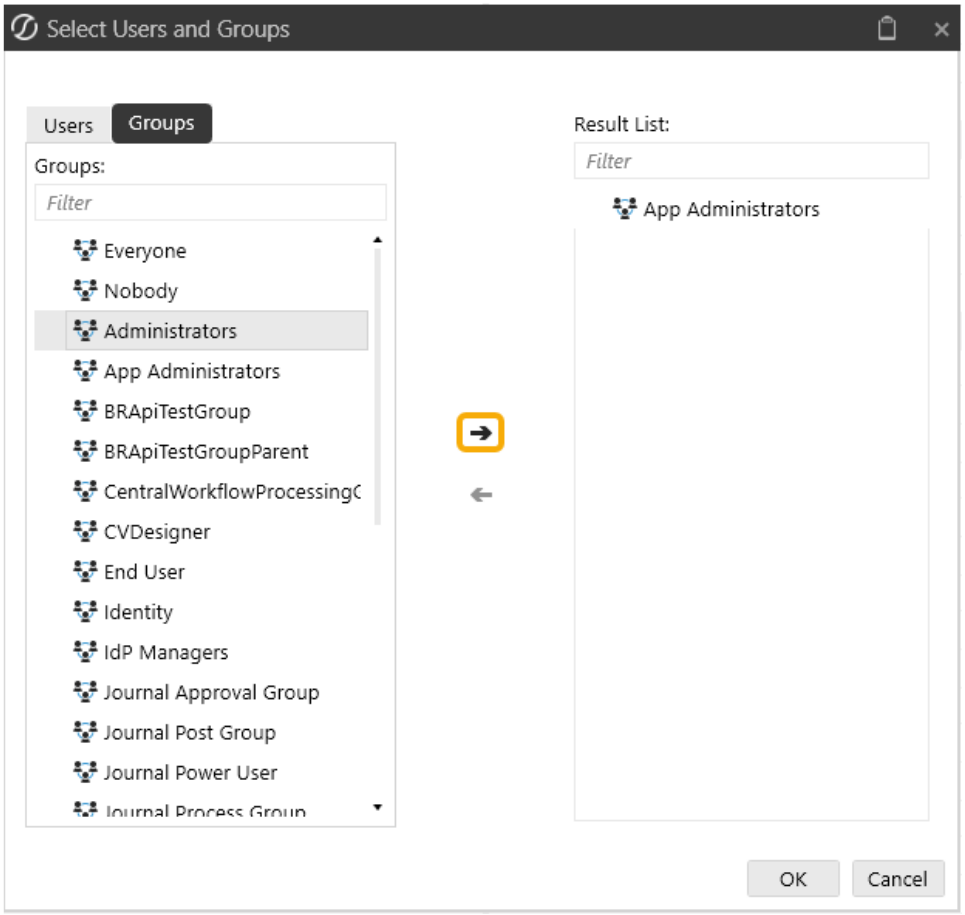
> **NOTE:** If there is no email server available, this feature is disabled.

Notification Connection

Email server connections are automatically identified. Select the appropriate server.

Notification User and Groups

Select users and groups to be notified when the data management sequence runs. Select the ellipses ... to display the Select Users and Groups dialog box. Select users and groups, and use the **Add Selected Item** icon to move them to the **Results List** pane. Select the **OK** button.



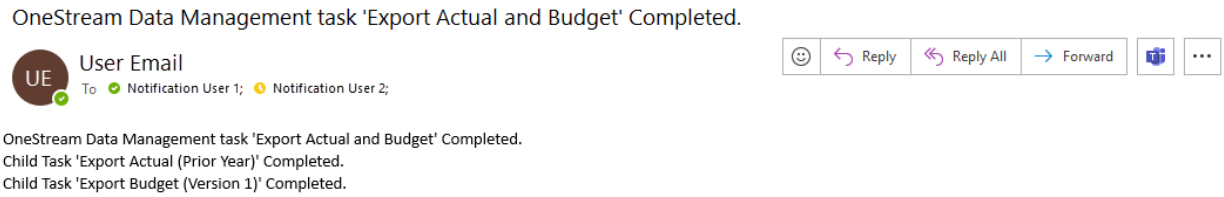
> **NOTE:** The error log captures information about users with no, invalid, or deactivated email accounts.

Notification Event Type

Select the notification event to trigger the email. You can choose from the following options:

- **Not Used**: This disables the feature when Enable Email Notification is set to True.
- **On Success**: This sends the email when the data management sequence runs successfully.
- **On Failure**: This sends the email when the data management sequence fails.
- **On Success or Failure**: this sends the email when the data management sequence finishes regardless of status.

The following example is a sample notification email.



Task Activity

The data management sequence monitors the server's CPU and evaluates other queued tasks to ensure they are processed in order as resources become available. If the server resource utilization is greater than the limit, the job status is set to Queued and the task progress bar stays on the queued task step until enough CPU resources are available to start the task. This can be monitored in Task Activity.

Use Queueing

When set to True, the default value, queueing is used for this task to have better control of the application server's CPU utilization. This task will not start running until CPU utilization is below the specified value or until all previously queued tasks have been completed.

Maximum % CPU Utilization To Run

Enter the maximum CPU utilization allowed on the application server before this task can transition from queued to running. Enter a number between 1 and 100 or leave this setting blank to use the default value of 70%. This value can be modified in the Application Server Configuration.

> **NOTE:** Do not set a value less than 10 or the task may never start.

Maximum Queued Time Before Canceling (minutes)

Enter the number of minutes that this task is allowed to be queued before it cancels automatically. Enter a number or leave this setting blank to use the default value of 20 minutes. This value can be modified in the Application Server Configuration.

When a task is queued, the Task Progress dialog box stays on Task Queued. Open Task Activity to monitor the queued task and the current server CPU utilization.

> **NOTE:** Batch processing queue overrides other workflow batch queue settings. The batch processing queue does not apply to batch script, only the batch screen in the workflow.

Parameter Substitutions 1-8

Parameter Name (Optional)

The name of a parameter being used as a variable to be passed into the sequence.

Value

The value passed into the related parameter variable.

Sequence Steps Tab

Use the Sequence Steps tab to add steps to a sequence. Select the **Add Step** icon to display the **Add Step** dialog box. Choose a step and select the **OK** button.

Exposing Data Management Automation through OneStream Web API

OneStream Web API is a RESTful web service designed to expose OneStream data automation functions when interacting with third-party API client applications.

OneStream Web API must be installed on a web server. It also must be configured for external authentication providers supporting OAuth2.0/OpenID connect authorization protocol. Identity providers currently supported are Okta, Azure AD (Microsoft Entra ID), and PingFederate.

OneStream Web API is API client agnostic. It accepts and outputs data in JSON format making it possible for every API client application that supports this format to also interact with this service. One of the most widely used API clients is Postman, a Windows application. For more information about how to configure OneStream Web API to interact with Postman, see the autogenerated documentation at http(s)://[servername]:[port]/onestreamapi.

OneStream Web API endpoints

Authentication endpoint. This represents a RESTful service for authentication.

- **POST api/Authentication/LogonAndReturnCookie**: This is used primarily by the enablement team to verify that Web API installations are completed successfully. This returns a one-time cookie value that holds authentication state or a message indicating failure along with a proper HTTP code.

DataManagement endpoint. This represents a RESTful service of data management.

- **POST api/DataManagement/ExecuteSequence**: This runs a data management sequence and returns a success/failure message along with a proper HTTP code.
- **POST api/DataManagement/ExecuteStep**: This runs a data management step and returns a success/failure message along with a proper HTTP code.

DataProvider endpoint. This represents a RESTful service of data provider.

- **POST api/DataProvider/GetAdoDataSetForAdapter**: This returns a JSON representation of a DataSet,a given dashboard adapter, or a failure message along with a proper HTTP code.
- **POST api/DataProvider/GetAdoDataSetForCubeViewCommand**: This returns a JSON representation of a DataSet for a given Cube View or a failure message along with a proper HTTP code.
- **POST api/DataProvider/GetAdoDataSetForSqlCommand**: This returns a JSON representation of a DataSet for a given SQL query or a failure message along with a proper HTTP code. The administrator role is required for this functionality.
- **POST api/DataProvider/GetAdoDataSetForMethodCommand**: This returns a JSON representation of a DataSet for a given pre-defined list of method commands used by XFDataProvider to fill a DataSet or a failure message along with a proper HTTP code. The administrator role is required for this functionality.

Steps

To create a new data management step, select the **Create Step**  icon.

Name
The name of the data management step.

Description (Optional)
The description of the data management step.

Data Management Group
The data management group the step belongs to.

Step Type
The type of data management step chosen. There are various data management step types.

The following properties display based on the step type selected.

Calculate

A step can be created to use one of the consolidation/calculation options.

Calculation Type

Select a calculation type. You can choose from the following options:

- **Calculate**: This runs a calculation when dashboard components are changed. It runs calculations at the entity level within the local member of the Consolidation dimension without translating or consolidating.

- **Translate**: This runs a translate when dashboard components are changed. It runs the Calculate step above at the entity level and then translates data within the translated member of the Consolidation dimension for each applicable relationship.
- **Consolidate**: This runs a consolidate when dashboard components are changed. This runs the Calculate and Translate steps and then completes the calculations required all the way up the Consolidation dimension.
- **Force Calculate, Translate, or Consolidate**: The Force menu, for example, Force Consolidation, items run as if every cell included is marked as needing to be calculated, translated, or consolidated.
- **Calculate, Translate, or Consolidate with Logging**: The Logging items, for example, Force Translation with Logging, trigger additional detailed logging which can be viewed in the Task Activity area. Drill into a log to see the length of time and details for every calculation.
- **Force Calculate, Translate, or Consolidate with Logging**: This runs a Force Calculation, Translation, or Consolidation with Logging.

Execute Cube or Scenario Dynamic Source Data Copy
Execute the cube's or scenario's Dynamic Source Data Copy as part of the calculation if applicable. When Use Default is selected, the cube's or scenario's ExecuteCopyDuringCalc option is deferred to. When set to True, the copy is run. When set to False, the copy is skipped regardless of the cube's or scenario’s options.

Data Units

Cube
Specify the cube where the consolidation/calculation will run.

Entity Filter
Specify the entity, list of entities, or any combination of entity hierarchies to be included in the consolidation/calculation.

Parent Filter
If alternate hierarchies are used, a parent may be specified and included in the consolidation/calculation.

Consolidated Filter
Specify the Consolidation member or members to be included in the consolidation/calculation.

Scenario Filter
Specify the Scenario member or members to be included in the consolidation/calculation.

Time Filter
Specify the Time member or members to be included in the consolidation/calculation.

Execute Scenario Dynamic Source Data Copy

This copies base-level cube data from a specified source scenario, using the Dynamic Source Data settings, to a specified target scenario. Data copy occurs when a standard calculation is run on the target scenario and, by default, follows the standard OneStream calculation sequence.

Clear Data

General (Step)

Use Detailed Logging
When set to True, a detailed log found in the Task Activity Log provides additional detail about the process.

Data Units

Cube
Select the cube where the consolidation/calculation will run.

Entity Filter
Select the entity, list of entities, or any combination of entity hierarchies to be included in the consolidation/calculation.

Scenario Filter
Select the Scenario member or members to be included in the consolidation/calculation.

Time Filter
Select the Time member or members to be included in the consolidation/calculation.

Origin

Clear Imported Data
This indicates whether the Import member of the Origin dimension should be included in the step.

Clear Forms Data
This indicates whether the Forms member of the Origin dimension should be included in the step.

Clear Adjustment Data and Delete Journals
This indicates whether the Adjustment members of the Origin dimension should be included in the step.

Clear Data Cell Attachments
This indicates whether the data cell attachments of the Origin dimension should be included in the step.

Copy Data

General

Use Detailed Logging
When set to True, a detailed log found in the Task Activity log provides additional detail about the process.

Source Data Units

Source Cube
Select the source cube for the data copy.

Source Entity Filter
Select the source entity, list of entities, or any combination of entity hierarchies to be included in the data copy.

Source Scenario
Select the source Scenario member or members to be included in the data copy.

Source Time Filter
Select the source Time member or members to be included in the data copy.

Source View
Select the View member to be included in the data copy. This selection enables users to copy YTD data into a Periodic member and the other way around.

Source Data Units

Destination Cube
Select the destination cube for the data copy.

Destination Entity Filter
Select the destination entity, list of entities, or any combination of entity hierarchies to be included in the data copy.

Destination Scenario
Select the destination Scenario member or members to be included in the data copy.

Destination Time Filter
Select the destination Time member or members to be included in the data copy.

Destination View
Select the destination View member to be included in the data copy.

Origin

Copy Imported Data
Choose True or False to copy O#Import data

Copy Forms Data
Choose True or False to copy O#Forms data.

Copy Adjustment Data
Choose True or False to copy O#Adjustments data.

> **NOTE:** If the Source View and Destination View properties are left blank, the data copies to the same View member. For example, YTD will copy to YTD or Periodic will copy to Periodic.

Copy Data Cell Attachments
Select the data cell attachments to be included in the data copy.

Custom Calculate

The typical use of the custom calculate step is flexibility and speed of calculations during data entry. For instance, a user could make on-the-spot changes in a form, run a custom calculate and quickly experience what-if analysis based on the limited amount of data on the form. Instead of running a full calculate or consolidation on a Data Unit, this step can be used to run a calculation on a slice of data within one or many Data Units. For example, the calculation could be executed from within data management by selecting Save on a form in workflow through a Forms Event Handler or related a dashboard button could be used to enter budget data.

This type of calculation does not create audit information for each data cell affected and will therefore run faster than using the Copy Data step type.

The user running this must be a member of the scenario's manage data group or the step will fail. This helps prevent unauthorized users from launching steps like this which could alter or clear data unintentionally.

General

Use Detailed Logging
When set to True, a detailed log found in the Task Activity Log provides additional detail about the process.

Data Units

These are the Data Unit settings for what are affected by the applied calculation. A single cube and scenario can be affected by this step. Single or multiple entities, entity parents, consolidation members, and time periods can be affected through the Filter properties for entity, parent, consolidation, and time in this section using the Member Filter Builder dialog box. The related business rule entered in the business rule property runs once for each of the number of Data Units specified here. These Data Unit members, for example, entity, scenario, and time, can be referenced in the related business rule with functions such as API.Pov.Entity.Name. Otherwise, they will not need to be mentioned within the context of the business rule. Upon running, this resets the calculation status of the affected Data Units needing calculation.

Point of View

These are the single member entries of dimensions not in the Data Unit to be affected by the calculation. The properties are for view, account, flow, origin, IC and UD1-UD8. These values are provided in the step as a convenience and to be referenced from within the business rule. For instance, the UD1 member listed here in the POV setting could be referenced from within the business rule as api.Pov.UD1.Name. This can make the business rule flexible since the same rule could be used against multiple data management custom calculate steps but run differently based on the POV setting.

Business Rule

Business Rule and Function Name
Enter the name of the finance-type business rule and contained function to run when the step runs. This enables you to specify settings such as Durable Storage within the rule. A simple example of a business rule which calculates data with Durable Storage is displayed below:

The following example shows a business rule which calculates data with Durable Storage:

VB.Net

```
Select Case api.FunctionType
    Case FinanceFunctionType.CustomCalculate
        If args.CustomCalculateArgs.FunctionName.XFEqualsIgnoreCase("myFunctionName") Then
            api.Data.Calculate("A#Profit = A#Sales - A#Costs", True)
        End If
End Select
```

> **NOTE:** When a calculation or consolidation runs on this same Data Unit after this data management step is run, the data saved as calculated by this step is cleared unless it is saved with a Storage Type of Durable. ClearCalculatedData is the first step in the standard calculation sequence that runs during a calculation or consolidation on cube data. In this case, durable data is ignored during a calculate or even a force calculate or consolidate unless a ClearCalculated function is used within the business rule or Member Formula to purposely clear the durable data. However, if a calculation recalculates even data marked as Durable, it is replaced by the newly calculated data.

It is suggested that if this calculation within a custom calculate data management step is to be replicated within a separate Member Formula or business rule, both can refer to the same saved finance business rule function by name.

See Examples of Key Functions in Use for examples of using API.Data.Calculate and API.Data.ClearCalculated to store and clear Durable data as desired.

Parameters

Refer to any parameters inside the business rule by listing them here in name-value pairs.

> **Example:** Name1=Frankfurt, Name2=[Houston Heights]

Custom parameters can be used by using the correct syntax. This displays a prompt to the user at run-time.

> **Example:** Name3=|!myParam!|

Execute Business Rule

Use Detailed Logging
When set to True, a detailed log found in the Task Activity log provides additional detail about the process.

Business Rule

Business Rule
Select from the available application business rules to run custom scripts or procedures.

Parameters (Optional)
This field is provided to pass parameters or variables into the selected business rule.

Export Data

Use Detailed Logging
When set to True, a detailed log found in the Task Activity log provides additional detail about the process.

Output Files

File Share File Name
The name given to the file being exported to the application file share.

Include Cube in File Name
When set to True, the cube name is included in the file name when exporting.

Include Entity in File Name
When set to True, the entity name is included in the file name when exporting.

Include Parent in File Name
When set to True, the parent name is included in the file name when exporting.

Include Cons in File Name

When set to True, the consolidation member name is included in the file name when exporting.

Include Scenario in File Name

When set to True, the scenario name is included in the file name when exporting.

Include Time in File Name

When set to True, the time name is included in the file name when exporting.

Overwrite Existing Files

Set this to True to overwrite a previously exported file when exporting.

Options

Include Zeroes

When set to True, zero amount records are included when exporting.

Include Member Descriptions

When set to True, member descriptions are included when exporting.

Include Cell Annotations

When set to True, cell annotations are included when exporting.

Include Input Data

When set to True, input type data is included when exporting.

Include Calculated Data

When set to True, calculated data is included when exporting.

Data Units

Cube

Specify the cube where consolidation/calculation will run.

Entity Filter
Specify the entity, list of entities, or any combination of entity hierarchies to be included in the consolidation/calculation.

Parent Filter
If alternate hierarchies are used, a parent may be specified in order to be included in the consolidation/calculation.

Consolidated Filter
Specify the consolidation member or members to be included in the consolidation/calculation.

Scenario Filter
Specify the scenario member or members to be included in the consolidation/calculation.

Time Filter
Specify the time member or members to be included in the consolidation/calculation.

Combinations of Data Filters (use #All for all stored base-level data)

Use standard Member Filter functionality to select specific data required for all dimensions listed below:

- Account Filter (e.g., “A#IS.Descendants, A#All”)
- Flow Filter
- Origin Filter
- IC Filter
- UD1-8 Filter

Specific Data Filters

Use standard Member Filter functionality to have fine grain control over the data exported. Use this filter to choose more specific intersections of data.

Export File

Use the export file data management step to export an Extensible Document or Report Book to OneStream's File Share.

General

Use Detailed Logging
If set to True, a detailed log found in the Task Activity log will provide additional detail about the process.

Output File

File Share Folder
Enter the name of the File Share folder where the desired file is saved.

File Name Suffix
Enter a suffix to include in the name of the file.

Overwrite Existing Files
Set to True to overwrite a previously exported file when exporting.

Source File

File Source Type
Select a file source type. You can choose from the following options:

- **URL**: Display a file from an internal or external web page.
- **Application/System Dashboard File**: Display a file stored in the Workspace maintenance unit File section.
- **Application/System Database File**: Display a file stored in the application or system Database Share.
- **File Share File**: Display a file from the File Share.

URL or Full File Name

The URL or name of the file being used. Enter the full URL name or select the ellipsis  and browse to the desired file.

Process Extensible Document

When set to True, the data management step runs and processes the attached Extensible Document file. When set to False, the unprocessed file displays. This is mainly used for testing purposes.

> **NOTE:** An Extensible Document is a Text, Word, PowerPoint, or Excel file that uses parameters in its content. The file name must contain .xfDoc before the extension. For example, StatusReport.xfDoc.docx. See "Presenting Data With Extensible Documents" in the *Presentation Guide*.

Read Parameter Names from Extensible Doc

When set to True, the parameters used in the Extensible Document prompt the user when the data management sequence is run. When this is set to False, the parameters can be specified ahead of time in the parameters property below.

Parameters

Enter a comma separated list of name value pairs.

> **Example:** ParameterName1=ValueName1, ParameterName2=[Value Name2]

Export Report

Use Detailed Logging
When set to True, a detailed log found in the Task Activity log provides additional detail about the process.

Output File

File Share File Name
The name given to file being exported to the application file share.

Include Object Name in File
When set to True, the dashboard object is included in the file name when exporting.

Overwrite Existing Files
Set to True to overwrite a previously exported file when exporting.

Report File Type
Select a report file type. You can choose from the following options:

- **Combined PDF File**: This combines and exports all report PDF's into one PDF file.
- **PDFs in Zip File**: This exports all contained PDF's individually in a ZIP file.

Object Type
Indicate whether a dashboard or dashboard profile object is to be exported.

Object Name
The name of the dashboard or dashboard profile being used.

Object Parameters (Optional)
Enter parameters to pass or variables to use when running the report normally.

Reset Scenario

This step is similar to the clear data step, except it clears additional related application data. It will run faster because it does not create audit information for each data cell affected. It clears data within a range (including parent entity data), audit information, workflow status, and calculation status. This is intended for administrator use.

The user running it must be a member of the scenario's manage data group or the step will fail. This helps prevent unauthorized users from launching this step, which could alter or clear data unintentionally. It is recommended to change the manage data group from Nobody to a select exclusive user group before running and then change back to Nobody afterward.

> **IMPORTANT:** Ensure that your application database is backed up before performing a Reset Scenario. Reset data will clear, including data marked as Durable.

General

Use Detailed Logging
If set to True, a detailed log found in the Task Activity log will provide additional detail about the process.

Data Units

Scenario
Choose one scenario to be reset.

Reset All Time Periods
This defaults to True, which removes data and resets the workflow status for all time periods. When set to False, the next three properties are enabled.

Start Year / End Year
The first and last years' worth of time to clear from the application and workflow status.

Start Time Period in First Year (Optional)
The time period referenced in First Year to begin clearing data.

Run a Sequence or Step

To run a sequence or step, select the **Run**  icon.

When running a sequence in data management, go to **System** > **Tools** > **File Explorer** to find any file export. Then, go to **File Share** > **Applications** , choose an application, and select the **Data Management** folder. Go to **Export** > **Username** > **Choose the latest folder**.

Load/Extract Application Artifacts

You can import and export sections of the application using an XML format.

> **NOTE:** Only administrators have access to this portion of the tool.

Extract

Once a file type is selected, you can use the Extract icon to start the extraction process and name the output file.

Load

Once a file is selected, you can use the Load icon to initiate the process.

Extract and Edit

This option is available for all extracts and enables you to edit the XML file as needed.

File Types

The following file types can be extracted.

Application Zip File (All except for Data and FX)
This option exports all application structures except data and FX rates. After selecting the Extract icon, a dialog box displays to save the file in a zipped format. After selecting yes, the file can be named. This is the most complete option in the list of file types and can be used to create a copy of the application.

Application Security Roles

This covers all security roles found in Application > Tools > Security Roles. All items available to extract are selected by default. The defined roles are exported accordingly.

Application Properties

This covers all application properties found in Application > Tools > Application Properties. All items available to extract are selected by default.

Workflow Channels

This covers all workflow channels found in Application > Workflow > Workflow Channels. All items available to extract are selected by default.

Metadata

This has multiple sections such as business rules, time dimension profiles, dimensions, and cubes. The metadata can be found in Application > Cube > Dimension Library. OneStream can search for dimension member and relationship changes by username and time stamp. For more details on the Find Modified Items feature, see Extracting and Loading Dimensions

Cube Views

This has multiple sections, such as groups and profiles, which go together. Cube Views can be found in Application > Presentation > Cube Views. All items available to extract are selected by default. Deselect sections as needed.

Data Management

This has multiple sections, such as groups and profile, that go together. The Data Management page can be found under Application > Tools > Data Management. All items available to extract are selected by default.

Application Workspaces

This has multiple sections, such as maintenance units and profiles. Under Workspaces, components, data adapters, and parameters go together. The Workspaces page can be found in Application > Presentations > Workspaces. All items available to extract are selected by default.

Confirmation Rules

This has multiple sections, such as groups and profiles, that go together. Confirmation rules can be found in Application > Workflow > Confirmation Rules. All items available to extract are selected by default.

Certification Questions

This has multiple sections, such as groups and profiles, that go together. Certification questions can be found under Application > Workflow > Certification Questions. All items available to extract are selected by default.

Transformation Rules

This has multiple sections, such as business rules, groups, and profiles, that go together. Transformation rules can be found in Application > Data Collection >Transformation Rules. All items available to extract are selected by default.

Data Sources

This has multiple sections, such as business rules and data sources, that go together. Data sources can be found in Application > Data Collection > Data Sources. All items available to extract are selected by default.

Form Templates

This has multiple sections, such as groups and profiles, that go together. Form templates can be found under Application > Data Collection > Form Templates. All items available to extract are selected by default.

Journal Templates

This has multiple sections, such as groups and profiles, that go together. Journal templates can be found in Application > Data Collection > Journal Templates. All items available to extract are selected by default.

Workflow Profiles

This has multiple sections such as Workflow Profiles and Workflow Profile templates. Workflow Profiles can be found in Application > Workflow > Workflow Profiles. All items available to extract are selected by default. When loading Workflow Profiles through an XML file extracted previously or from a different application, the load process clears old property settings if they are not specified in the loaded XML file. This approach ensures that any property edits made to the Workflow Profile upon extracting it are honored when the same XML file is re-loaded.

Extensibility Rules

This only exports the extensibility rules in the business rules section. All others are exported with the object to which they are tied. For example, if there is an account description rule in the Parser section, it will be exported with the data source. Extensibility rules can be found in Application > Tools > Business Rules > Extensibility Rules. All items available to extract are selected by default.

FX Rates

Rates have their own cube. This has the sections All FX Rate Types and All Time Periods. FX rates can be found under Application > Cube > FX Rates. All items available to extract are selected by default.

FX Rate CSV

This is a CSV extract of the application FX rates which contain the FxRateType, Time, SourceCurrency, DestCurrency, Amount, and HasData. FX rates can be found in Application > Cube > FX Rates.

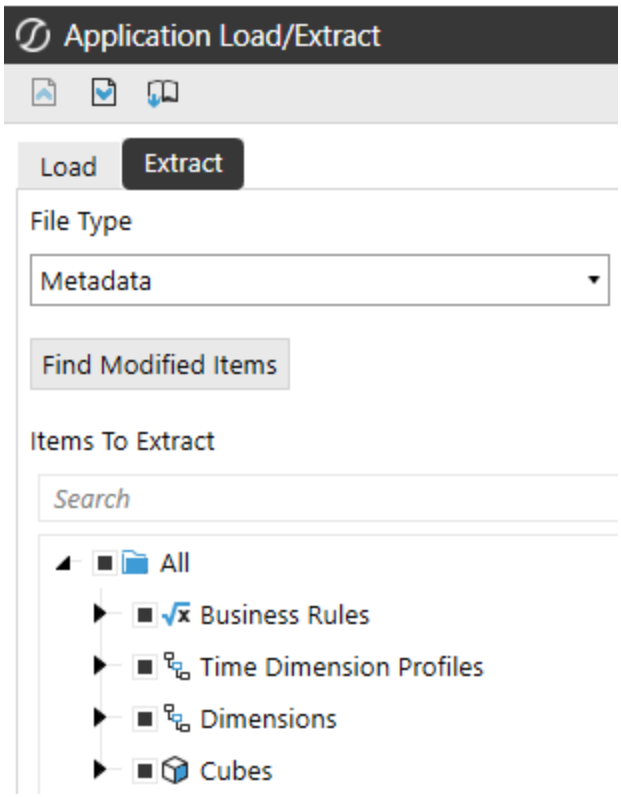
Extracting and Loading Dimensions

The Application Load/Extract can find dimension changes and export them into an xml to make the metadata migration process easier.

You can extract applications as an XML or zip file. If the file size is larger than 2GB, you must use a zip file.

When importing, you may have extracted an XML file that was larger than 2GB in size. You must do a zip extract to successfully load the file.

> **NOTE:** This feature only applies to dimensions.

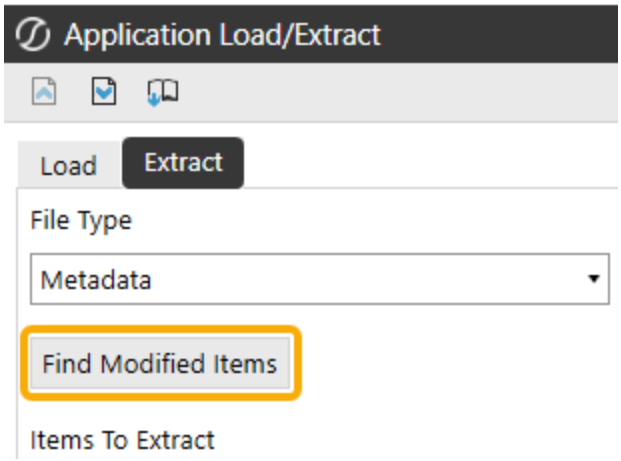

Application Load/Extract
Load
Extract
File Type
Metadata
Find Modified Items
Items To Extract
Search
All
Business Rules
Time Dimension Profiles
Dimensions
Cubes

Extracting Dimension Changes

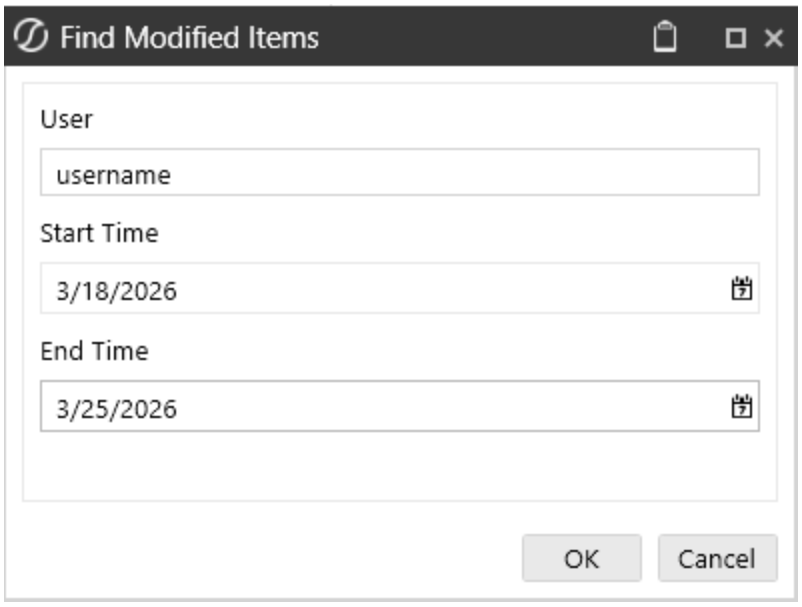
Modified items can consist of adding/deleting dimension members and adding/deleting dimension relationships.This enables users to find dimension changes by time and user.

To modify items, complete the following steps:

1. Select the **Find Modified Items** button.

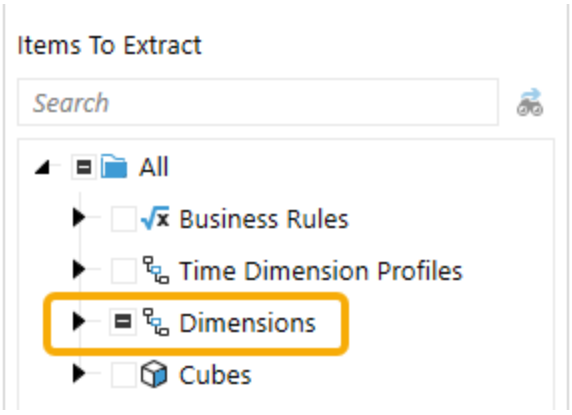


2. In the **Find Modified Items** dialog box, enter a user name and a range of time to find dimension member and dimension member relationship changes. To find changes by all users, leave the User field empty.

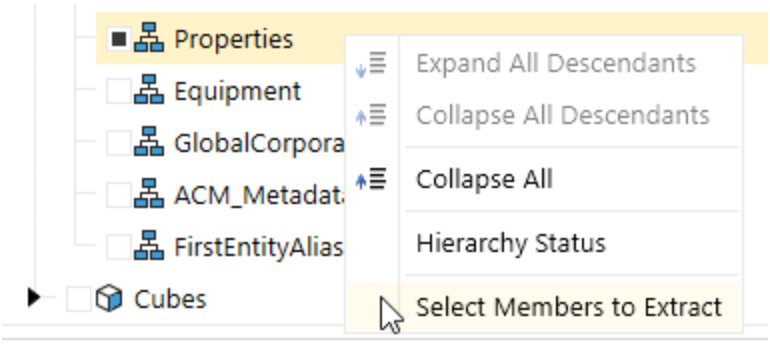


> **Example:** In the application's Dimension Library, a member was created, another member was deleted, and a relationship was deleted. Once the user and time filters are entered, select the **OK** button, and these changes are highlighted.

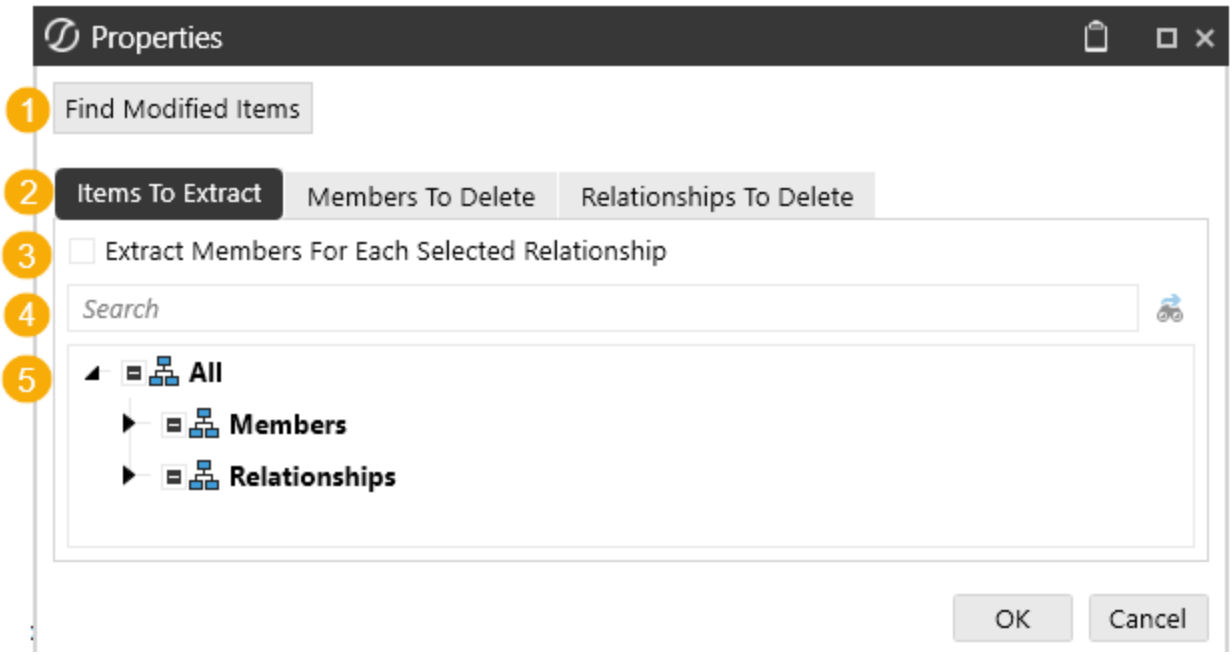
The hierarchy indicates where changes were made. The example below indicates that there was a partial change to the dimensions.



3. You can expand the appropriate node to see what was changed. In this example, Dimensions is expanded.
4. Right-click on the dimension name and select **Select Members to Extract**.

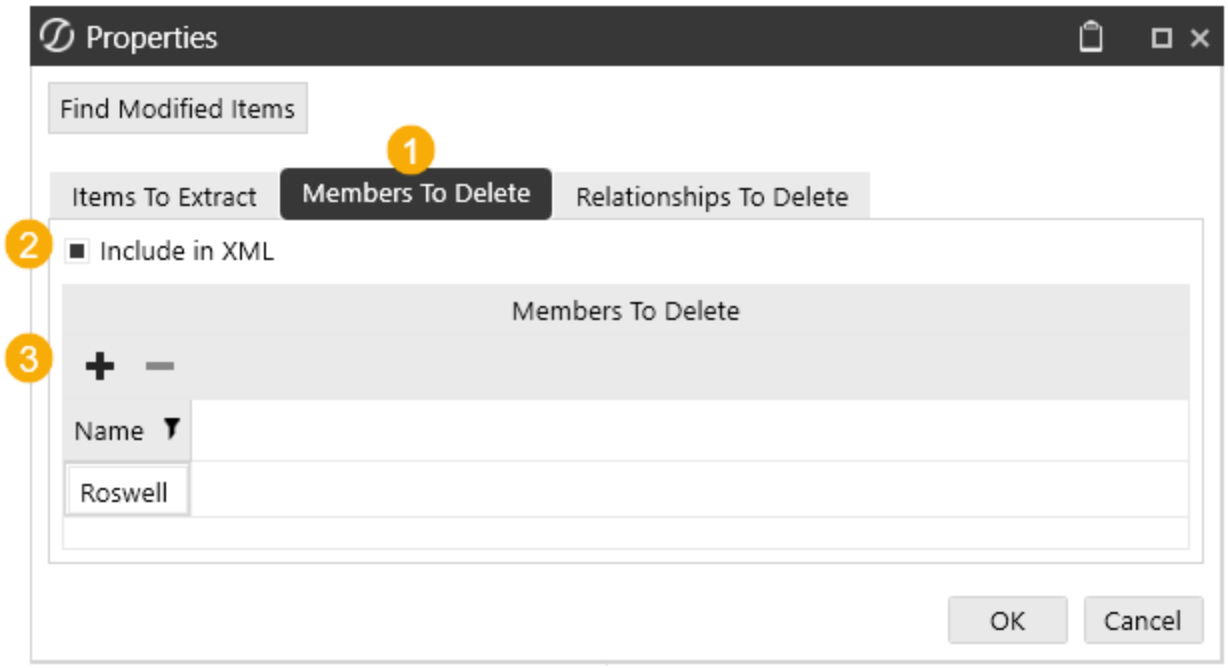


This launches an extraction dialog box for the specific dimension selected.



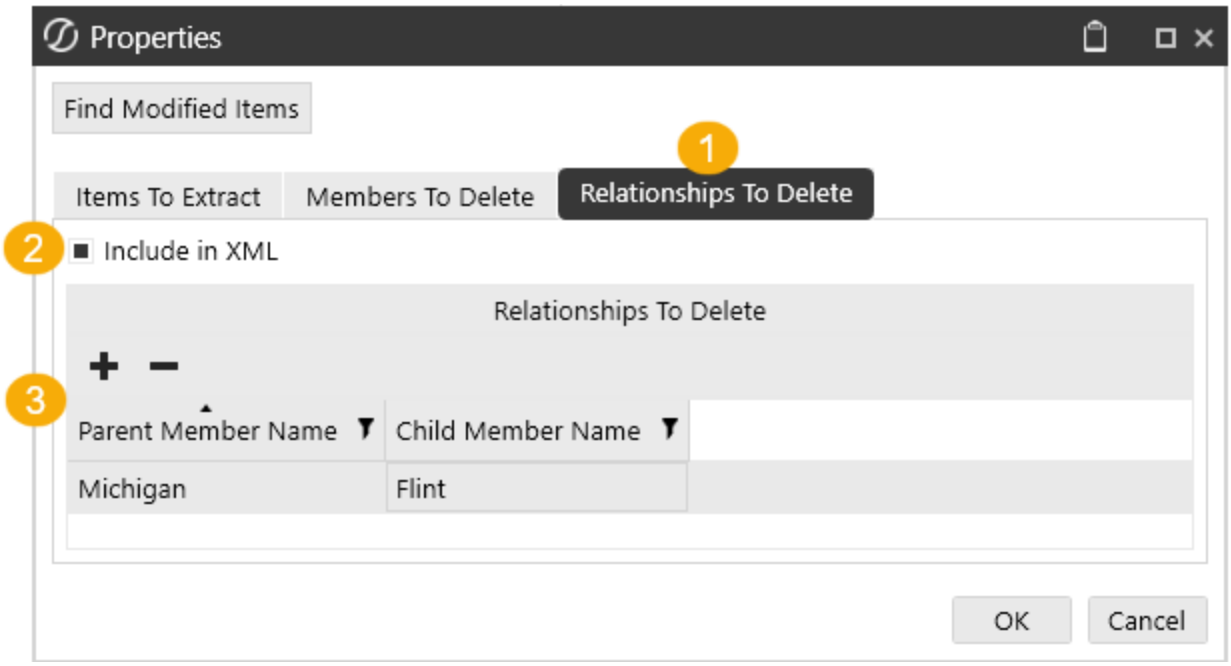
1. **Find Modified Items**: By default, everything is selected. Select the **Find Modified Items** button and enter the username and desired date range to find the specific changes made to the selected dimension.
2. **Items to Extract Tab**: This tab displays the changes found in the hierarchy. Users can also manually select members and relationships to extract.
3. **Extract Members for Each Selected Relationship Checkbox**: Select this checkbox to include the members used in the selected relationships.
4. **Search**: Search for a specific member in the selected dimension.
5. **Dimension Hierarchy**: Scroll through the dimension hierarchies to see where changes occurred. You can manually select or deselect members or relationships to extract.

> **TIP:** You can right-click on any parent member under Relationships and click **Select All Descendants** to select all child members within the hierarchy and include them in the extraction.



1. **Members To Delete Tab**: This tab displays any deleted members found within the user/time parameters.
2. **Include in XML Checkbox**: By default, this checkbox is deselected. Select this checkbox to include the displayed members in the extract. If selected, any members to delete will have an Action=Delete in the xml file and be deleted upon loading the xml into another application.
3. Any deleted members found within the user/time parameters display here. To manually add members to delete, select the **Insert Row** icon. To exclude a particular entry from the xml, select the line to exclude and select the **Delete Row(s)** icon.

> **NOTE:** Members with stored data cannot be deleted from an application.



1. **Relationships to Delete Tab**: This tab displays any deleted relationships found within the user/time parameters.
2. **Include in XML Checkbox**: Select this checkbox to include the displayed member relationships in the extract. If selected, any relationships to delete have an Action=Delete in the xml file and be deleted upon loading the xml into another application.
3. To manually add relationships to delete, select the **Insert Row** + icon and indicate the parent and child. To exclude a particular entry from the xml, select the line to exclude and select the **Delete Row(s)** − icon.

Loading the XML File

When importing an extracted XMLfile, it processes in the following order:

1. New or changed members
2. Deleted members
3. New relationships
4. Deleted relationships

The processing order is important because if any errors occur during the XML file load, OneStream applies as many modifications as it can up to the point of the error.

> **Example:** An XML file contains ten member changes, three deleted members, and one new relationship. If an error occurs when trying to delete the first member, the ten member changes still take place because they are processed first during the XML load. Any modifications in the XML prior to the error occur and any after the error do not.

> **NOTE:** If a user receives an error during the load process, the error must be resolved to complete the metadata migration.

Common Load Errors and Resolutions

Deleting a Member with Data

If a member without data is deleted from the source application, but that same member contains data in the destination application, an error occurs and the member will not be deleted. To resolve the error, do one of the following solutions:

- Clear the member's data in the destination application and reload the XML file.
- Create a new XML extract excluding that member from the file.
- Edit the XML file to exclude the member and the action.

Deleting a Member without Data

Proceed with care when deleting entity members. Even though the member may not have data, it may be in use as an Intercompany Partner. Ensure the member is not in use on any data records or on records as an Intercompany Partner.

Undefined Security Group

If a security group is assigned to a dimension member in the source application, but does not exist in the destination application, an error will occur. To resolve the error, do one of the following solutions:

- Create the security group in the destination application and reload the XML file.
- Create a new XML extract excluding this member and its changes from the file.
- Edit the XML file to exclude the member and the action.

Undefined FX Rate Type

If a FX Rate Type is assigned to a scenario member in the source application, but does not exist in the destination application, an error will occur. To resolve the error, do one of the following solutions:

- Create the FX Rate Type in the destination application and reload the file.
- Create a new XML extract excluding this member and its changes from the file.
- Edit the XML file to exclude the member and the action.

Invalid Characters in the XML File

If the XML file was edited and invalid characters were entered for a member name, an error will occur To resolve the error, do one of the following solutions:

- Make the member modifications in the source application and extract an XML file without invalid characters.
- Edit the XML file and remove the invalid characters.

Project Extract and Load

This extract is for Application Project Designers who are building solutions that span many artifacts, such as Workspace maintenance units, business rules, cubes, dimensions, Cube Views, and more. For example, designers building a solution to be hosted on OneStream Solution Exchange.

This application Extract and Load option allows all defined objects, such as dashboards and business rules, to be collected as a single file export package and later reloaded as a package. XFProject is used as a convenient way to organize OneStream Solutions or similar solutions into a folder structure which can be integrated with a version control system, such as Git. This could enable multiple team members to work on a solution simultaneously. The developer must create an XML file, which defines the contents of the project export.

Project File

The user must first manually define an XML file to support the export of objects as a project file. The file is saved with the file extension .xfProj and saved to a local project folder, which could also support a version control system.

The following example displays a sample file.

```
<xfProject topFolderPath="" defaultZipFileName="">
  <projectItems>
    <projectItem projectItemType="DashboardMaintenanceUnit" folderPath="" name="Corporate" includeDescendants="true" />
    <projectItem projectItemType="DashboardMaintenanceUnit" folderPath="" name="Corporate Templates" includeDescendants="true" />
    <projectItem projectItemType="DashboardComponent" folderPath="" name="btnSaveAndCalculate" includeDescendants="true" />
    <projectItem projectItemType="BusinessRule" folderPath="" name="FXRates" includeDescendants="true" />
  </projectItems>
</xfProject>
```

The file is structured in the following way:

- **xfProject**: The root node to start a .xfProj that contains two attributes:
    - **TopFolderPath**: Will create and define the starting folder location of where the specified files are extracted to.
    - **DefaultZipFileName**: Will create a standard default file name for .zip file extracts.
- **projectItems**: A list structure containing the project items needed to extract. No attributes are needed.
- **projectItem**: The item reflecting what is needed to extract from OneStream or load from the file system. It has four attributes:
    - **ProjectItemType**:
        - BusinessRule
        - Cube
        - CubeViewGroup
        - CubeView
        - CubeViewProfile
        - DashboardMaintenanceUnit
        - DashboardFile
        - DashboardString
        - DashboardParameter
        - DashboardGroup
        - DashboardAdapter
        - DashboardComponent
        - Dashboard

        - DashboardProfile
        - DataManagementGroup
        - DataManagementStep
        - DataManagementSequence
        - DataManagementProfile
        - DataSource
        - Dimension
        - TransformationRuleGroup
        - TransformationRuleProfile
    - **FolderPath**: The name of the sub folder where the project item type is extracted to.
    - **Name**: The name of the project item.
    - **IncludeDescendants**: The default is true and it only affects these project item types:
        - CubeViewGroup
        - DashboardGroup
        - DashboardMaintenanceUnit
        - DataManagementGroup

File Extract

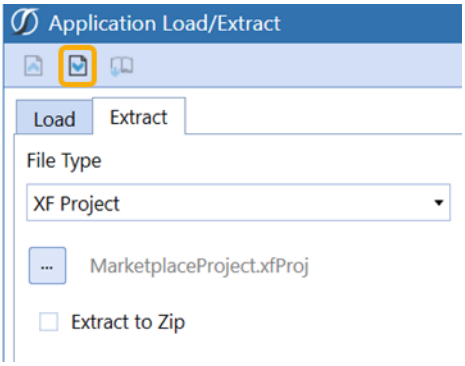
The .xfProj file is placed in a local folder, such as the user's desktop. The defined folder path folders will be generated here as the target location for application exports and loads.

There are two file extract options available on the Windows application:

- **zip**: This export option collects all objects defined in the .xfProj file as a Zip file to the location of the .xfproj file.
- **File**: This standard export exports all objects defined in the .xfProj file to the folderpath locations defined in .xfproj file.

To extract a file, complete the following steps:

1. Select the **Extract** tab.
2. Click the ellipses and select the .xfProj file.
3. Select or deselect the **Extract to Zip** checkbox as needed.
4. Select the **Extract** icon.



The following example shows the contents generated in the defined folder paths.

MarketplaceProject.xfProj

```
<xfProject topFolderPath="XFProjectFiles" defaultZipFileName="OSMarketplace">
    <projectItems>
        <projectItem projectItemType="DashboardMaintenanceUnit" folderPath="" name="Legal" includeDescendants="true" />

    </projectItems>
</xfProject>

```

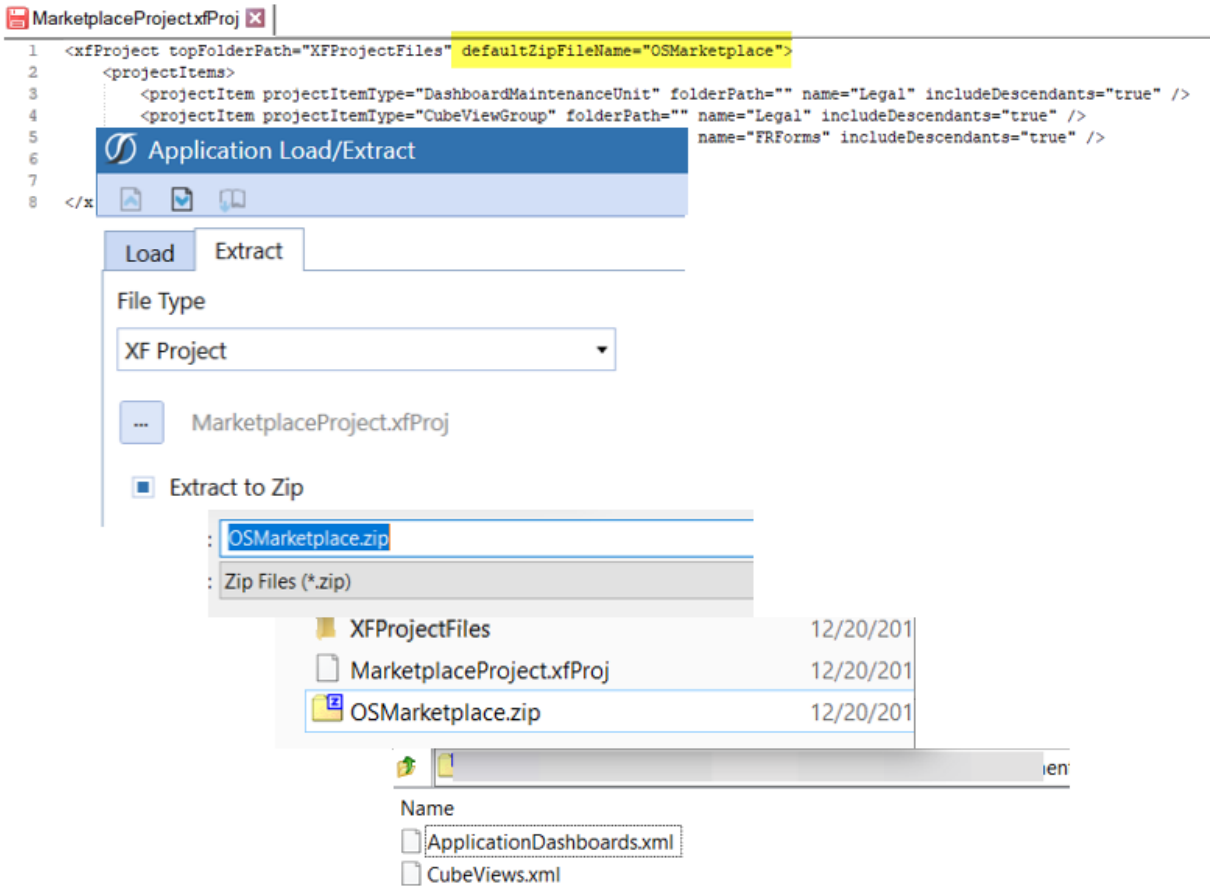
Name	Date modified	Type	Size
XFProjectFiles	12/20/2018 6:21 PM	File folder	
MarketplaceProject.xfProj	12/20/2018 5:58 PM	XFPROJ File	1 KB

Exports > XFProjectFiles > Application Dashboards > Maintenance Units > Legal

Name	Date modified	Type
Dashboard Adapters	12/20/2018 6:21 PM	File folder
Dashboard Components	12/20/2018 6:21 PM	File folder
Dashboard Files	12/20/2018 6:21 PM	File folder
Dashboard Groups	12/20/2018 6:21 PM	File folder
Dashboard Parameters	12/20/2018 6:21 PM	File folder
Dashboard Strings	12/20/2018 6:21 PM	File folder

Zip Extract

The zip file extract creates an application zip file containing all the objects defined.



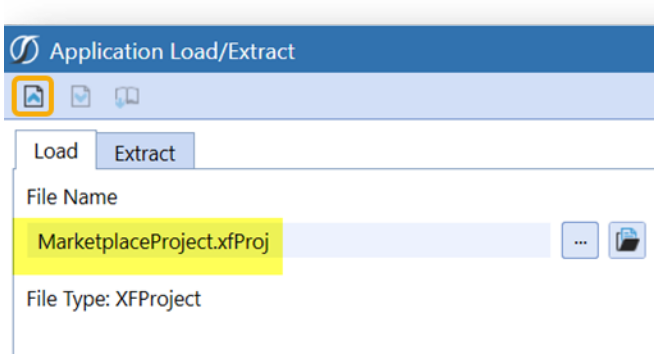
File Load .xfProj

The file load using the defined .xfProj file provides a seamless link to the project files. When loading an .xfProj file, the user is presented with options to Merge or Replace the target files. The only files affected are those defined by the .xfproj file.

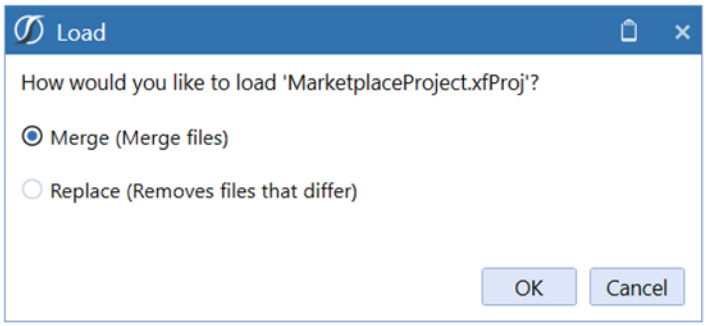
> **NOTE:** If you select Replace, it only removes files that differ for CubeViewGroups, DataManagementGroups, DashboardMaintenanceUnits, and DashboardGroups. For all other items, such as, business rules or extensibility rules, if you select Replace it will act as a Merge.

To load a file, complete the following steps:

1. Select the **Load** tab.
2. Select the ellipses and select the .xfProj file.
3. Select the **Load** icon.



4. Select **Merge** or **Replace**.



Zip Load

The zip file load functions as any other application file load. The contents of the file are merged into the application. The zip file load is not supported by alternative merge or replace file load options.

Spreadsheet

The OneStream Windows application Spreadsheet feature enables users to incorporate spreadsheet functionality experienced by Excel Add-In users without requiring Excel on their computer. It can be used for ad hoc querying/reporting, analysis, data entry, and formatted reports. Similar to the Excel Add-in, Spreadsheet leverages OneStream's re-usable Cube Views for fast and easy analysis. See the *Excel Add-in Guide*.

Spreadsheet provides the following benefits:

- Change point-of-view, interact with Forms, assign Cube Views, drill through to source data and update workflow status
- Utilize familiar retrieve functions for ad-hoc reporting and analysis
- Eliminate risk of errors and duplication of efforts with standardized and centralized spreadsheet controls
- Safely edit, update, and analyze data as spreadsheet forms respect application and workflow and security
- Eliminate spreadsheet maintenance when metadata changes because Cube Views are read dynamically
- Sheet-based calculations remain intact even when rows or columns are added
- Include multiple Cube View results in the same sheet

Spreadsheet can perform many accomplished by the Excel Add-in, with some limitations. These limitations include, but are not limited to:

- Macros
- Solver
- Document properties
- No option to insert copied/cut cells in spreadsheet
- Preview format does not update for format types Effects (Superscript, Subscript, Strikethrough)
- Ability to manage conditional formatting. Color scales are reversed.
- Shift+End
- Spreadsheet border button does not store last selection
- Text Direction option from Alignment formatting
- Underline formatting does not appear same as in Excel
- Not all chart options are available in Spreadsheet
- Ability to change chart colors in Spreadsheet
- References to other sheets in Spreadsheet is a different process
- Ctrl+PageUp / Ctrl+PageDn correctly changes the tab but does not reset tab focus in spreadsheet
- CTRL+N creates a new spreadsheet, however, if an existing spreadsheet was opened with unsaved changes, it closes the existing spreadsheet without saving edits made since last save

Some functionality is currently not supported in Spreadsheet. Functionality that is not supported in the Spreadsheet feature is listed below:

- Fill number formatting
- The ability to format part of a cell with different formatting
- Justify alignment option
- Managing Rules under spreadsheet conditional formatting
- Spreadsheet alignment cell format "Center Across Selection"
- Undo/Redo

The following Excel Column Charts are incompatible and unavailable with Spreadsheet:

- Stacked 3-D
- 100% Stacked 3-D
- Stacked Cylinder
- 100% Stacked Cylinder
- Stacked Cone
- 100% Stacked Cone
- Stacked Pyramid
- 100% Stacked Pyramid

The following Excel Line chart is incompatible and unavailable with Spreadsheet:

- 3-D Line

The following Excel Bar charts are incompatible and unavailable with Spreadsheet:

- Stacked 3-D Bar
- 100% Stacked 3-D Bar
- Stacked Horizontal Cylinder

- 100% Stacked Horizontal Cylinder
- Stacked Horizontal Cone
- 100% Stacked Horizontal Cone
- Stacked Horizontal Pyramid
- 100% Stacked Horizontal Pyramid

Microsoft Excel 2016 Chart Types

There are seven new chart types added to the Spreadsheet component in the Windows application. These charts were first introduced to Microsoft Excel 2016 and are now available in the Spreadsheet component as an alternative to using Excel. This is available to administrators or general users with security rights to access it.

Chart Types

Microsoft Excel 2016 introduced several chart types that help you visualize financial, statistical, and hierarchical data. They include:

- Box and Whisker
- Waterfall
- Histogram
- Pareto
- Funnel
- Sunburst
- Treemap

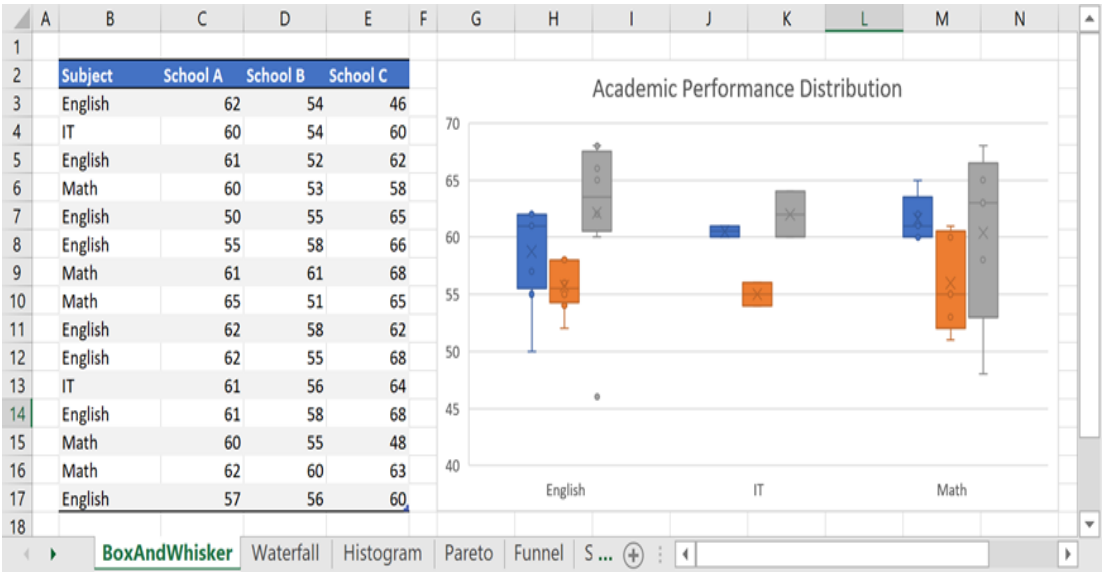
You can add charts to Spreadsheet in the same manner as any other chart type. Chart styles enable you to quickly change chart appearance. Styles change the background fill, specify the color of the data series, and apply different shape effects and outlines to the chart.

To apply Excel 2016 chart types to a predefined chart style, right-click on the chart and select one of the available styles.

Box and Whisker

A box and whisker chart shows distribution of data in quartiles, highlighting the mean and outliers. Boxes may have lines extending vertically called whiskers. These lines indicate variability outside the upper and lower quartiles, and any point outside those lines or whiskers is considered an outlier.

Box and whisker charts are commonly used in statistical analysis. For example, you could use a box and whisker chart to compare medical trial results or teachers' test scores.



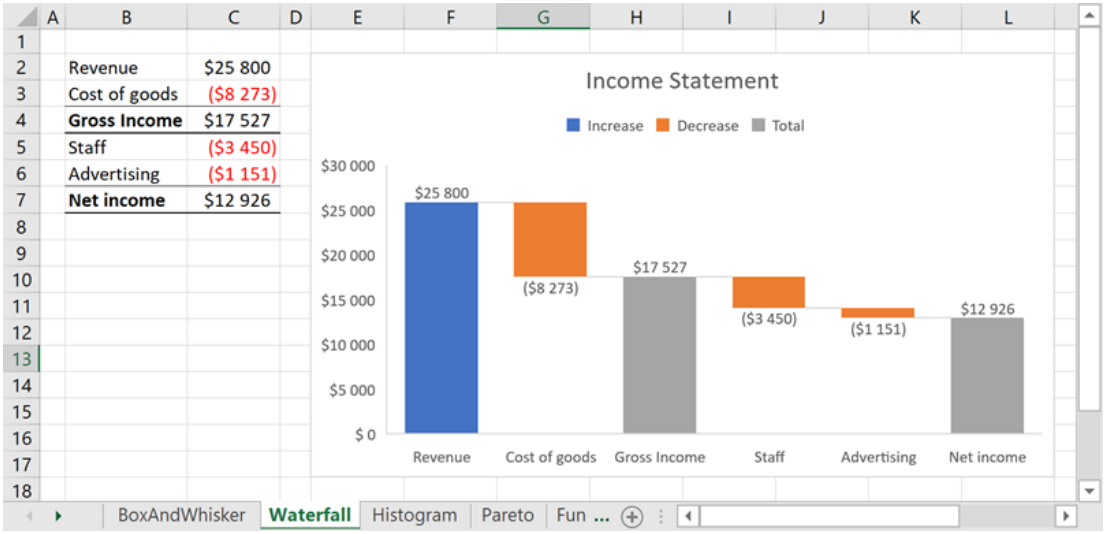
Subject	School A	School B	School C
English	62	54	46
IT	60	54	60
English	61	52	62
Math	60	53	58
English	50	55	65
English	55	58	66
Math	61	61	68
Math	65	51	65
English	62	58	62
English	62	55	68
IT	61	56	64
English	61	58	68
Math	60	55	48
Math	62	60	63
English	57	56	60

Waterfall

A waterfall chart shows a running total as values are added or subtracted. It is useful for understanding how an initial value, like net income, is affected by a series of positive and negative values. The columns are color coded so you can quickly tell positive from negative numbers. The initial and the final value columns often start on the horizontal axis, while the intermediate values are floating columns.

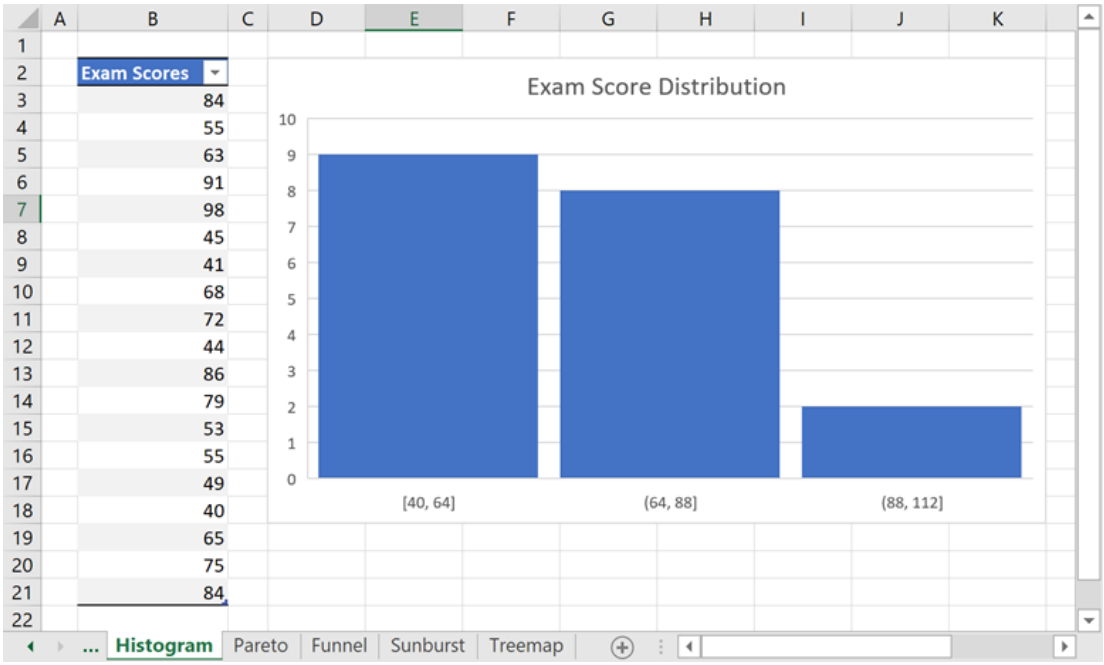
If your data includes values that are considered subtotals or totals, like net income, you can set those values so they start on the horizontal axis at zero and don't float. To do this, complete the following steps:

1. Right-click the chart and select **Set as Total**.
2. In the dialog box, select the checkbox for values in the dataset to be set as total.
3. To make the column float again, deselect the **Set as total** checkbox.



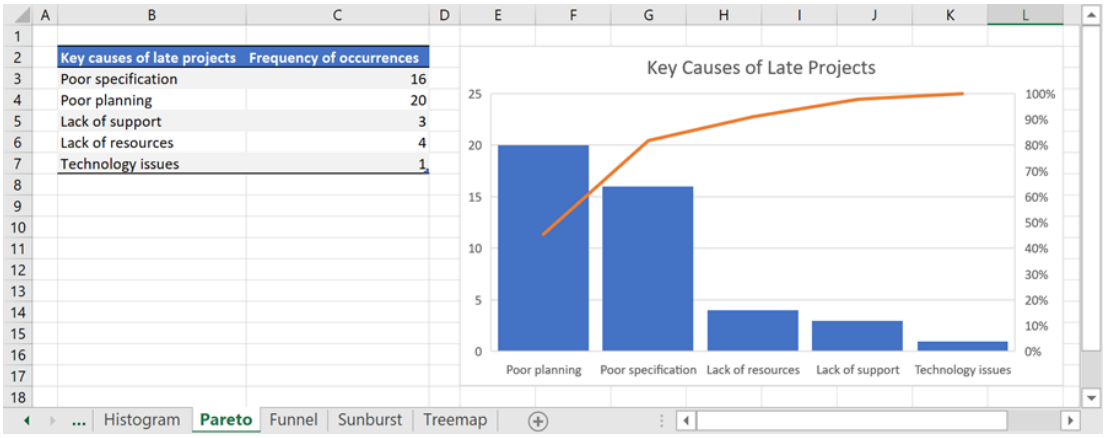
Histogram

Data plotted in a histogram chart shows the frequencies within a distribution. Each column of the chart is called a frequency bin.



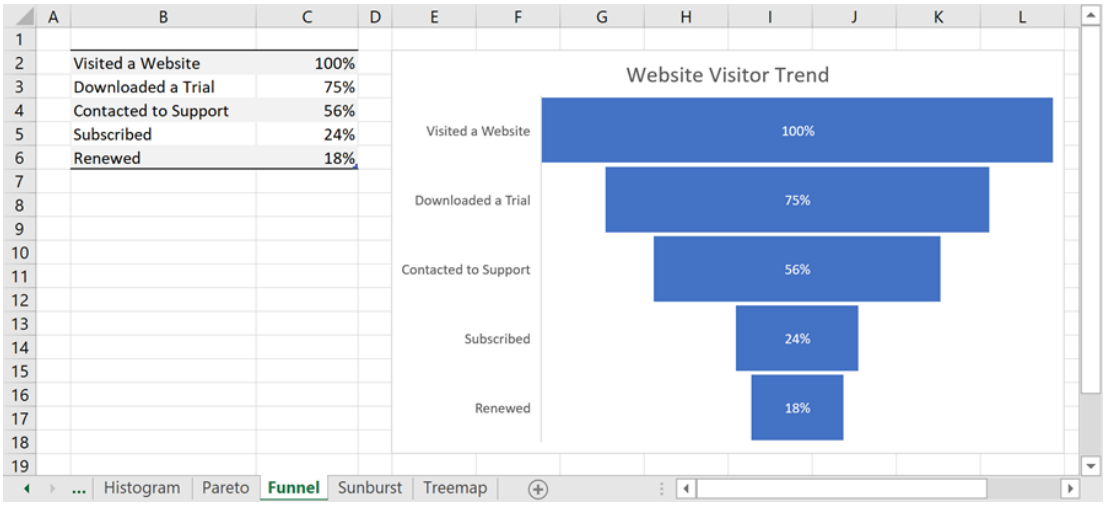
Pareto

A pareto chart is a sorted histogram that contains columns sorted in descending order along with a line representing the cumulative total percentage.



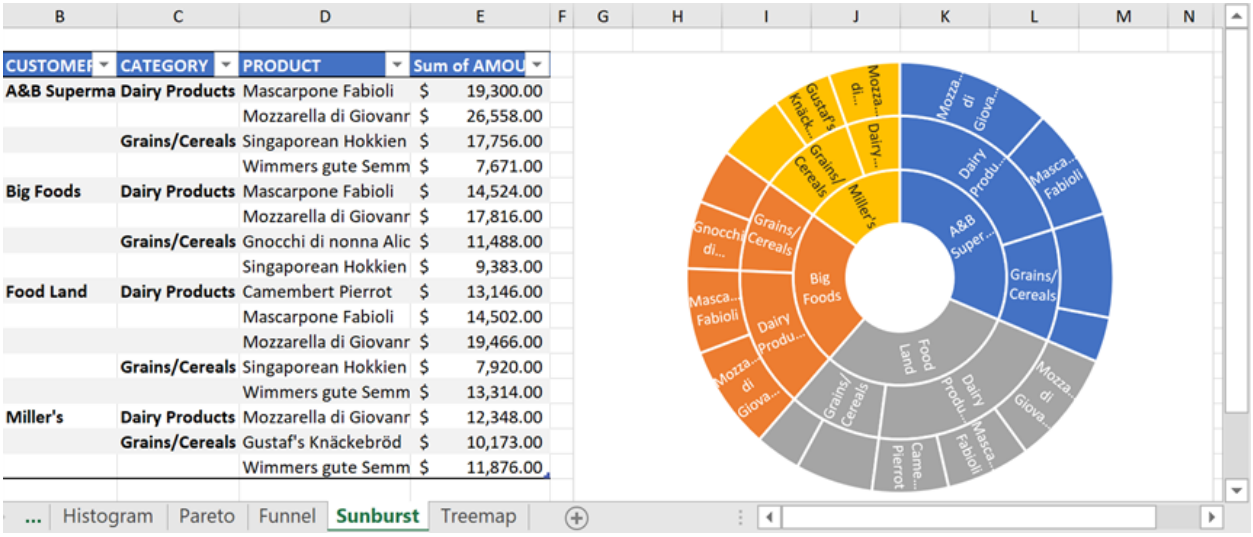
Funnel

Funnel charts show values across multiple stages in a process. Typically, the values decrease gradually, causing the bars to resemble a funnel. For example, you could use a funnel chart to show the number of sales prospects at each stage in a sales pipeline.



Sunburst

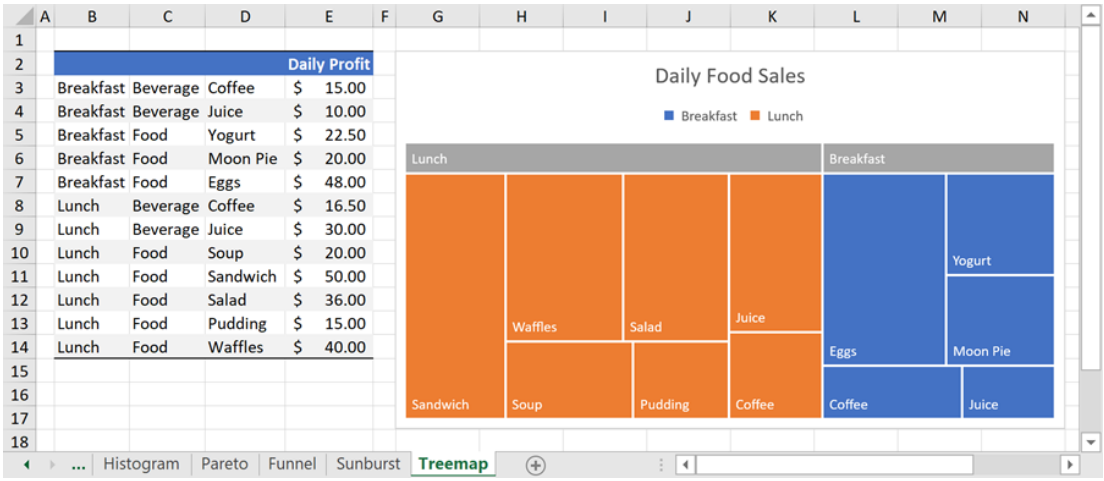
The sunburst chart displays hierarchical data and can be plotted when empty (blank) cells exist within the hierarchical structure. Each level of the hierarchy is represented by one ring or circle with the innermost circle as the top of the hierarchy. A sunburst chart without hierarchical data (one level of categories), looks similar to a doughnut chart. However, a sunburst chart with multiple levels of categories shows how the outer rings relate to the inner rings. The sunburst chart is most effective at showing how one ring is broken into its contributing pieces.



CUSTOMER	CATEGORY	PRODUCT	Sum of AMOU
A&B Superma	Dairy Products	Mascarpone Fabioli	$ 19,300.00
		Mozzarella di Giovanr	$ 26,558.00
	Grains/Cereals	Singaporean Hokkien	$ 17,756.00
		Wimmers gute Semm	$ 7,671.00
Big Foods	Dairy Products	Mascarpone Fabioli	$ 14,524.00
		Mozzarella di Giovanr	$ 17,816.00
	Grains/Cereals	Gnocchi di nonna Alic	$ 11,488.00
		Singaporean Hokkien	$ 9,383.00
Food Land	Dairy Products	Camembert Pierrot	$ 13,146.00
		Mascarpone Fabioli	$ 14,502.00
		Mozzarella di Giovanr	$ 19,466.00
	Grains/Cereals	Singaporean Hokkien	$ 7,920.00
		Wimmers gute Semm	$ 13,314.00
Miller's	Dairy Products	Mozzarella di Giovanr	$ 12,348.00
	Grains/Cereals	Gustaf's Knäckebröd	$ 10,173.00
		Wimmers gute Semm	$ 11,876.00

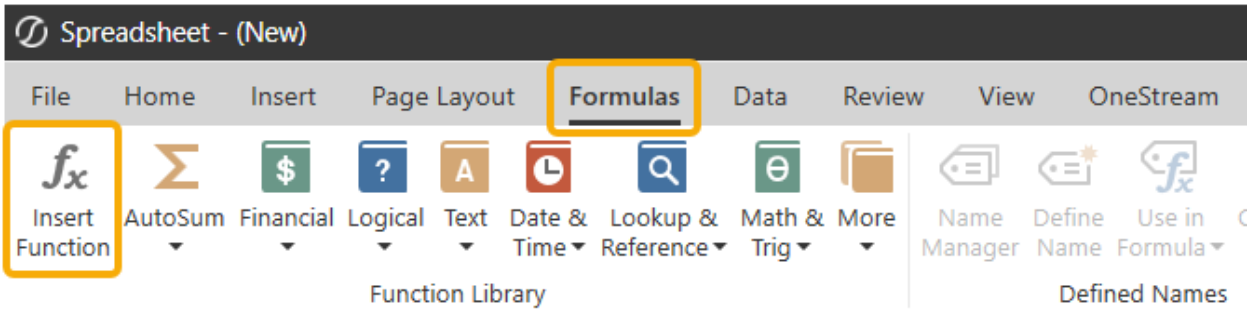
Treemap

The treemap chart provides a hierarchical view of your data and an easy way to compare different levels of categorization. The treemap chart displays categories by color and proximity and can show data which might be difficult to display with other chart types. The treemap chart is plotted when empty (blank) cells exist within the hierarchical structure. Treemap charts are good for comparing proportions within the hierarchy.

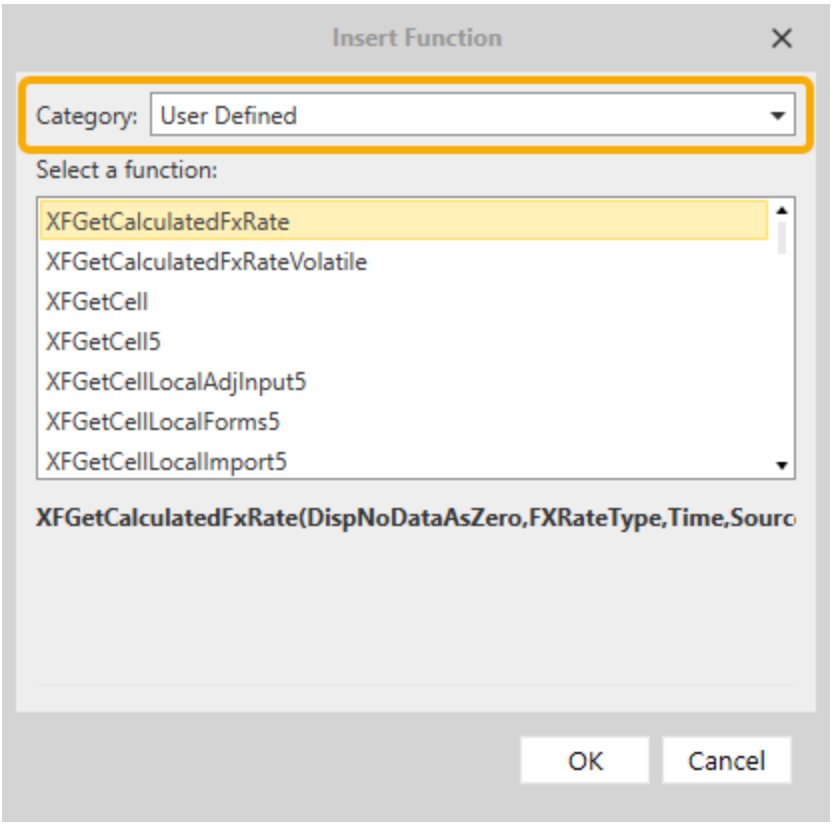


Retrieve Functions

Retrieving and changing data can be done by using functions. To see the functions and their Parameters, open Spreadsheet and select the **Formulas** tab. Select **Insert Function**.



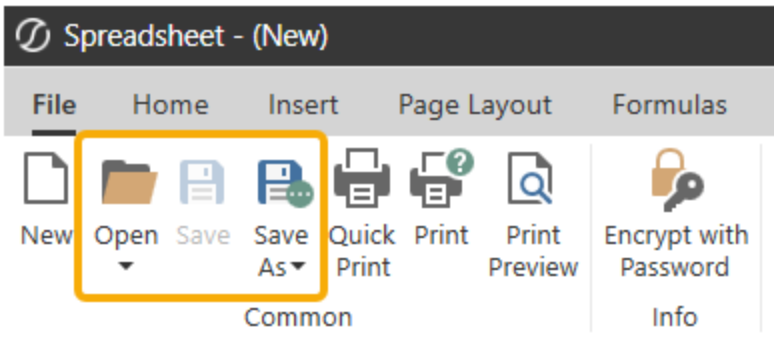
The Insert Function dialog box displays. From the **Category** drop-down menu, select **User Defined**.



See "Retrieve Functions" in the *Excel Add-In Guide*.

Open and Save As Functionality

There are additional options for opening and saving files within Spreadsheet .



Open

Select a file type to open. You can choose from the following options:

- **Local Folder**: Select a file to be opened from a location on the local computer/network.
- **OneStream File System**: Select a file to be opened from a location within the File Explorer.

> **NOTE:** Displayed files can also be opened from here or from OnePlace > Documents by right-clicking and selecting one of the three options: Open in Spreadsheet Page; Open (opens file directly in Excel if the application is found on the local computer) or Open With… (user specifies program).

- **Application Workspace File**: Select a file to be opened from an Application Workspace.
- **System Workspace File**: Select a file to be opened from a System Workspace.

Save As

Select a file type to be saved as. You can choose from the following options:

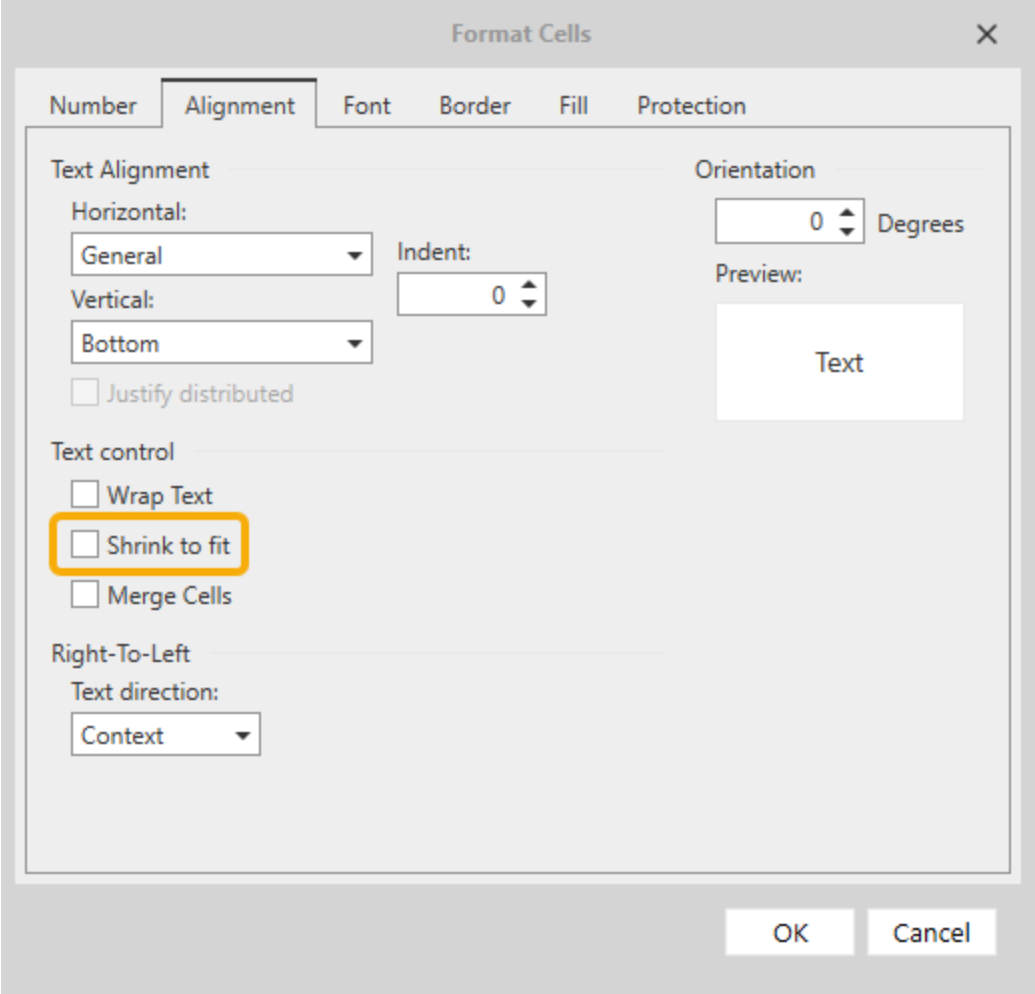
- **Local Folder**: Select a location on local computer/network to save a file to.
- **OneStream File System**: Select a location within File Explorer to save a file to.
- **Application Workspace File**: Select an Application Workspace location to save a file to.
- **System Workspace File**: Select System Workspace location to save a file to.

Reduce Cell Text Size

You can automatically reduce text size to fit inside a spreadsheet cell. This eliminates the need to resize the column width and prevents text from being cut off when it exceeds the cell width.

To reduce text size, complete the following steps:

1. Inside the spreadsheet, select a single cell, multiple cells, or all cells.
2. Right-click on the selected cells and select **Format Cells** to display the Format Cells dialog box.
3. In the **Alignment** tab, under **Text Control**, Select the **Shrink to fit** checkbox.

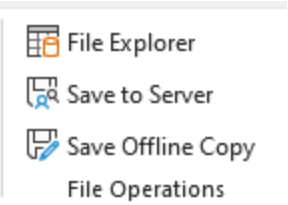


4. Select the **OK** button.

OneStream Ribbon

Spreadsheet includes the OneStream ribbon. See "OneStream Ribbon" in the *Excel Add-In Guide* for an explanation of the items within Spreadsheet that function identically to those in Excel Add-In.The items in this section function differently than Excel Add-in or do not exist in Spreadsheet.

File Operations

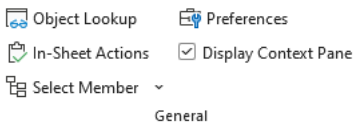


The File Operations grouping in the Excel Add-In does not exist in Spreadsheet. The Save to Server and File Explorer ribbon options are not available because they are integrated with the OneStream application within Spreadsheet. Save an Offline Copy is under the General Grouping in Spreadsheet.

Explore

Create Copy of Selected Quick View is available in Spreadsheet and copied Quick Views can only be pasted within the same workbook. See "Quick Views" in the *Excel Add-In Guide* for more options.

General



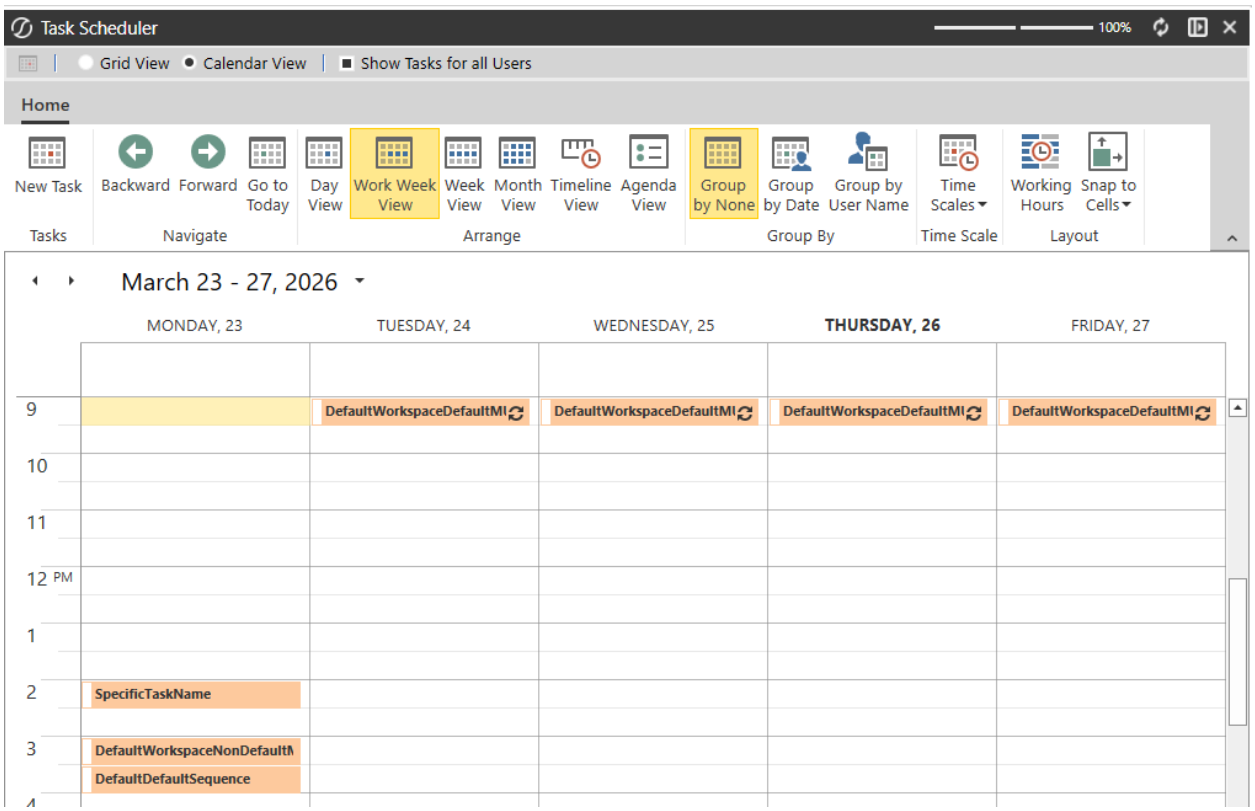
The Display Context Pane option does not exist in the Spreadsheet feature because it is not needed. The OneStream task pane automatically appears when using the Spreadsheet feature.

Under Preferences, the General options are not needed in the Spreadsheet for a Windows application only tool. These options manage how Microsoft Sign In is handled, how Macros are handled, and how data is refreshed in an open workbook. The Spreadsheet feature does not utilize this functionality.

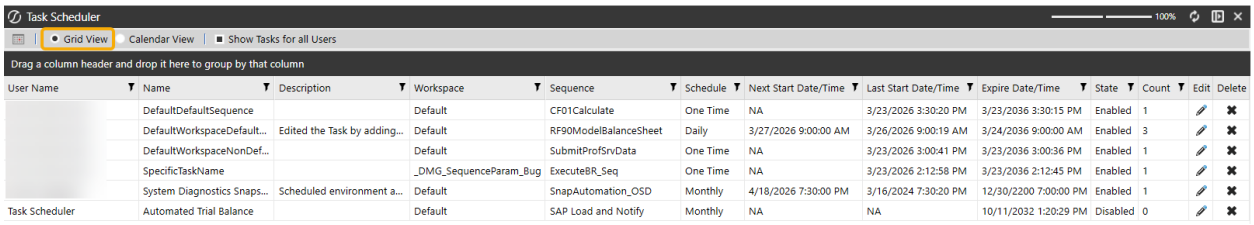
Task Scheduler

Task Scheduler enables you to schedule data management sequences that execute a data management step within the application. If the sequence does not have a step, the job will fail.

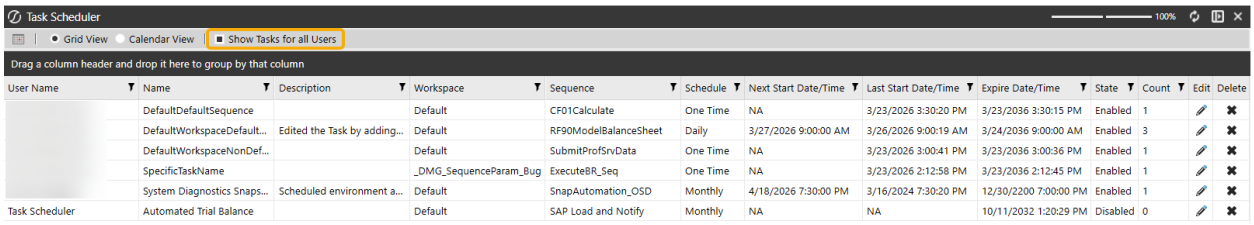
The default view for Task Scheduler is the calendar view.



To change the view to a grid view, select the **Grid View** radio button. This view is initially blank until sequences are scheduled. Whatever view you are currently on is the view you will come back to the next time you open the page.



The Show Tasks for all Users checkbox is selected by default. Deselect the checkbox to show tasks you've created.

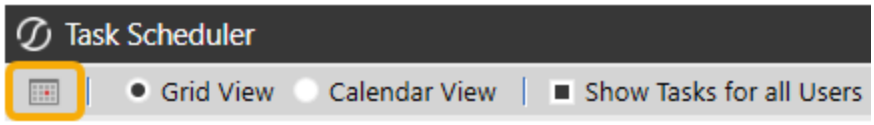


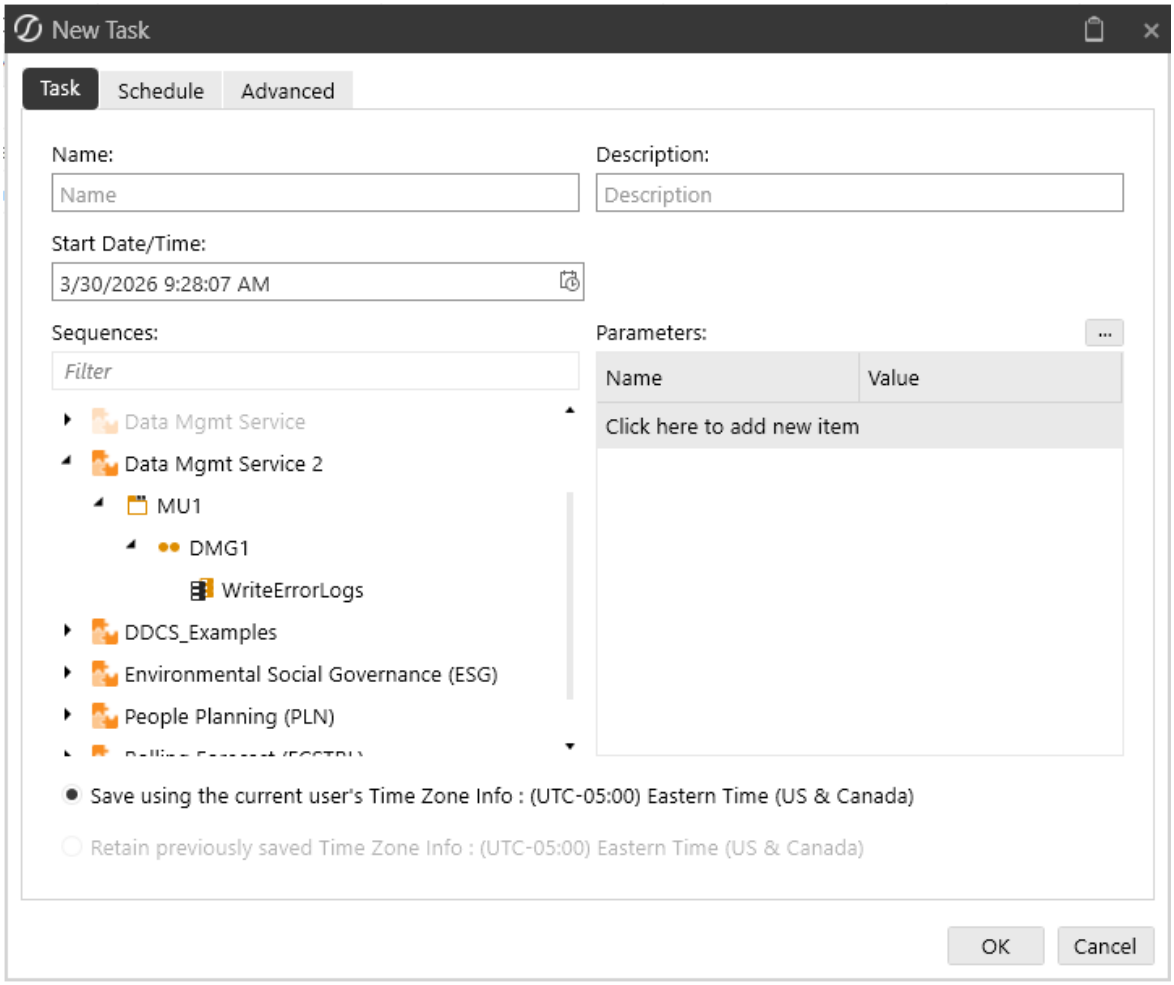
Manage Tasks

> **NOTE:** When upgrading to Platform Version 8.4.0, existing tasks must be edited by opening the task and selecting the **OK** button. Doing so applies the time zone logic without recreating tasks.

To create a new task, complete the following steps:

1. Select the **Create Scheduled Task** icon to display the New Task dialog box.



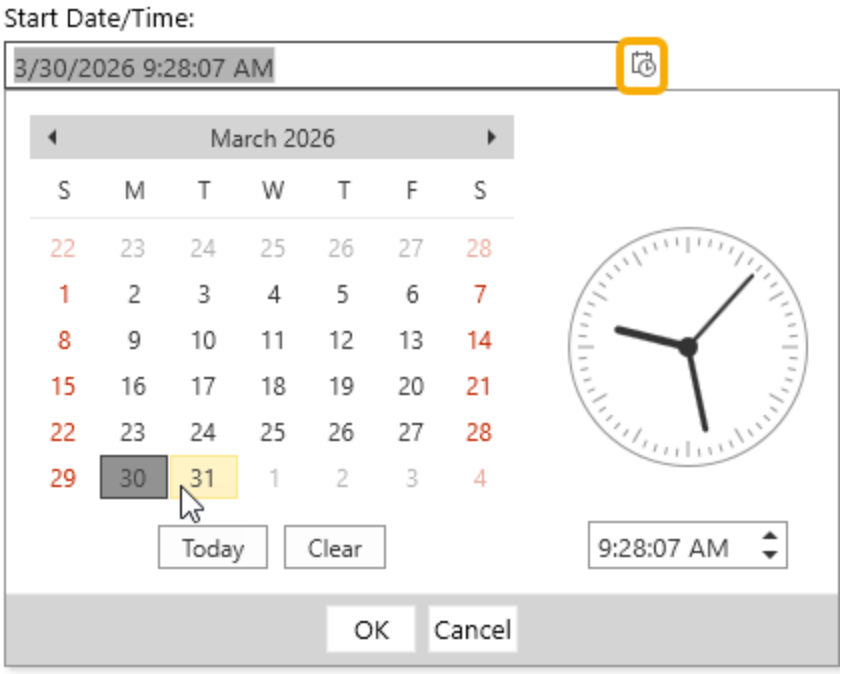


2. In the Task tab, enter the following details:

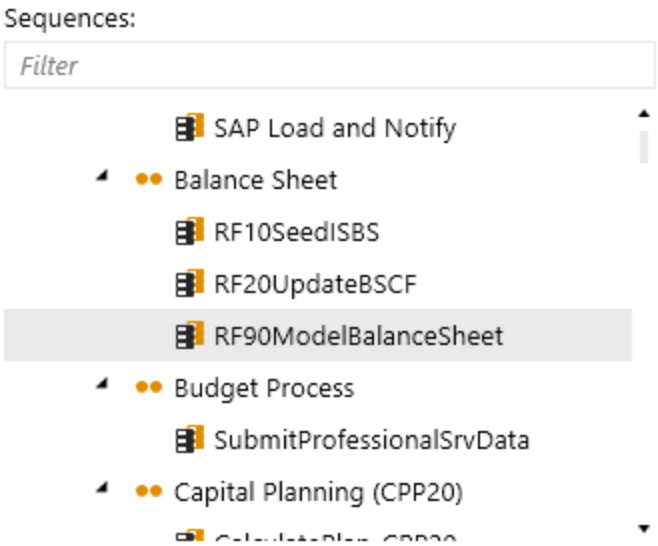
a. Name the task.

b. Enter a description for the task.

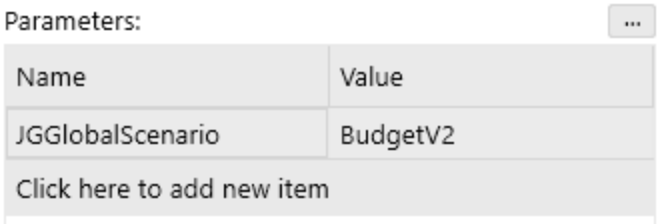
c. Set a start date/time. Ether type a value in the field or select the icon and use the dialog box. Set it to the time you want the task to start running. If you do not specify a time, the value defaults to the current date and time.



d. In the Sequences pane, select the data management sequence from the navigation tree.

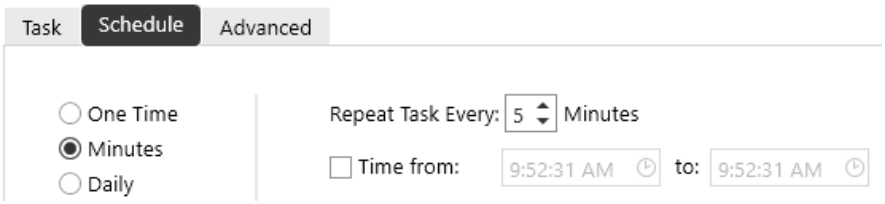


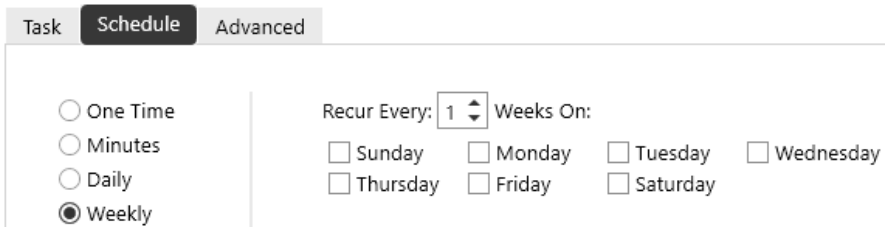
e. When you select a sequence, if there are parameters that have been set up for the sequence, they will display in the parameters pane. You can add additional validated parameter settings. If the parameters are not valid, the job will fail.

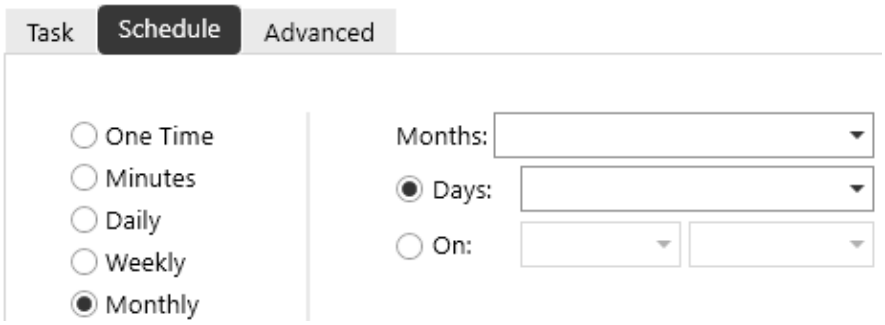


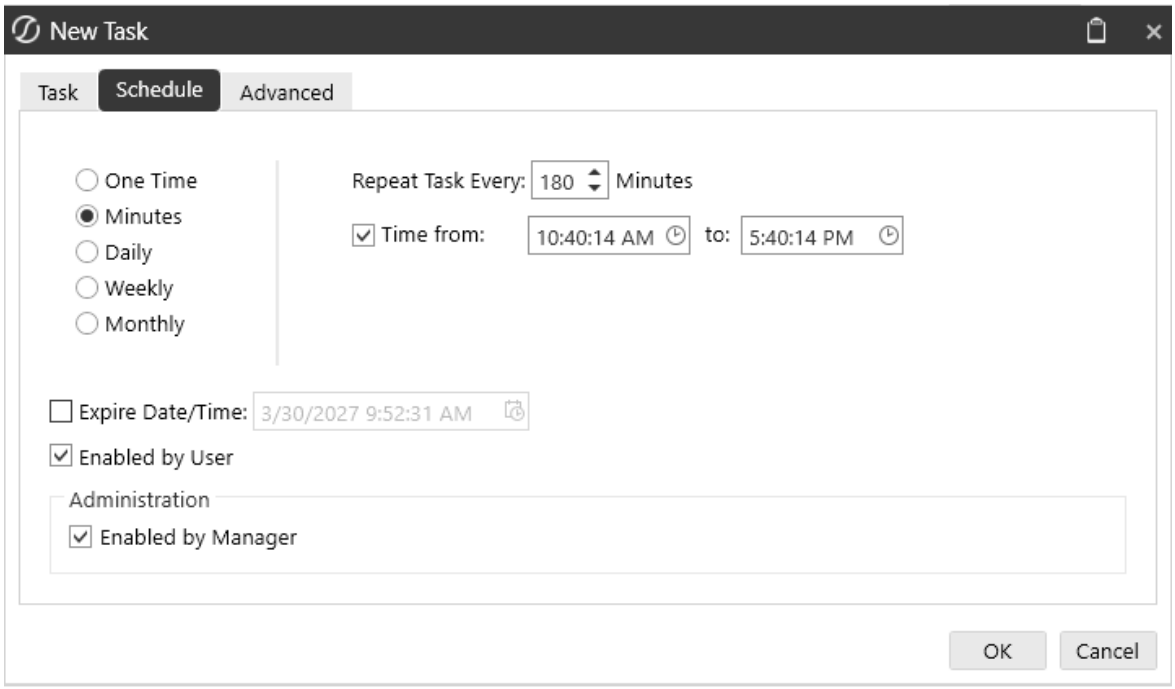
> **NOTE:** When new weekly and monthly recurring tasks are scheduled, they are created in the inputting user's local time zone and saved in the UTC time zone. The Save using the current user's Time Zone icon radio button is selected by default.

3. In the **Schedule** tab, make your scheduling selections:

Property	Description
One Time	This triggers the task to run once based on the time in the Start/Date Time field in the Task tab.
Minutes	This triggers the task to run on a recurring basis from 5 – 180 minutes. You can set tasks to run during predetermined times by typing the start time in the Time from field, then typing the stop time in the To field. By default, the Time from field is unchecked.  If you create a task, for example, that is set to run every 30 minutes, starting at 2:30 PMand ending at 5:30 PM, the first run of the task occurs at 2:30 PM and runs every 30 minutes during that timeframe. **NOTE:** Calendar entries are created in the Calendar view even though they may fall outside of the selected run-time frame. This means, for example, that a task scheduled to run every 30 minutes between the hours of 2:00 PM and 5:00 PM, will display all day every 30 minutes.

Property	Description
Daily	Select this to choose how many times the task recurs. 
Weekly	Select this to choose how many times the task recurs and the days that it runs. 

Property	Description
Monthly	Select this to choose how many times the task recurs and the days that it runs. 
Expire Date/Time	Select the checkbox if needed and select a date/time for expiration.
Enabled by User	The task is enabled to run if the checkbox is selected.
Enabled by Manager	Administration enabled if checked to run.



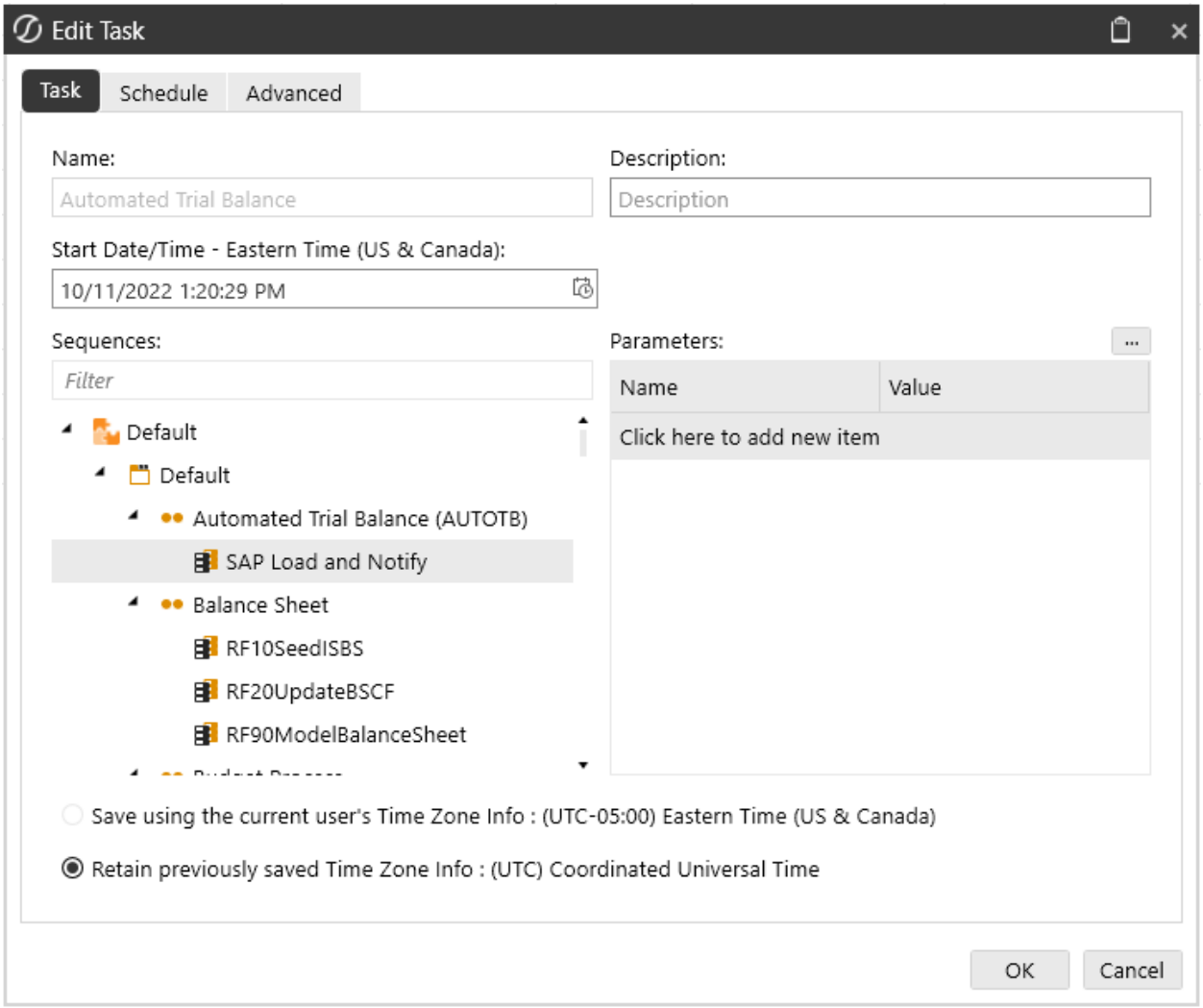
4. In the Advanced tab, you can set the number of times to retry a task if it fails. The maximum is three.



5. Select the **OK**button and the new task is added to the grid view and the calendar view.

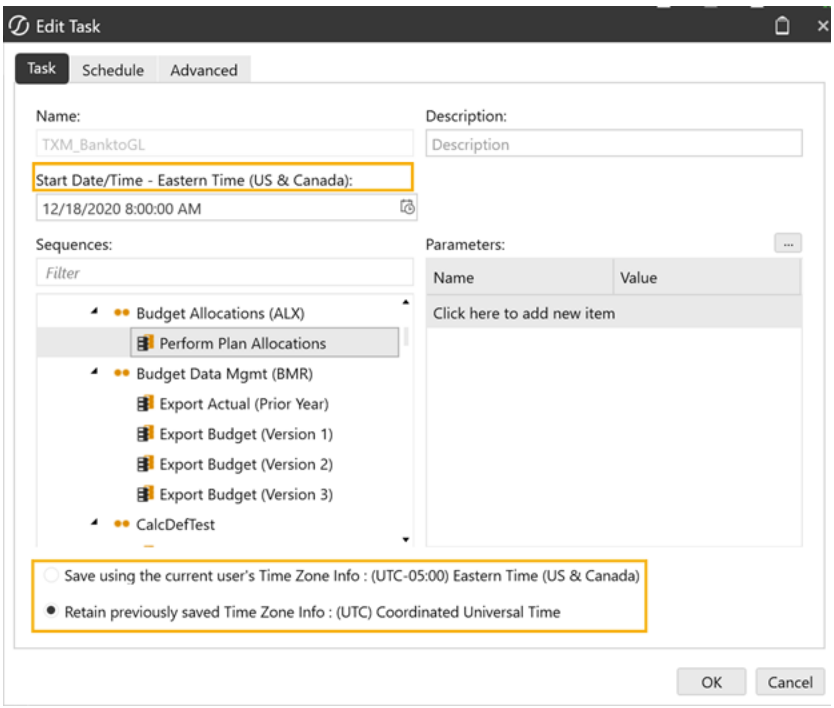
Edit a Task

To edit a task, double-click the task or select the **Edit** icon. The tasks' name cannot be edited, and the Start Date/Time is your local time.

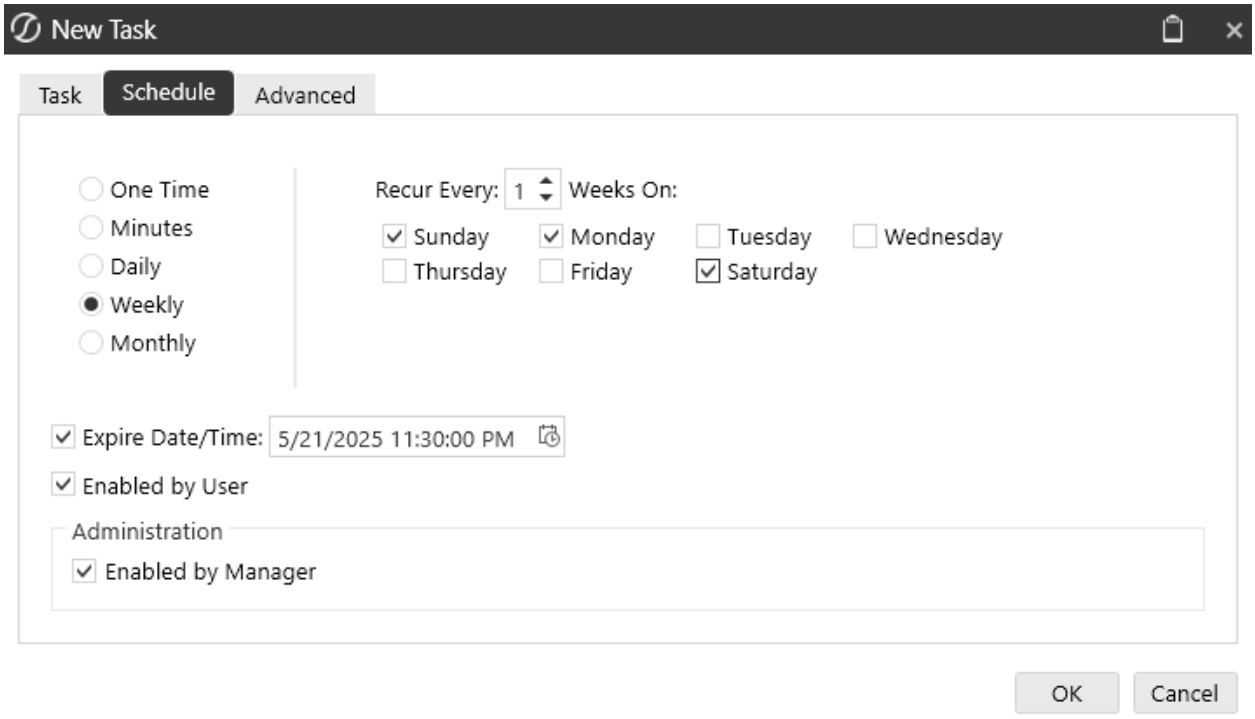


The Start Date/Time label also displays the user's current time zone to provide the user the time zone . For example, Eastern Time (US & Canada).

NOTE: Run logic begins with the Start Date then the reoccurring day of the week.



The following example details a sample task schedule.



NOTE: The Enabled By Manager checkbox can only be changed by an administrator.

Start Date/Time: 5/7/2024 11:55:00 AM (Wednesday)

Recur Every: 2 Weeks

On: Sunday, Monday, Friday

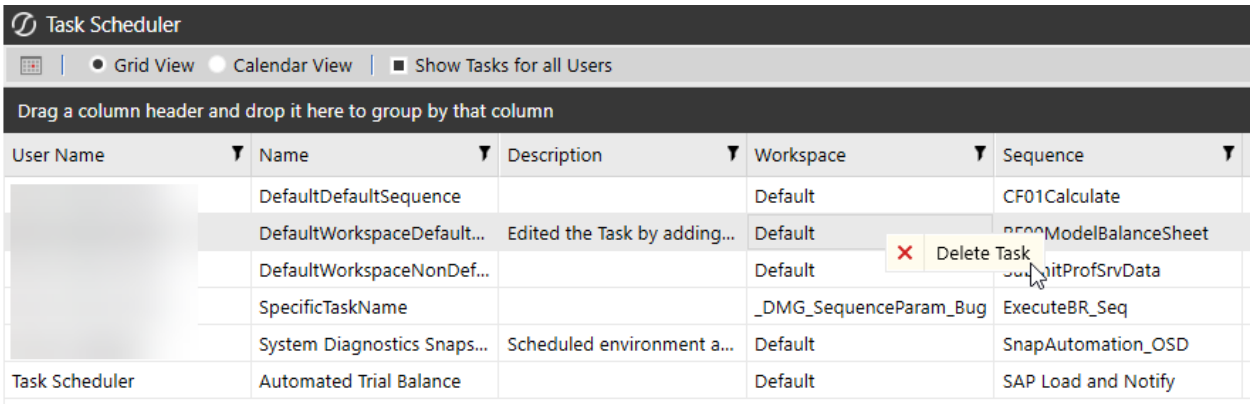
Expire Date/Time: 5/28/2024 11:30:00 AM

Expected Result:

Friday 5/9	Run
Sunday 5/11	Run

Monday 5/12	Run
Friday 5/16	Skip
Sunday 5/18	Skip
Monday 5/12	Skip
Friday 5/23	Run
Sunday 5/25	Run
Monday 5/26	Run

To delete a task, you can select the **Delete** ✖ icon or right-click the task and select **Delete Task**.

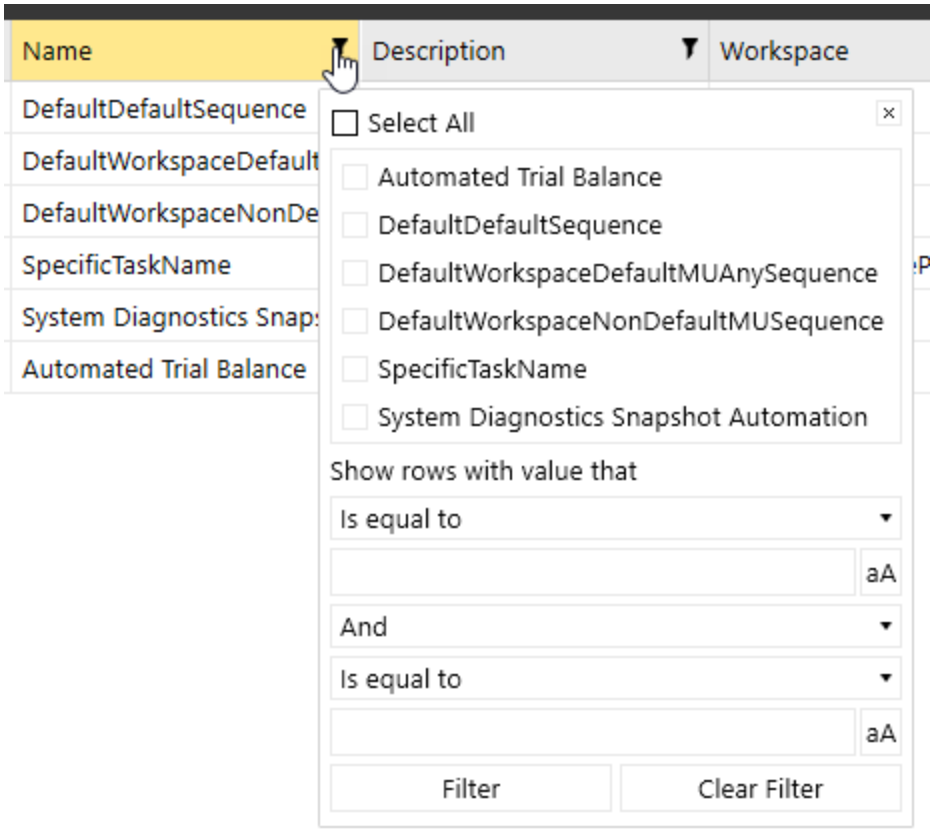


Task Scheduler Details

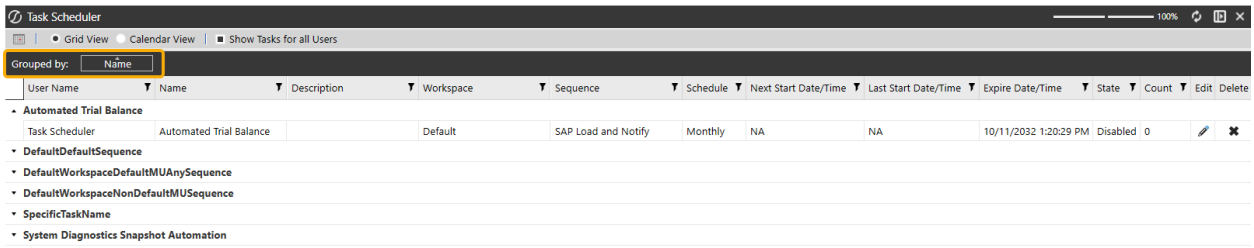
Field	Definition
User Name	The user that created the task.
Name	Name of the task.
Description	Description of the task.
Sequence	Data management sequence that is run by the task scheduler.
Schedule	The time frame implemented for the specific task.
Next Start Time	The next time the task is scheduled to run.
Last Start Time	The last time the task ran.
Expire Date/Time	The time and date the task will expire and no longer run.
State	Enabled or disabled.
Count	The amount of time the task has ran.
Edit	Use the Edit icon to edit.
Delete	Use the Delete icon to delete.
If enabled by an administrator:	

Field	Definition
Invalidate Date/Time	The time and date the task will be suspended and not run until validated.
Validate Task	Validate task to keep active.

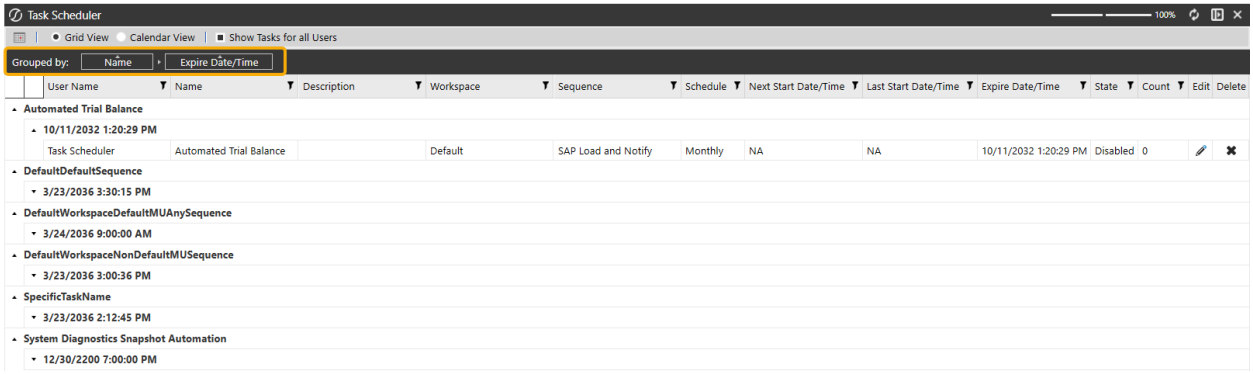
You can filter on any fields in the grid that have the filter icon, and you can filter on multiple selections.



You can group by column name by dragging and dropping the group into the header bar.



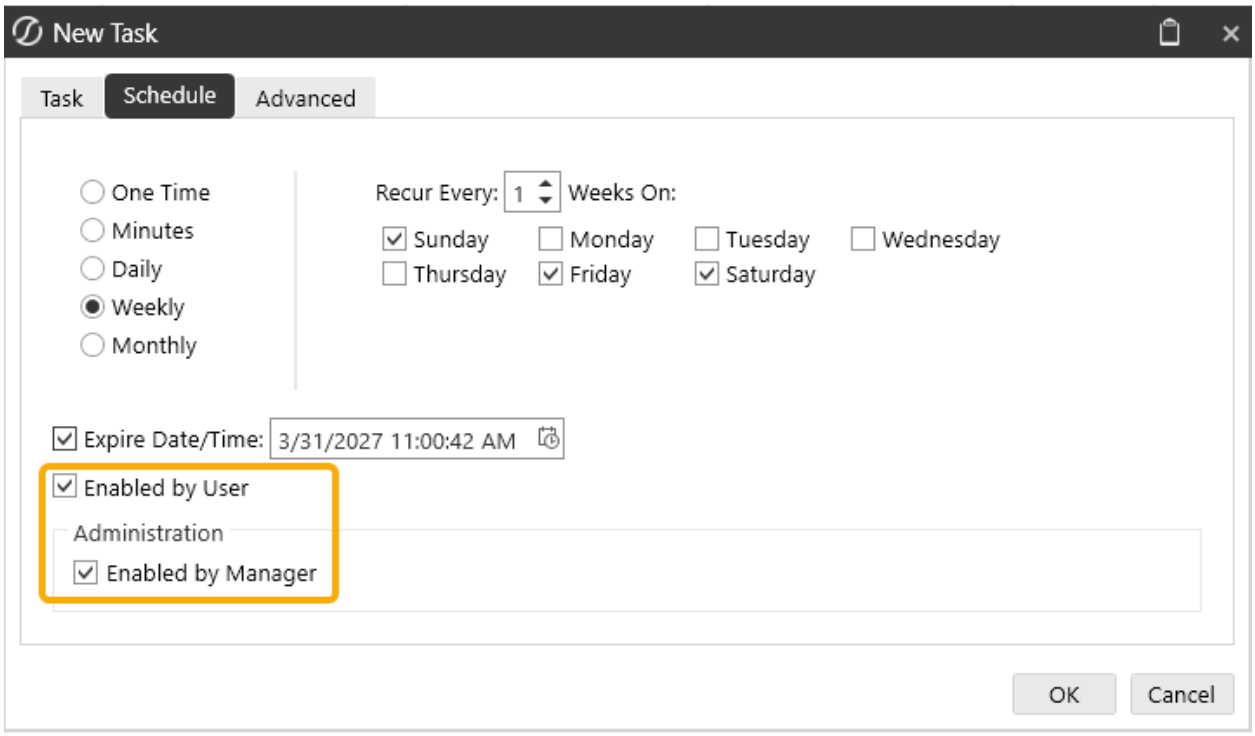
You can group by more than one column. Each row can be expanded to show task details.



Task Manager Security Roles

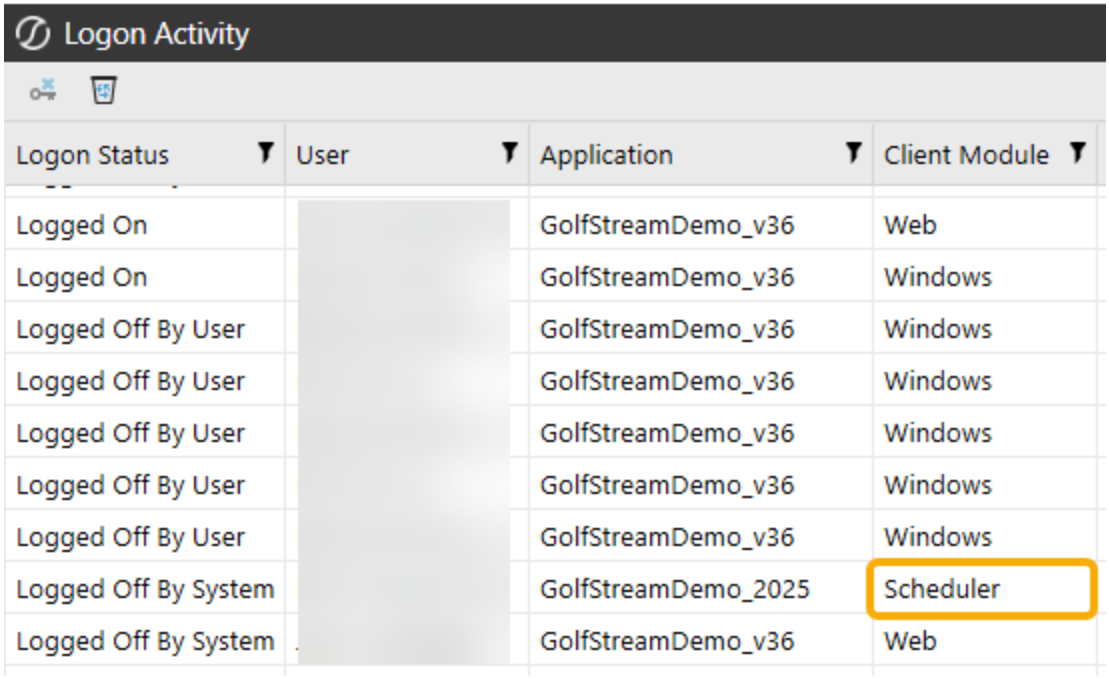
Security Role	Description
TaskSchedulerPage	This role is required to view the Task Scheduler page and view all user tasks. This is found in **Application** > **Tools** > **Security Roles** > **Application User Interface Roles**.

Security Role	Description
Task Scheduler	This role enables you to create new tasks, edit your tasks, and validate tasks if the setting is on. You can view all user tasks, but you can only edit your own tasks. You will not have access to load and extract. You cannot change the task name. This is found in **Application** > **Tools** > **Security Roles** > **Application Security Roles**.
ManageTaskScheduler	This role enables you to create your own tasks, view every task no matter who created it, edit your own tasks and other users' tasks, and delete or disable your own tasks. You can load and extract. You cannot change the task name. This is found in **Application** > **Tools** > **Security Roles** > **Application Security Roles**.



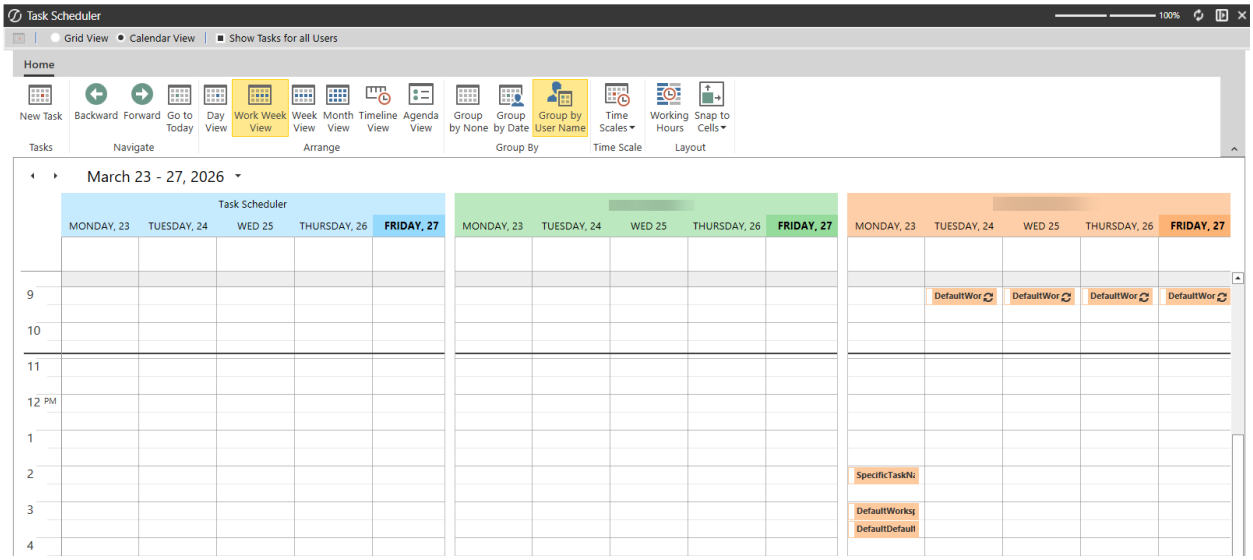
Logon Activity

Go to **System** > **Logon Activity**. In the Client Module column, you can see that the log in was through Scheduler.

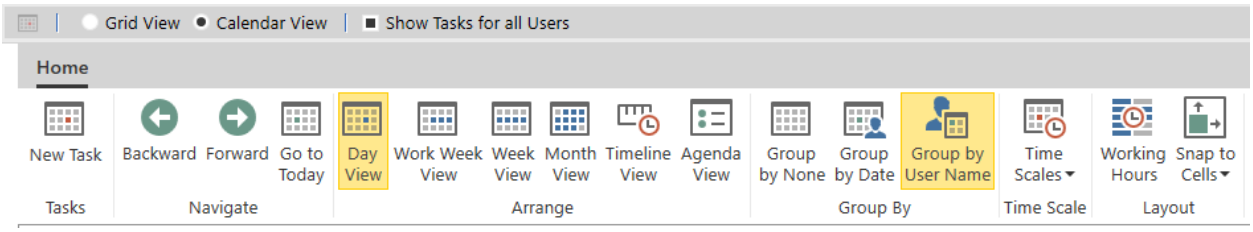


Calendar View

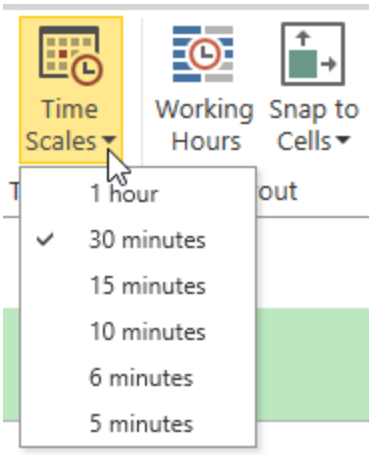
To show tasks for all users in the calendar view, select the **Show Tasks for all Users** checkbox and then select the **Group by User Name** icon. The tasks are color coded when multiple users are displayed.



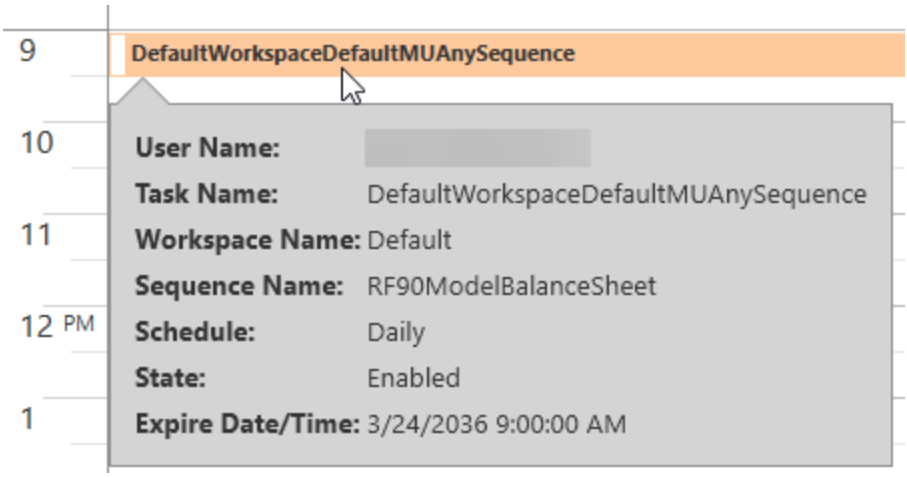
You can switch between the different views, and you can view by the current day, work week, month, timeline, and agenda. You can group by the user name, date, or none.



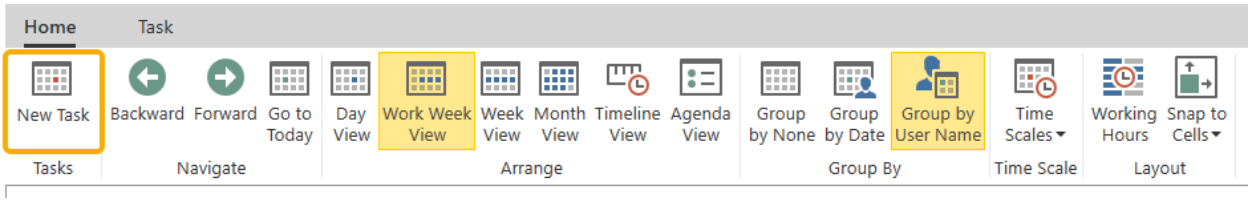
When activated, you can choose to view the calendar by time scales or working hours.



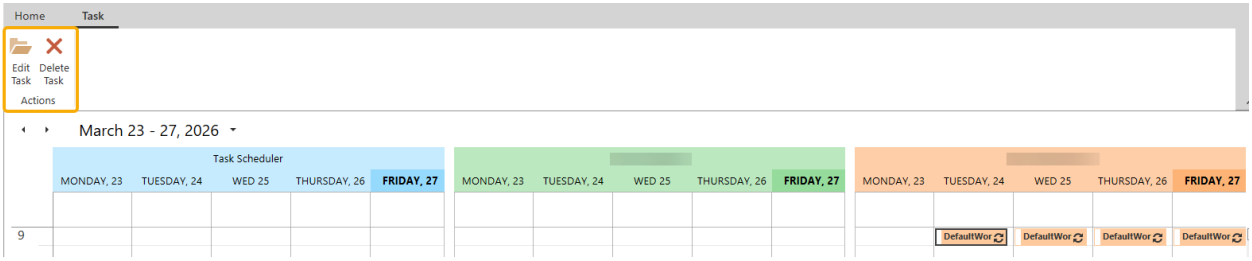
When you hover over a task on the calendar, information specific to the task displays, including the user, task, Workspace, and sequence names, schedule, state, expire date and time, and, if enabled, the invalidate date and time.



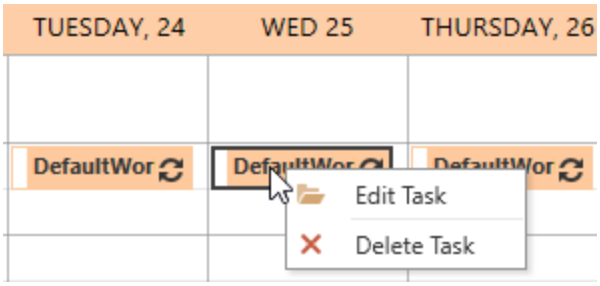
You can create a new task by selecting the New Task icon.



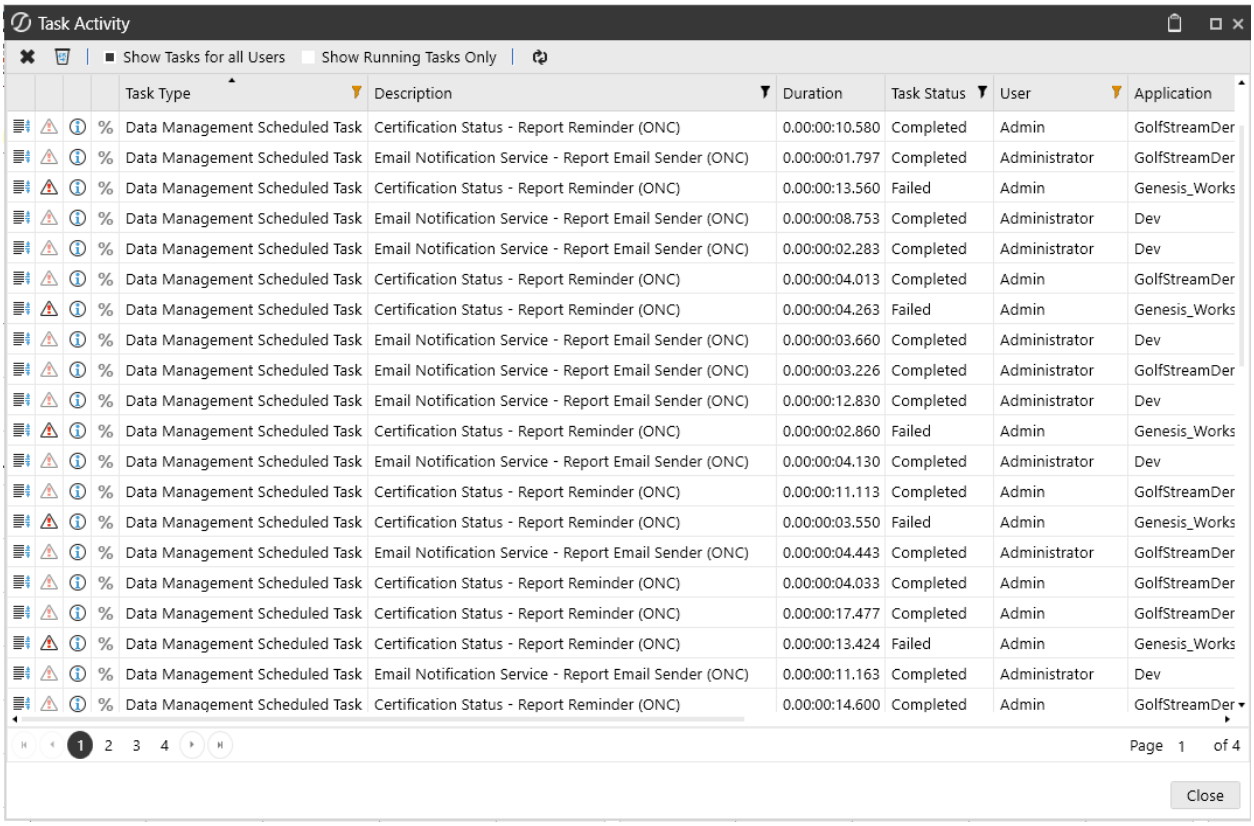
You can edit an existing task by selecting a task in the calendar and then selecting the Edit Task icon. And, you can delete a task from the calendar view by selecting the task in the calendar and then selecting the Delete Task icon.



You can also double-click on the task to edit or right-click to edit or delete the task.



Once the job runs, you can see the status of the job in Task Activity. You can filter the Task Type by Data Management Scheduled Task. The description is the name of the task separated by a hyphen followed by the sequence.



Load and Extract

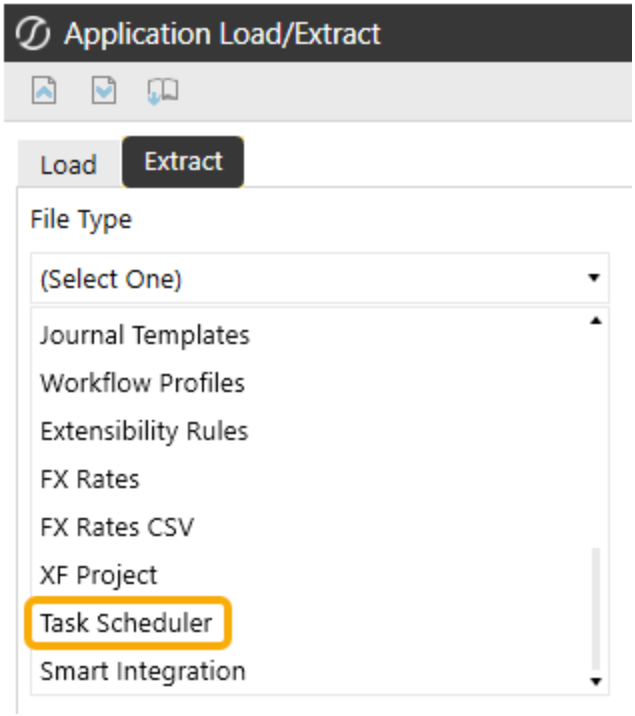
Administrators can load and extract additional task scheduler files.

Load

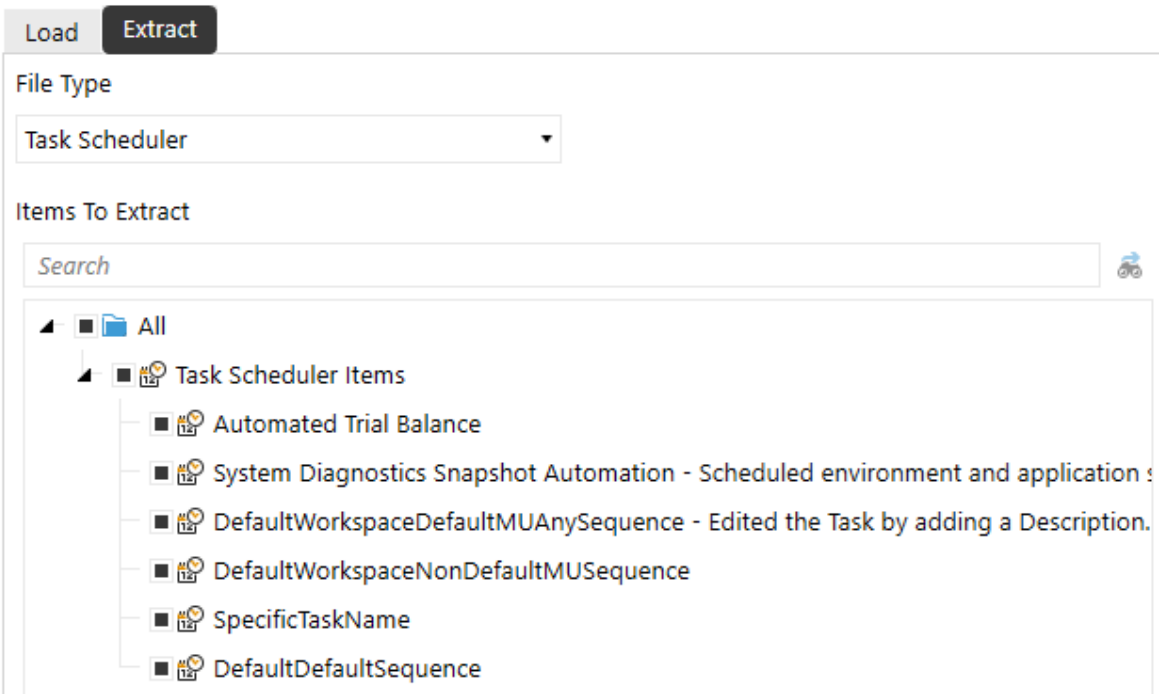
1. Go to **Application** > **Tools** > **Load/Extract**.
2. Select the file to load.
3. Select the **Load** icon.
4. Navigate to Task Scheduler to view.

Extract

1. Go to **Application** > **Tools** > **Load/Extract**.
2. Select **Extract**.
3. From the **File Type** drop-down menu, select **Task Scheduler**.



4. Select the task and select the **Extract** icon.



Text Editor

Text Editor is used to create, edit, and view text documents, like those created in Microsoft Word. This component only works in the OneStream Windows application. Text Editor can perform many of the tasks that can be completed in Microsoft Word, with some limitations. These limitations include, but are not limited to:

- Mail Merge
- Application of pre-defined Styles
- The Shape Fill and Shape Outline tools
- Previous and Next Comment buttons
- Spelling
- Insert Fields
- Add Watermarks

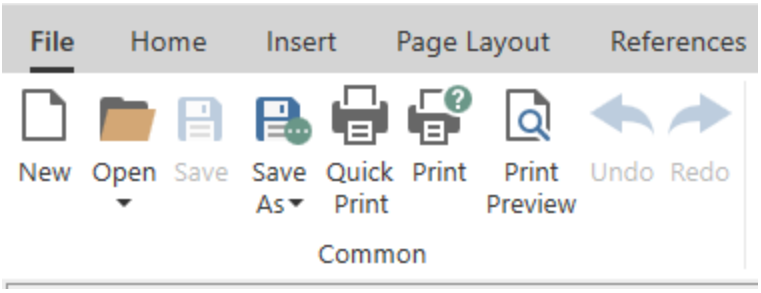
The following shortcut keys function differently in the Text Editor feature than they do in Microsoft Word. Review the following Text Editor shortcut keys :

- **CTRL+N**: Used to create a new blank document. If an existing document exists and has not been saved, using this shortcut closes the existing document without saving and create a new document.
- **SHIFT + F3**: Used to launch the Find command.
- **SHIFT + F4**: Used to change case of selected word.
- **CTRL+SHIFT+Page Up**: Used to move the cursor to the previous page.
- **ALT+I**: Used to increase the letter/number increments within an outline or numbered list.

Text Editor Ribbon

The following section details all options in the Text Editor ribbon.

File



New: This creates a new text editor document. When an open document has unsaved changes, a window displays asking if the current changes should be saved before the new document is created.

Open: The file type to open. You can open a file from:

- **Local Folder**: Select a file to be opened from a location on the local computer/network.
- **OneStream File System**: Select a file to be opened from a location within the File Explorer.

  > **NOTE:** Displayed files can also be opened from here or from OnePlace > Documents by right-clicking and selecting one of the three options: Open in Text Editor Page, Open (opens file in Compatibility mode, directly in Word if the application is found on local computer), or Open With… (user specifies program).

- **Application Workspace File**: Select an Application Workspace file to be opened in Text Editor.
- **System Workspace File**: Select a System Workspace file to be opened in Text Editor.

Save: Save changes to the open file using the current file name. This is only available after the file is given a name.

Save As: The file type to be saved as. You can save a file from:

- **Local Folder**: Select location on local computer/network to save a file to.
- **OneStream File System**: Select location within File Explorer to save a file to.
- **Application Workspace**: Select Application Workspace to save a file to.
- **System Workspace File**: Select System Workspace to save a file to.

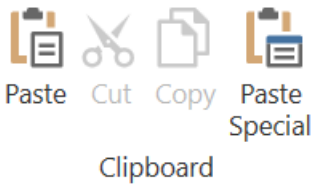
Quick Print: Send the document to the default printer without changing any printer/printing properties.

Print: Displays the Print dialog box for setting options to print the displayed document.

Print Preview: Displays the Print Preview dialog box that shows how the printed document will look. The Print Preview ribbon associated with this button remembers the setting used last that dictates if it is hidden or displayed. The next time Print Preview is opened, the ribbon is initialized the same way it was left.

Home

Clipboard



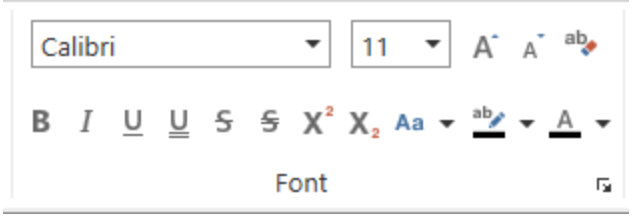
Paste: Inserts copied data into the document.

Cut: Removes and transfers selected data from the document to the clipboard for placement in a different location.

Copy: Copies selected data to the clipboard for inclusion in a different location.

Paste Special: Displays the Paste Special dialog box for additional pasting options.

Font



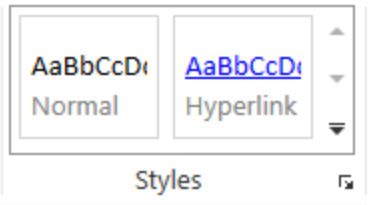
This provides options for changing the text found in the document. Some examples include bold, italic, underline and text color.

Paragraph



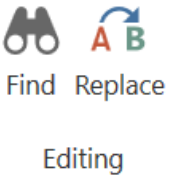
This provides options for formatting text in the document such as bullets, numbering, indenting and paragraph alignment.

Styles



Provides styling for formatting text in the document such as Normal and Hyperlink.

Editing

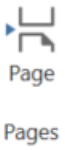


Find: Enter text to be found within the document.

Replace: Enter information to replace located text within the document.

Insert

Pages



Page: This insert a page break at the current location within the document.

Tables

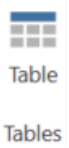


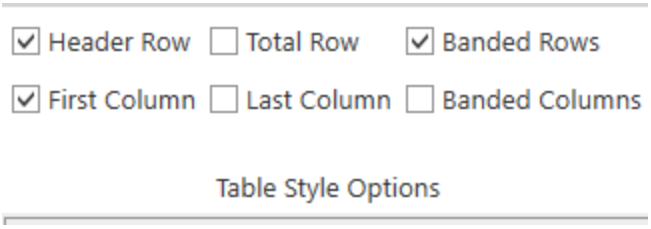
Table: This inserts a table at the current location within the document.

Design

This tab only displays when you are working within a table.

Table Style Options

This toolbar in the Design tab only displays when working within a table.



This provides options for formatting tables in the document such as Header Row, First Column, and Total Row.

Table Styles and Borders & Shading

This toolbar in the Design tab only displays when working within a table.



Table Styles: This provides options for table styles such as format titles, quotes, and other text using the gallery of styles.

Line Style and Weight: This provides options to change the style and width of the line used to draw borders.

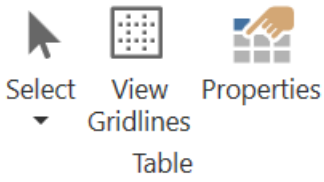
Pen Color: This is used in combination with the Borders button. This button changes the color of the border lines selected when using the Borders button. Select the Borders button first to select the border color, then use the Borders button to define where border lines should be displayed.

Borders: This is used to customize the borders of the selected cells. It is used in combination with the Pen Color button.

Shading: This is used to add color to the background of the selected cells.

Layout

This tab will only display when working within a table.



Select: This is used to select the current cell, column, row or entire table.

View Gridlines: This is used to show or hide the gridlines within a table. When turned on, the gridlines only display where the display of cell borders is turned off.

Properties: This is used to display the Table Properties dialog box. Advanced formatting options such as margins, alignment, text wrapping, borders, and shading can all be managed from here.

Rows & Columns

This toolbar within the Layout tab only displays when working within a table.



Delete: This is used to delete cells, rows, columns or the entire table.

Insert Above: This is used to add a new row directly above the selected cell.

Insert Below: This is used to add a new row directly below the selected cell.

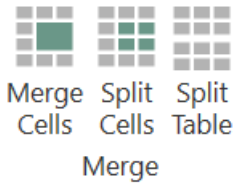
Insert Left: This is used to add a new column directly to the left of the selected cell.

Insert Right: This is used to add a new column directly to the right of the selected cell.

Insert Cells: This is used to insert a single cell into the table. The Insert Cell dialog box displays with options to shift cells right, shift cells down, insert an entire row, or insert an entire column.

Merge

This toolbar in the Layout tab only displays when working within a table.



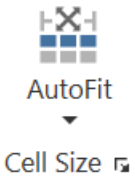
Merge Cells: This is used when two or more cells are selected to merge into one cell.

Split Cells: This is used to split the selected cell into smaller cells. The Split Cells dialog box displays and the number of new columns and rows needed can be entered.

Split Table: This is used to split the table selected into two tables. The row that the selected cell belongs to becomes the first row of the new table created.

Cell Size

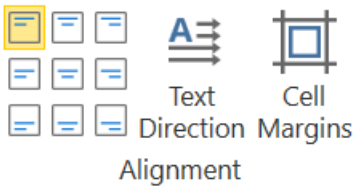
This toolbar in the Layout tab only displays when working within a table.



AutoFit: This is used to automatically resize the width of the column based on the text within. The options include autofitting each cell based on the content within it, autofitting the table to take up the width of the window, and setting the columns to a fixed width.

Alignment

This toolbar in the Layout tab only displays when working within a table.

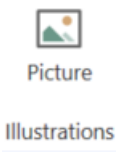


Various options are available to adjust the alignment of text with the table cells. Some examples include align top left, center left, bottom left, top center, center, bottom center, top right, center right, and bottom right.

Text Direction: This change the direction within the selected cells.

Cell Margins: This customizes cell margins and the spacing between cells.

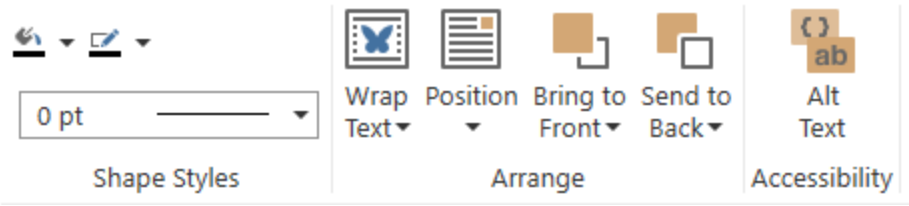
Illustrations



Picture: This inserts a picture file at the current location within the document. Standard picture file options are available to select from.

Format

This toolbar in the Format tab only displays when an inserted picture is selected.



Shape Styles

Shape Styles: This lets you style the picture through shape fill, shape outline, and shape outline weight.

Arrange

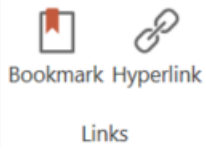
Wrap Text: This changes the way text wraps around the selected object. Seven different wrap options are available including In Line with Text, Square, Tight, Through, Top and Bottom, Behind Text, and In Front of Text.

Position: This positions the selected object on the page. Text is automatically set to wrap around the object.

Bring to Front: This brings the selected object forward so that it is hidden by fewer objects in front of it. Three options are available including Bring Forward, Bring to Front, and Bring in Front of Text.

Send to Back: This sends the selected object backward so that it is hidden by the objects in front of it. Three options are available including Send Backward, Send to Back, and Send Behind Text.

Links

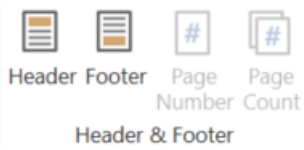


Bookmark: This creates a bookmark for selected text and assigns a name to that specific area of the document. Hyperlinks can be made to move directly to that location.

Hyperlink: This creates a link to a webpage, a file, an application, an email address, or a place in the same document.

> **NOTE:** To send an email enter the following in the address field – mailto:emailusername@domainhostname. For example, mailto:username@onestreamsoftware.com.

Header & Footer



Header: Insert a Header into the document or go to the Header section if a Header already exists within the document.

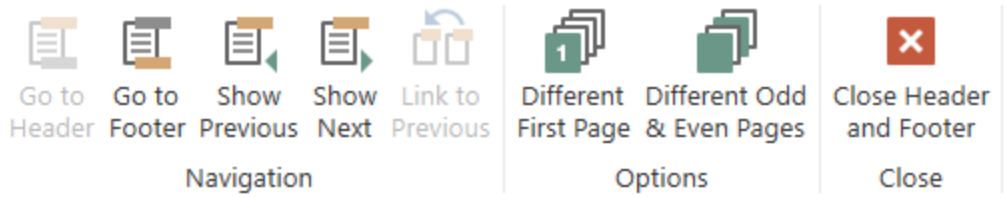
Footer: Insert a Footer into the document or go to the Footer section if a Footer already exists within the document.

Page Number: This inserts the current page number wherever the cursor is located within the Header/Footer areas of the document.

Page Count: This inserts the total number of pages in the document wherever the cursor is located within the Header/Footer areas of the document.

Navigation

This toolbar in the Design tab only displays when working within the Header/Footer areas of the document.



Go to Header/Go to Footer: These activate the Header/Footer section on the page so they can be edited.

Show Previous: If the document has been broken into sections, this navigates to the previous section's Header/Footer.

Show Next: If the document has been broken into sections, this navigates to the next section's Header/Footer.

Link to Previous: This creates a link to the previous section so the Header/Footer in this section contains the same content as the previous section.

Options

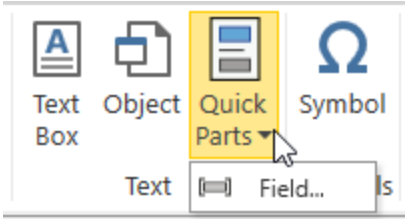
Different First Page: This is used to insert a unique Header/Footer for the first page only of the document.

Different Odd & Even Pages: This is used to insert different Header/Footers on even and odd pages.

Close

Close Header and Footer: This is used to close out of the Header/Footer Tools and return to the document.

Text



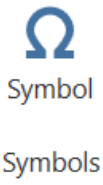
Text Box: This inserts a text box into the document.

Object: This inserts an embedded object, such as another Word document or an Excel chart.

Quick Parts: This inserts document variable syntax for content and formatting strings.

Field: The field to insert document variables and arguments.

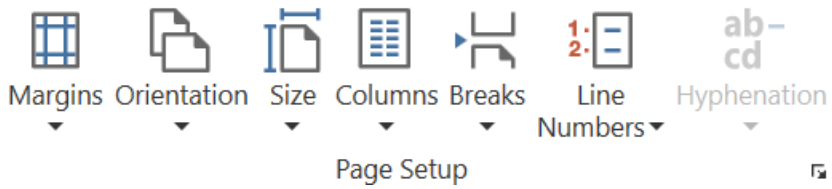
Symbols



Symbol: This inserts standard characters and those not found on the keyboard into the document.

Page Layout

Page Setup



Margins: This is used to set pre-defined or custom margin sizes for the entire document or the current section.

Orientation: This is used to change the pages between landscape and portrait layouts.

Size: This is used to set the paper size for the current section.

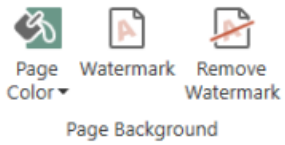
Columns: This is used to split the text into two or more columns.

Breaks: This is used to insert page, column, or section breaks: Next, Continuous, Even, or Odd.

Line Numbers: This is used to insert line numbers in the margins on each line of the document. The options include restarting numbering on each page, each section, suppressing numbers for current paragraph, and custom settings.

Page Setup: This opens the Page Setup dialog box where margins, orientation, paper size, headers/footers, and header/footer placement can all be set/edited.

Page Background



Page Color: This is used to choose a background color for all the pages in the document.

Watermark: Insert and configure text and image watermarks within Text Editor documents. See Text Editor Watermarks.

Remove Watermark: Remove all watermarks from the Text Editor document.

References

Table of Contents

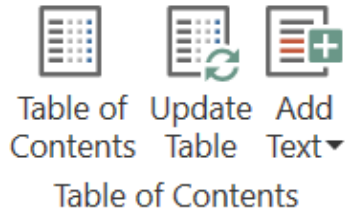
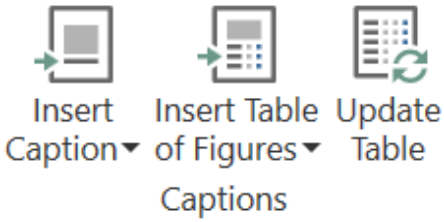


Table of Contents: This is used to create a Table of Contents for the document.

Update Table: This is used to update the Table of Contents so that all entries reference the correct page number.

Add Text: This is used to add the current paragraph into the Table of Contents.

Captions



Insert Caption: This is used to add a caption to a picture or another image. It is used to describe the object associated with it and displays below the object. Captions can be created for the following items:

- **Figures Caption**: This is used to add a caption to a figure or picture within the document.
- **Tables Caption**: This is used to add a caption to a table within the document.
- **Equations Caption**: This option is not used or supported because equations cannot be created in the OneStream Windows application Text Editor.

Insert Table of Figures: Similar to the Table of Contents, this is used to insert a table that lists all the figures or tables that are contained in the document. The following table types can be created:

- **Table of Figures**: This includes a list of all Figures in the document that have captions associated with them.
- **Table of Tables**: This includes a list of all Tables in the document that have captions associated with them.

- **Table of Equations**: This option is not used or supported because equations cannot be created in the OneStream Windows application Text Editor.

Update Table: This is used to update the Table of Figures to include all entries within the document.

Footnotes



Insert Footnote: Add a footnote.

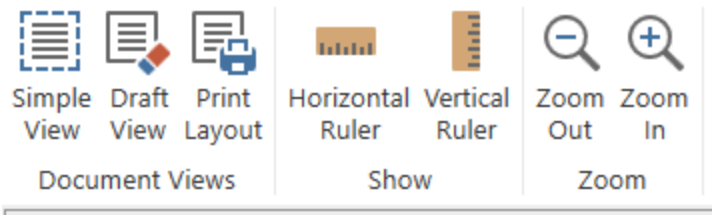
Insert Endnote: Add an endnote.

Next Footnotes: Jump to next footnote.

- **Next Footnote**: Jump to next footnote.
- **Previous Footnote**: Jump to previous footnote.
- **Next Endnote**: Jump to next endnote.
- **Previous Endnote**: Jump to previous endnote.

Show Notes: Jump to the footnote or endnote.

View



Document Views

Simple View: This is used to show the document as a simple memo. Page Layout features are ignored in this view.

Draft View: This is used to view the document as a draft for faster editing. Certain features, such as Headers/Footers, are not visible when using this view.

Print Layout: This is used to show how the document will look on the printed page.

Show

Horizontal Ruler: This is used to display the Horizontal Ruler, which is used to measure and line up objects in the document.

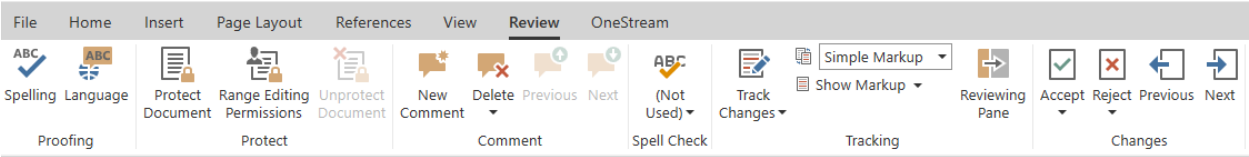
Vertical Ruler: This is used to display the Vertical Ruler, which is used to measure and line up objects in the document.

Zoom

Zoom Out: This is used to change the view to see more of the page at a reduced size.

Zoom In: This is used to change the view to get a close-up view of the document.

Review



Proofing

Spelling: This is used to check the spelling of text in the document.

Language: This is used to set the language used to check the spelling and grammar of the selected text.

Protect

Protect Document: This is used to add a password to the document, so that when protected, only users who know the password can edit the document or add comments. There are options for both of these and only one can be implemented at a time.

Unprotect Document: This is used to unprotect a document that has been locked for editing. Select the Unprotect Document button and provide the previously created password to unprotect the document.

Comment

New Comment: This is used to add comments to the document. This automatically numbers the comments as they are added to the document. The comment located at the top of the document is numbered 1 and additional comments are numbered sequentially down the document based on their position. If a comment is added or deleted, remaining comments in the document are automatically renumbered based on their location within the document.

Delete: This is used to delete comments in the document. The options include the following:

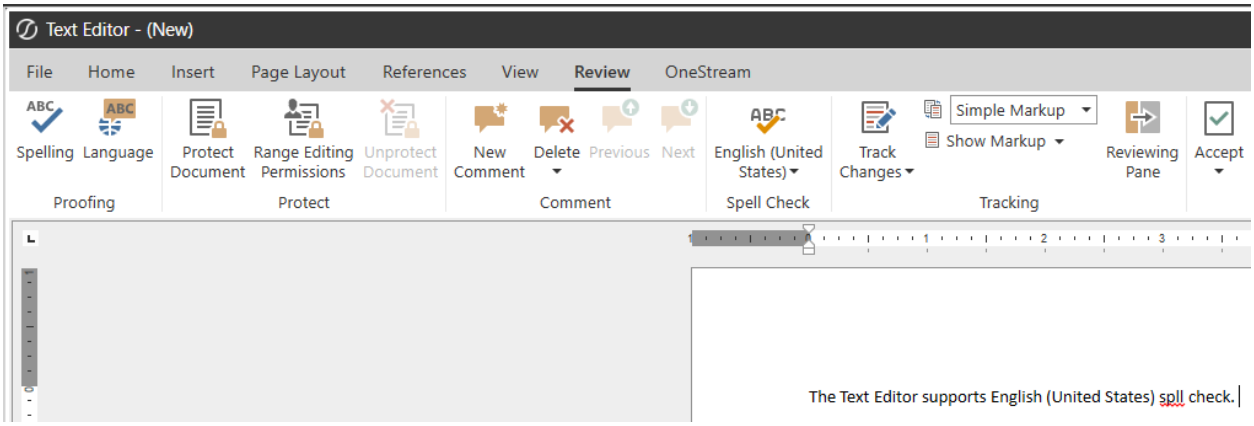
- **Delete Comment**: This is used to delete the selected comment only.
- **Delete All Comments Shown**: This option is not supported in the Text Editor for OneStream Windows application.
- **Delete All Comments**: This is used to delete all comments in the document.

Spell Checking

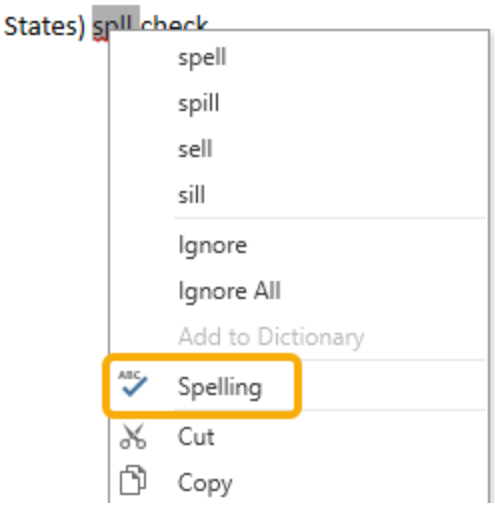
A text spell check feature is available only when using the Windows application. This feature is set to inactive by default. To enable the Spell Check feature, users must have access to **Application** > **Tools** > **Text Editor**. In Text Editor, the Review ribbon enables the user to activate Spell Check using the Spell Check button.

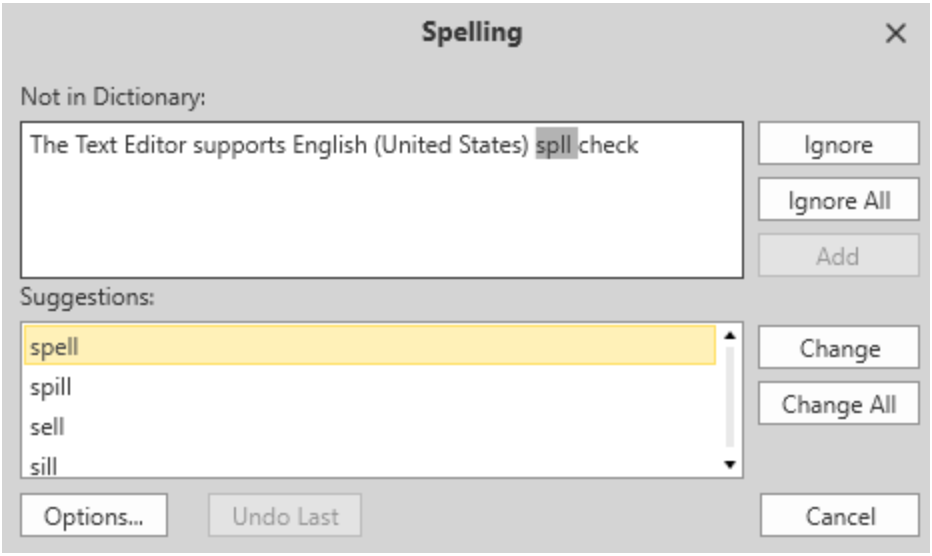
The Spell Check feature is enabled for English Culture only. The culture is determined by each user's culture assigned in OneStream User Security. The culture is assigned to the OneStream application in the Application Server Configuration Utility as en-US. Users with cultures other than English (United States) will not have Spell Check available.

Spell Check in Text Editor: The Spell Check is active as the user enters text, identifying any errors.

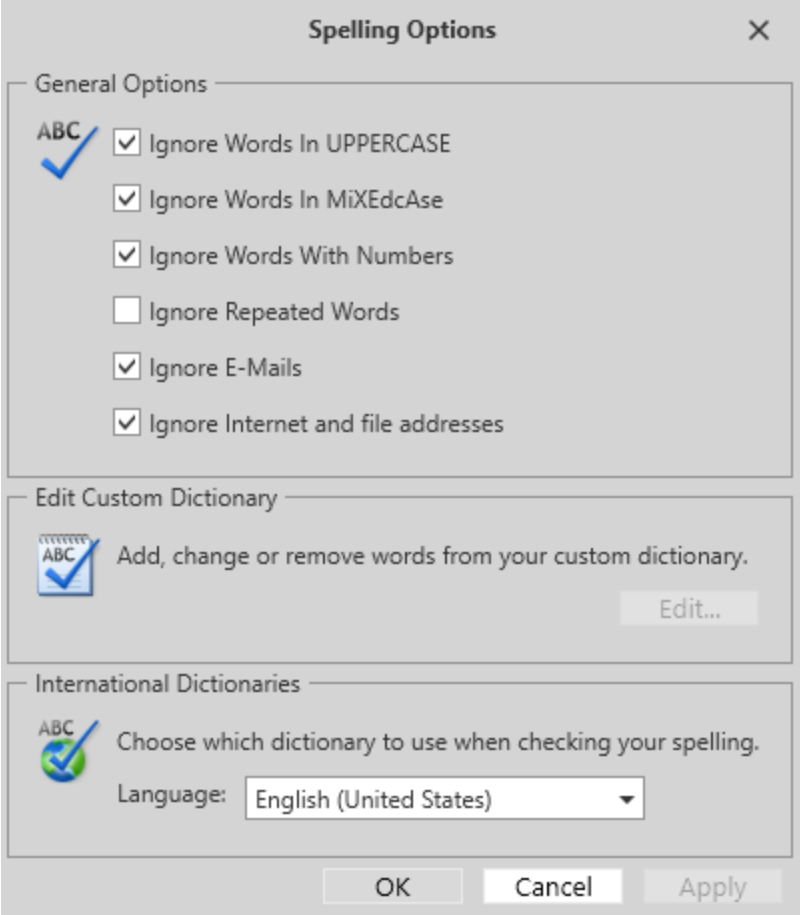


To display Spell Check options, select the error and right-click to display Spell Check menu choices. Choosing the Ignore option will only be retained for the current session. Closing and re-opening to the edit mode will re-check any previously ignored items. Select **Spelling** to display the Spelling dialog box.





The Options button enables the user to modify the Spell Check behavior within the current task session. The settings are not persisted as user preference.



Tracking

Track Changes: Keep track of changes made to this document. Options include the following:

- **Track Changes**: Keep track of changes made to this document.
- **Lock Tracking**: This uses a password to keep others from turning Track Changes on and off.

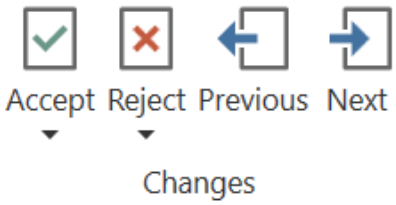
Display for Review: Choose how you would like to see changes in the document. Options include: Simple Markup, All Markup, and No Markup.

Show Markup: Choose what types of markup to show in your document. Options include the following:

- **View Comments**: This is used to show comments that have been added to the document. They are displayed in a column on the right side of the document.
- **Reviewers**: Select which reviewers' comments to highlight in the document.

Reviewing Pane: This is used to show/hide comments within the document in a separate window. The window opens on the left side of the document and can be used to select comments.

Changes



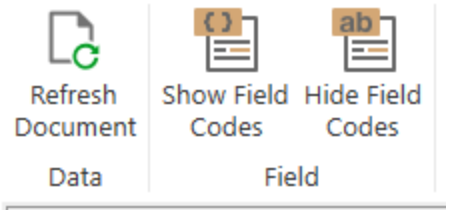
Accept: Keep this change and move on to the next one.

Reject: Undo this change and move on to the next one.

Previous: Jump to the previous tracked change.

Next: Jump to the next tracked change.

OneStream

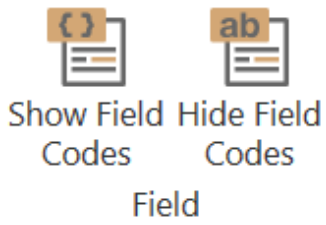


Data

Refresh Document: This enables you to save and refresh your changes after making edits to the document.

> **NOTE:** This is useful for Document Variables as the Refresh Document button will re-process variables.

Field



Show Field Codes: This enables you to show field codes for document variables of the embedded content.

Hide Field Codes: This enables you to hide field codes for document variables of the embedded content.

Text Editor Watermarks

Watermarks help you to protect a company's assets, such as documents, from unauthorized use. In OneStream, you can create watermarks using images or text.

Create an Image Watermark

1. In Text Editor, select the **Page Layout** tab and then, in **Page Background**, select the **Watermark** button. The Watermark dialog box displays.
2. Select the **Picture watermark** radio button.
3. Browse to an image on your local drive and select it.
4. From the **Scale** drop-down menu, select a scale and adjust the image size by percentage if needed.
5. Select the **Washout** checkbox to apply a washout effect to the image. Deselect the checkbox to keep the original colors.
6. From the **Apply to** drop-down menu, select how to apply the watermark:
    - **Whole document**: This applies the watermark to every page in the document.
    - **Section**: This applies the watermark to the section in which the cursor is currently positioned,

    > **NOTE:** Sections are created using the Breaks menu on the Page Layout tab.

7. Select the **Apply** button or select the **OK** button.

Create a Text Watermark

1. In Text Editor, select the **Page Layout** tab and then, in the **Page Background** section, select the **Watermark** button. The Watermark dialog box displays.
2. Select the **Text watermark** radio button.
3. Type the watermark text or, from the **Text** drop-down menu, select a pre-defined option.
4. Apply font, font size, and color options as needed.
5. Select the **Semitransparent** checkbox to apply a semitransparent effect to the text. Deselect the checkbox to keep the original opacity.
6. Select the **Diagonal** or **Horizontal** radio button to determine the layout.
7. In the **Apply to** drop-down menu, select how to apply the watermark:
    - **Whole document**: This applies the watermark to every page in the document.
    - **Section**: This applies the watermark to the section in which the cursor is currently positioned.

    > **NOTE:** Sections are created using the Breaks menu on the Page Layout tab.
8. Select the **Apply** button or select the **OK** button.

Modify a Watermark

You can modify a watermark by selecting the Watermark button. The Watermark dialog box displays and you can make modifications as needed.

Remove a Watermark

1. In Text Editor, select the **Page Layout** tab and then, in the **Page Background** section, select the **Remove Watermark** button. The Watermark dialog box opens. This removes all watermarks from the document.
2. Alternatively, open the Watermark dialog box by selecting the **Watermark** button.
    a. In the Watermark dialog box, select the **No Watermark** radio button.
    b. Select the **Apply** button or select the **OK** button.