API Overview Guide

8.2.0 Release

Copyright © 2024 OneStream Software LLC. All rights reserved.

Any warranty with respect to the software or its functionality will be expressly given in the Subscription License Agreement or Software License and Services Agreement between OneStream and the warrantee. This document does not itself constitute a representation or warranty with respect to the software or any related matter.

OneStream Software, OneStream, Extensible Dimensionality and the OneStream logo are trademarks of OneStream Software LLC in the United States and other countries. Microsoft, Microsoft Azure, Microsoft Office, Windows, Windows Server, Excel, .NET Framework, Internet Information Services, Windows Communication Foundation and SQL Server are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. DevExpress is a registered trademark of Developer Express, Inc. Cisco is a registered trademark of Cisco Systems, Inc. Intel is a trademark of Intel Corporation. AMD64 is a trademark of Advanced Micro Devices, Inc. Other names may be trademarks of their respective owners.

Table of Contents

Introduction 1
Development Technologies 2
Programming Language 2
User Interface Technology 2
Server Technology 2
Database Technology 3
OneStream API Details and Database Documentation 3
Developer Fundamentals 4
VB.Net and C# 4
In-Solution Documentation 4
Business Rules Editor Overview 4
Helpful Resources 5
Platform Engines 7
Workflow Engine 7
Stage Engine 7
Finance Engine 7
Data Quality Engine 8
Data Management Engine 8
Presentation Engine 8

BRApi 9
Business Rules 10
Anatomy of a Business Rule 10
Business Rule Definition 10
Business Rule Classifications 12
Event Handler Business Rules 13
Complex Expressions 16
Business Rule Types 21
Organizing and Referencing Business Rules 30
API Structure and Organization 36
Namespaces 36
Namespaces Defined 37
Namespace Hierarchy 37
Microsoft Financial Calls 39
In-Solution Development 40
Custom Development 41
Using System Tools 42
System Business Rules 42
Database 43
Tables 43

Tools 43
Data Records 43
Event Listing 44
Event Handler Business Rules 44
Event Firing Sequences 47
Finance Functions APIs 79
Member ID 80
Api.Pov.Time.MemberId 80
Api.Pov.Time.MemberId Usage 82
Api.Pov.Entity.MemberId 83
Api.Pov.Entity.MemberId Usage 84
Api.Pov.Account.MemberId 85
Api.Pov.Account.MemberId Usage 86
Dimension Primary Key - DimPk 87
DimPK Usage 87
Dimension Type Id 89
DimTypeID Usage 90
Data Unit Dimension POV 91
Data Unit Dimension POV Usage 91

Time Functions 93
Api.Time.GetYearFromId 93
Api.Time.GetPeriodNumFromId 93
Api.Time.GetPeriodNumFromId Usage 93
Api.Time.GetNumDaysInTimePeriod 94
Api.Time.GetNumDaysInTimePeriod Usage 94
Api.Time.AddTimePeriods 95
Api.Time.AddTimePeriods Usage 95
Api.Time.AddYears 96
Api.Time.AddYears Usage 96
Using Member Functions for Calculations 98
GetMember 98
GetMember Usage 98
GetMemberId 99
GetMemberID Usage 99
GetBaseMembers 100
GetBaseMembers Usage 100
Writing Stored Calculations 102
Overload Function 103
Api.Data.Calculate Usage 103

IsDurableCalculatedData 104
IsCurableCalculatedData Usage 104
Eval Function 104
Eval Function Usage 105
Summary 106
Remove Functions 107
RemoveZeros 107
RemoveNoData 107
Remove Functions Usage 108
GetDataBuffer Functions 110
GetDataBuffer Function 110
GetDataBuffer Usage 111
Unbalanced Math Functions 113
Unbalanced Math Functions 113
Unbalanced Math Functions Usage 114
GetDataBufferUsingFormula Function 114
FilterMembers 114
GetDataBufferUsingFormula Usage 114

Introduction

The purpose of the API Guide is to provide detailed information about the technologies and application programming interfaces available to consultants and developers interested in extending the functionality of OneStream.

This document contains information about the technologies used in the OneStream product, naming conventions and organizational approaches used by the OneStream engineering team. It also includes detailed reference listings for API methods and events exposed by OneStream.

For customers in a OneStream-hosted environment, see the *Identity and Access Management Guide* for information about authentication with OneStream IdentityServer and using personal access tokens (PATs).

Development Technologies

Programming Language

The OneStream platform is based on the Microsoft .Net Framework. OneStream's underlying codebase is predominately made up of C# libraries with a few VB.Net libraries in use as well. C# and Visual Basic .NET are the two primary programming languages used to code against the .NET Framework. C# and VB.NET have very different syntax elements, but Microsoft developed these languages simultaneously as part of a common .NET Framework development platform. Both C# and VB.Net are developed, managed, and supported by the same language development team at Microsoft. They compile to the same intermediate language *(IL)* which runs against the same .NET Framework runtime libraries. Although programming syntax is different for each language, almost every command in VB has an equivalent command in C# and vice versa. Both languages reference the same underlying .NET Framework Base Classes to extend their functionality.

User Interface Technology

The OneStream user interface is based on the Windows Presentation Foundation *(WPF)* in order to provide a truly rich end user experience. WPF employs XAML, an XML based language, to define and link various interface elements. WPF applications can be deployed as standalone desktop programs, or hosted as an embedded object in a website. Windows 10 Store application development provides another opportunity for WPF based applications to be deployed, but as Windows only applications.

Server Technology

All OneStream code is hosted and executed with Microsoft Internet Information Services *(IIS).* This means that both the Web Server *(service code)* and Application Server *(service code)* are executed within an IIS Application Pool process host. The code is running on the application server tier hosted within the application sever IIS application pool. This is a very important concept to keep in mind because there will be times when a Business Rule must interact with different elements of the system. The context in which the Business Rule is running needs to be understood in order to establish communication and/or interact with those other system elements.

Database Technology

OneStream was designed to run on all versions of the Microsoft SQL Server relational database engine *(Express, Standard, Data Center, Enterprise and Azure Database as a Service)*. For larger organizations, the SQL Server Enterprise edition is recommended because OneStream makes use of table partitioning. This enables maximum throughput during heavily multi-threaded operations such as data transformation and consolidation. The OneStream engineering team is committed to fully utilizing the capabilities of the most recent versions of SQL Server and to keeping the OneStream platform optimized for new versions of SQL Server as they become available.

OneStream API Details and Database Documentation

For more information on OneStream API functions and details on the OneStream Framework and Application database tables and indexes, the *OneStream API Details and Database Documentation* is available as part of the documentation. This can be found on MarketPlace under *Software Download*. Create a folder on the PC on which this will be loaded and copy the related zip file:

Right click and extract the zipped file's contents here. Double-click the file which ends in *chm* and this will launch the API Guide.

Contents are organized by the related Platform Engine (see Platform Engines). These are broken down into Classes (e.g. DataApi), Overload Lists, Methods (e.g. GetDataCell), Syntax and Parameters. The Index and Search tabs can be used to search by function name, enumerations, properties, etc.

Developer Fundamentals

VB.Net and C#

The OneStream platform is based entirely on the Microsoft .Net Framework as is the Business Rules engine. Therefore, VB.Net and C# are the logical choice for Business Rule syntax. At execution time, all Business Rules are compiled on demand and cached for fast and reliable execution. Writing a Business Rule in VB.Net or C# provides the end user with many advantages over older products based on VBScript. Business Rule writers can expect exceptional code performance, better error messaging, and better error handling because VB.Net and C# are a full featured programming language. In the end, these capabilities result in a more reliable Business Rule code.

> **NOTE:** There are two broad Business Rule Classifications: Shared Business Rules and Item Specific Business Rules. Shared Business Rules can be written in either VB.NET or C#, Item Specific Business Rules can be written in VB.NET only.

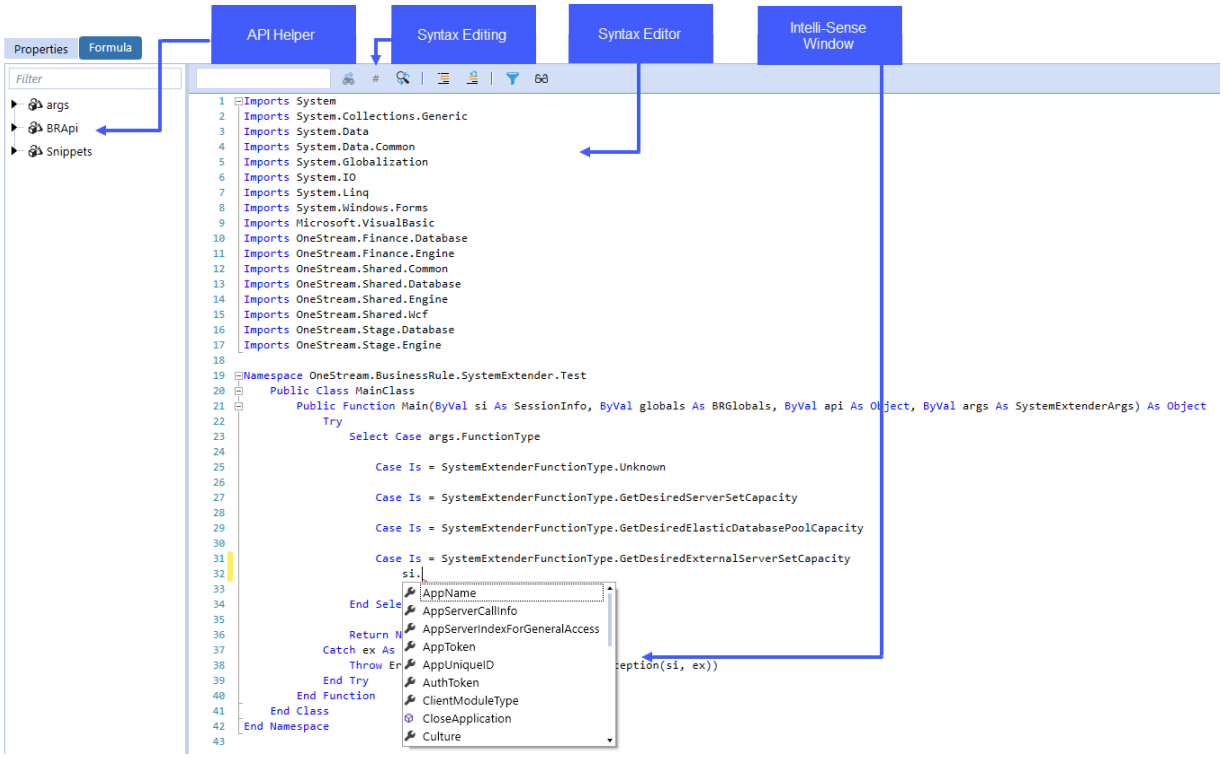
In-Solution Documentation

The Business Rule Editor includes context sensitive help for API properties and methods as well as Snippets *(code examples).* In-solution documentation makes the process of writing a Business Rule more efficient because both API Documentation, Objects, and Samples are presented within the Business Rule Editor window. In addition, useful coding examples accumulated by the OneStream engineering and consulting teams are also presented in context sensitive manner within the Business Rule editor. Companies and partners can author their own Snippets and include them in their application as an extension of the OneStream predefined Snippets (*Snippet Editor MarketPlace Solution required*).

Business Rules Editor Overview

The Business Rule editor is a powerful in-solution screen that provides integrated API context help, syntax editing with intelli-sense, and full outlining capabilities. The actual syntax content and Business Rule structure will be discussed at length in subsequent sections of this document.

The image below explains the major regions and elements of the Business Rule editor.



Helpful Resources

VB.Net

VB.Net is one of the most popular programming languages in use today. This language is especially popular amongst business users because the syntax is perceived to be more readable and business user friendly than other programming languages. VB.Net still shares many of the same syntax elements of older VB dialects such as VB6, VBA and VBScript. This means that users who have written Macros in Microsoft Excel or used VBScript to write Business Rules in first generation CPM solutions should feel comfortable with the core syntax elements of VB.Net. The main learning challenge business users face when migrating to VB.Net is understanding the object oriented nature of the language. In comparison to VBScript, VB.Net offers more elegant coding opportunities. Many of the statements and processes are manually created in VBScript, but in VB.Net they are encapsulated in object libraries on which users can simply call.

Microsoft VB.Net Learning

Getting comfortable with VB.Net takes a little awareness of the basic libraries and objects provided by the Microsoft .Net Framework. The link below points to some resources that business users may find helpful during the VB.Net learning process.

Microsoft Visual Basic

https://msdn.microsoft.com/en-us/library/2x7h1hfk.aspx

C#

C# (pronounced "See Sharp") is a modern, object-oriented, and type-safe programming language. This language is especially popular amongst developers as it enabled them to build many types of secure and robust applications that run in .NET. C# has its roots in the C family of languages and will be immediately familiar to C, C++, Java, and JavaScript programmers.

Microsoft C# Learning

The link below points to some resources that business users may find helpful during the C# learning process.

https://docs.microsoft.com/en-us/dotnet/csharp/

Platform Engines

The platform is comprised of multiple processing engines. These engines have distinct responsibilities with respect to system processing and consequently they 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 engine's core responsibilities.

Workflow Engine

The Workflow Engine is thought of as the controlling engine or the puppeteer. The main responsibility of this engine is to control and track 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 in order to check Workflow status during process execution. The Workflow Engine provides a very rich event model allowing each Workflow process to be evaluated and reinforced with customer specific business logic if required (*see Appendix 2: Event Listing*).

Stage Engine

The Stage Engine performs the task of sourcing and transforming 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 provides the opportunity to interact with source data as it is being parsed and transformed. In addition to parsing and transforming data, the Stage Engine also has a sophisticated calculation that enables data to be derived and evaluated based on incoming source data. The Stage Engine provides quality services to source data by validating, mapping, and executing 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 provides the opportunity to define sophisticated financial calculations through centralized Business Rules as well as 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 execute Confirmation Rules which are used to validate analytic data values.

Data Quality Engine

The Data Quality Engine is responsible for controlling data confirmation and certification processes. This Confirmation Engine is used to define and control the sequence of data value checks required to assert the information submitted from a source system is correct. The Certification Engine is responsible for managing user certifications and determining the Workflow dependents' completion status. This engine is primarily accessed through the BRApi and may be called from other engines in order to check data quality status during process execution.

Data Management Engine

The Data Management Engine provides task automation services to the platform. This engine executes batches of commands that are organized into sequences which contain steps. Steps represent entry points or mechanisms to execute features of other engines. For example, the Clear Data Step uses the services of the Finance Engine. In addition, the Data Management Engine has the ability to execute a Business Rule Step which executes 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 extensive data visualization services to platform. The Presentation Engine is made up of the following component engines: Cube View Engine, Dashboard Engine, Parameter Engine, Book Engine and Extensible Document Engine. The Presentation Engine is responsible for managing and delivering content to the end user as well as providing a development environment for custom user interface elements. This engine enables OneStream MarketPlace 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 product.

BRApi

The BRApi is common across all Business Rules, engines and APIs being run, so it is not an engine itself. A BRApi function runs outside of the other engines and can orchestrate certain functions from within other engines. In other words, a BRApi function be run from one engine (e.g. Parser) to tell other engines (e.g. Finance) to execute their own APIs (e.g. API.Data.GetDataCellUsingMemberScript). For another example, while the API.Data.GetDataCell function is available from within the Finance engine, a similar BRApi called GetDataCellUsingMemberScript can be run from any engine if given the appropriate arguments. A common use is BRApi.ErrorLog.LogMessage from any engine.

Business Rules

Anatomy of a Business Rule

This section provides a detailed explanation of the following:

- 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 Definition

A Business Rule is a class, meaning each business rule is an independent object encapsulating code written in either VB.Net or C#. 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 classes, methods and properties.

Each OneStream Business Rule has a predefined Namespace, a Public Class and a Public Function that the OneStream platform engines invoke when the Business Rule needs to be called.

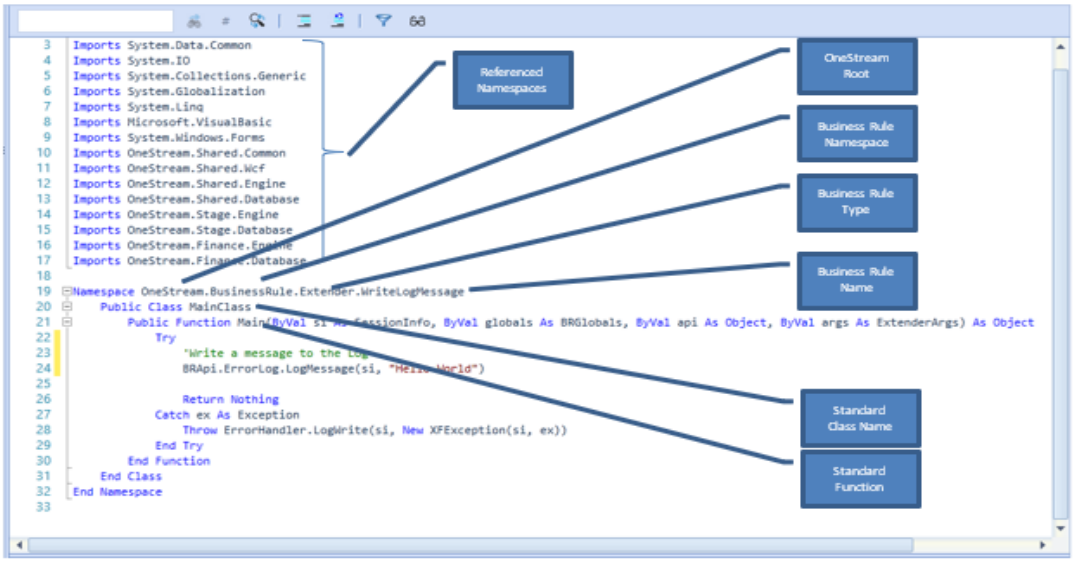
> **NOTE:** There are two broad Business Rule Classifications: Shared Business Rules and Item Specific Business Rules. Shared Business Rules can be written in either VB.NET or C#, Item Specific Business Rules can be written in VB.NET only. All code examples presented in this guide will be shown in VB.NET.

Predefined Object Names

- Namespace: OneStream.BusinessRule.<Business Rule Type>.<Unique Business Rule Name>

- Class: MainClass;
- Function: Main

Example Business Rule Structure



Function Prototypes

Each Business Rule has one standard entry point Function Title called Main. The Function definition below represents the standard prototype used by the Main Function in each OneStream Business Rule. The Main Function always has the same standard parameter layout, but the last two parameters, API and ARGS, contain different object references based on the type of Business Rule being executed.

Public Function Main

(

ByVal si As SessionInfo, Connection Object Required to use API

ByVal globals As BRGlobals, Global Variable Object Used to Share Values

ByVal api As Object, Specific API object (Different for each Type)

ByVal args As ExtenderArgs Specific Arguments (Different for each Type)

)

As Object

Business Rule Classifications

OneStream provides classifications for business logic organization. At the core, all business logic is delivered and executed as compiled VB.Net or C# code. This means no matter what type of business logic is used, there is a consistency in the syntax and compilation process. The reason for different classifications has to do with when and how the business logic is invoked and how the business rule is scoped.

There are two broad business rule classifications: shared business rules and item specific business rules. Each engine in the system may support one or both business rule classifications. Whenever a processing sequence is executed in the platform, the particular engine(s) involved evaluates how and what business logic is associated with the process. This may include shared business rules (named and event handlers) as well as item specific business rules (member formulas, logical expressions, and confirmation rules).

> **NOTE:** Shared business rules can be written in either VB.NET or C#, item specific business rules can be written in VB.NET only.

Finance Engine Example

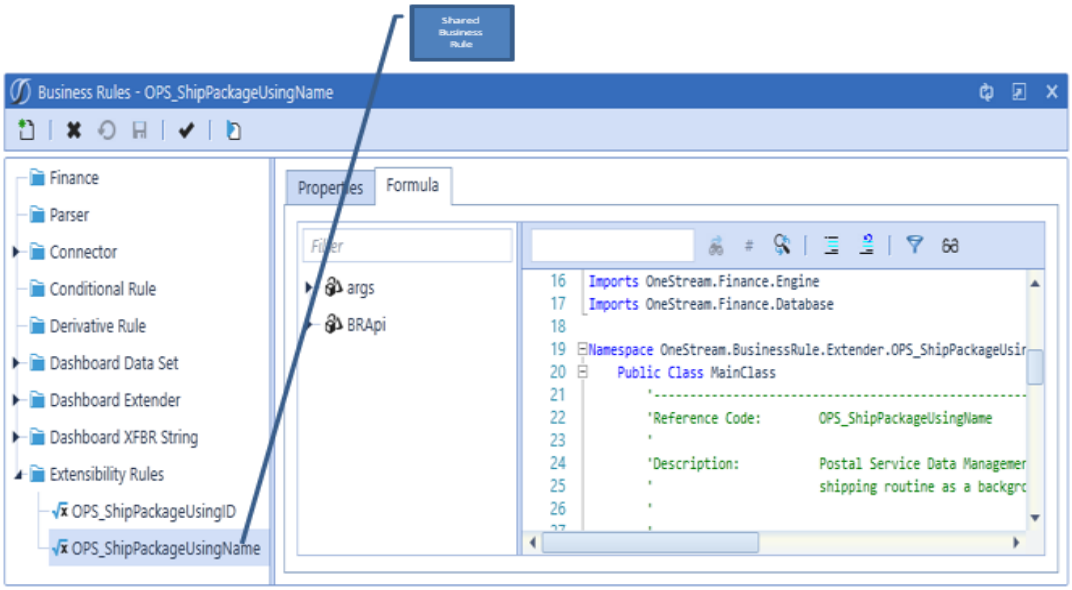
During a consolidation process, a Named Business Rule is associated with the Cube being processed. The Cube contains Member Formulas associated with some of its Dimensions. In this case, the Finance Engine compiles both the Named Business Rule and each individual Member Formula in preparation for the calculation sequence.

Stage Engine Example

A similar example applies to the Stage Engine. During a parse and transform Workflow process, a Named Business Rule is associated with the Data Source or Transformation Rules. In addition, individual Data Source Dimensions or Transformation Rules have associated Logical Expressions that are also fired. In this case, the Stage Engine compiles both Named Business Rules and each individual Logical Expression in preparation for execution during the parse and transform execution sequence.

Shared Business Rules

Shared Business Rules are reusable because the rule is written and stored centrally in the Business Rule Library. This means the same rule can be called or referenced by multiple platform components. For example, the Business Rule highlighted in the image below is a general Extensibility Rule. This rule can be executed from the Business Rule Editor, called by a Data Management Job or called by another Business Rule. Shared Business Rules are the code files seen in the tree when the OneStream Syntax Editor is open, they are organized by type, (*see Business Rule Types in Chapter 4: Business Rules*) and named by the user who created the rule.



Event Handler Business Rules

Event Handler Business Rules are a predefined set of Shared Business Rules and are always defined as an Extensibility Rule Type. Event Handler Rules are invoked during a processing sequence by their related platform engine in order to supplement the process. Determine/filter how/if the execution behaves for specific Workflows or the Cube POV. When an Event Handler Business Rule is called, the calling engine supplies information about the executed process providing context about the process and information about the specific sub-event executed.

Predefined Event Handler Business Rules

The list below details the specific predefined Event Handlers available in the platform. For details on the individual sub-events that fire for each Event Handler Business Rule, see *Event Listing*.

- Data Management Event Handler
- Data Quality Event Handler
- Forms Event Handler
- Journal Event Handler
- Save Data Event Handler
- Transformation Event Handler
- Workflow Event Handler
- Wcf Event Handler

Item Specific Business Rules

Item Specific Business Rules are complete rules like Shared Business Rules, however they are authored and stored with the specific platform item with which the rule is associated. There are different reasons for using Item Specific Business Rules vs Shared Business Rules.

For example, when creating a one-off rule without any reusable value to other components in the system, write an Item Specific Business Rule directly on the platform component because it requires a very specific piece of business logic. Another example, which is more common when creating calculation logic for an analytic model, is to write a Member Formula that directly associates a calculation with a Dimension Member. This creates system maintenance clarity and maintainability.

Item Specific Rules, in particular Member Formulas, can have a positive performance impact because they allow calculations to be broken down into formula passes and processed in a parallel *(multi-threaded)* fashion. The same formulas can be written in a Shared Finance Business Rule, but the calculations will always execute in the serial manner defined in the rule.

Item Specific vs Shared Code Structure

As mentioned above, an Item Specific Business Rule and a Shared Business Rule are identical in code structure. When writing an Item Specific Business Rule, the code editor presents some hidden sections in the code window:

- Formula Header
- Formula Footer
- Helper Function Header
- Helper Function Footer

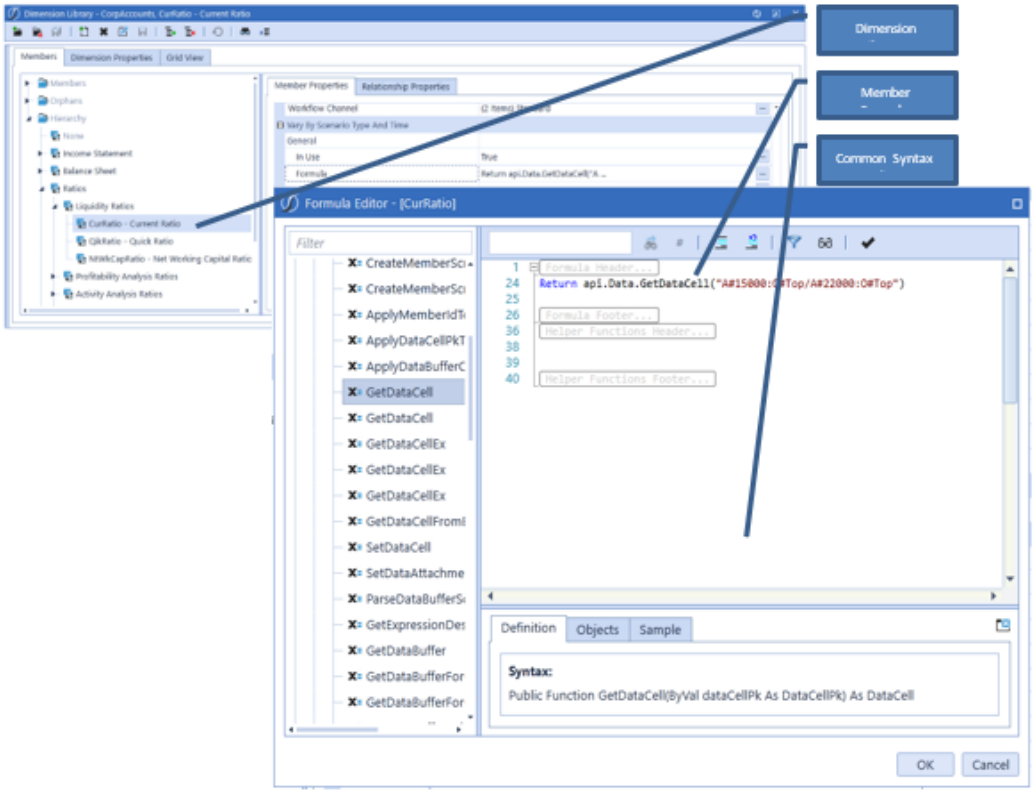
These hidden sections *(i.e. Regions)* keep the formula / expression as readable as possible. In a Shared Business Rule, these sections are visible which make the rule more verbose. The idea behind the Item Specific Business Rule is to create discrete code blocks that are easy to manage and have limited interdependencies. If one knows how to write a Shared Business Rule, then she/he also knows how to write an Item Specific Business Rule and vice versa.

Item Specific Rules are categorized into three types: Member Formulas, Complex Expressions, and Confirmation Rues. These relate to the platform engine with which they are associated.

Member Formulas

A Member Formula is assigned to a Dimension Member and executes within the Finance Engine during a Cube processing sequence (*see the Formula Design Guide in the OneStream Design and Reference Guide for more information on processing sequences*). Member Formulas provide the same level of syntax and logic capability that exist when writing a Finance Shared Business Rule, however custom consolidation, elimination, and translation logic cannot be written. Member Formulas are a great choice for writing logic limited to calculations based on a single Member and calculations that do not span Dimensions. If Member Formulas are written with these constraints in mind, then the Dimension Member and its formula can be reused in different Cubes without having dependencies on other Dimensions. This does not mean that a Member Formula cannot look at other Dimensions. Referencing Dimension Members outside of the specific Dimension where the formula exists will limit the reusability of the Dimension, or require all referenced Dimensions be used together in any new Cube.

Member Formulas are written directly on a Dimension Member within the Dimension Library. Navigate to the specific Member's *Formula* property and click the ellipsis in order to store a Member Formula. The example below is a simple working capital Member Formula.

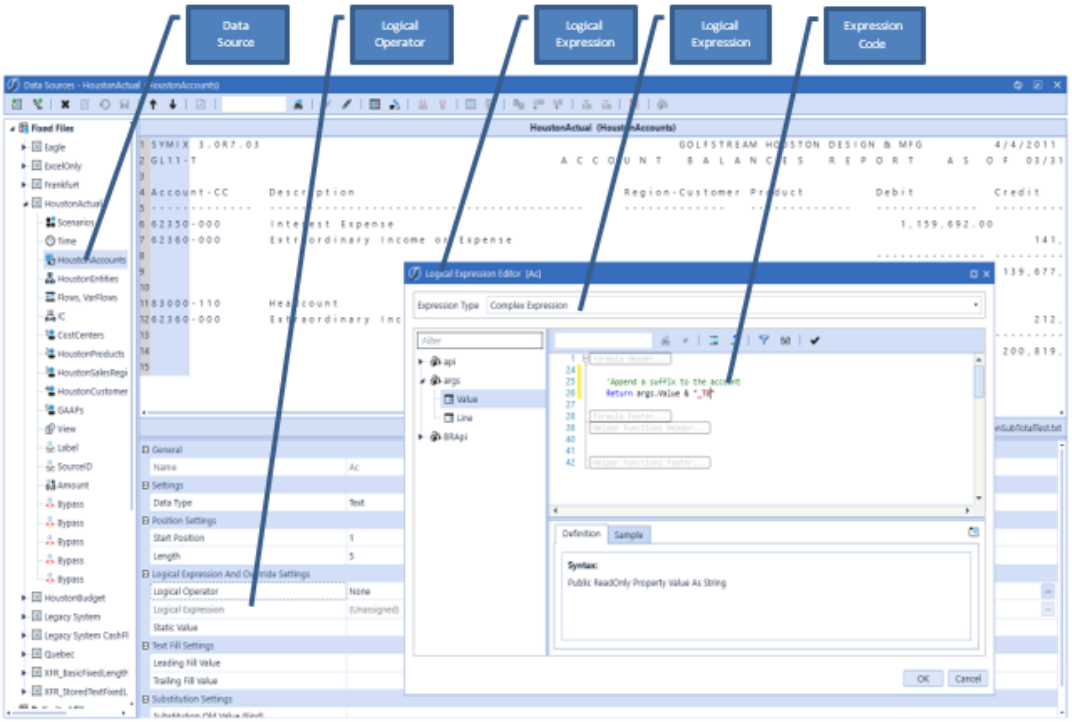


Complex Expressions

A Complex Expression is a Business Rule assigned to Data Source Dimensions, Derivative Rules, and Transformation Rules and execute within the Stage Engine during a transformation processing sequence. Complex Expressions provide the same level of syntax and logic capability that exist when writing a Stage Shared Business Rule. The primary reason for using a Complex Expression rather than a Stage Shared Business Rule is the logic being written has no reusability. Complex Expressions isolate the logic by associating it directly with a specific item.

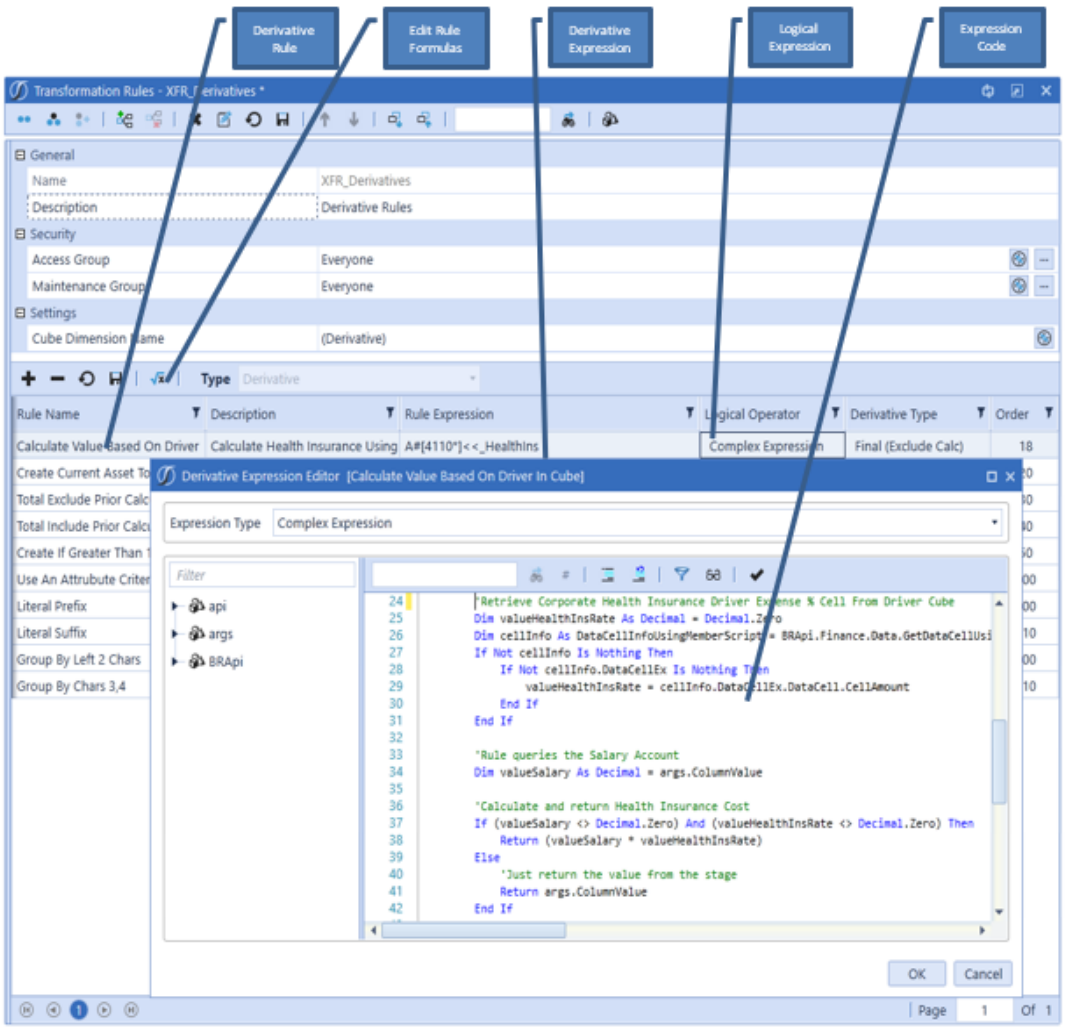
Using Complex Expressions in a Data Source

Apply Complex Expressions to a Data Source Dimension by selecting the Dimension requiring custom logic and setting the *Logical Operator*. The *Logical Operator* property opens the Logical Expression Editor dialog and allows the user to either select a *Shared Parser Business Rule* or write a *Complex Expression*. Both Shared Parser Business Rules and Parser Complex Expressions result in the exact same compiled Business Rule code. The exception is a Complex Expression is only executed for the Dimension to which it is applied and a Shared Parser Rule is shared and can be called by many Dimensions.



Using Complex Expressions in a Derivative Rule

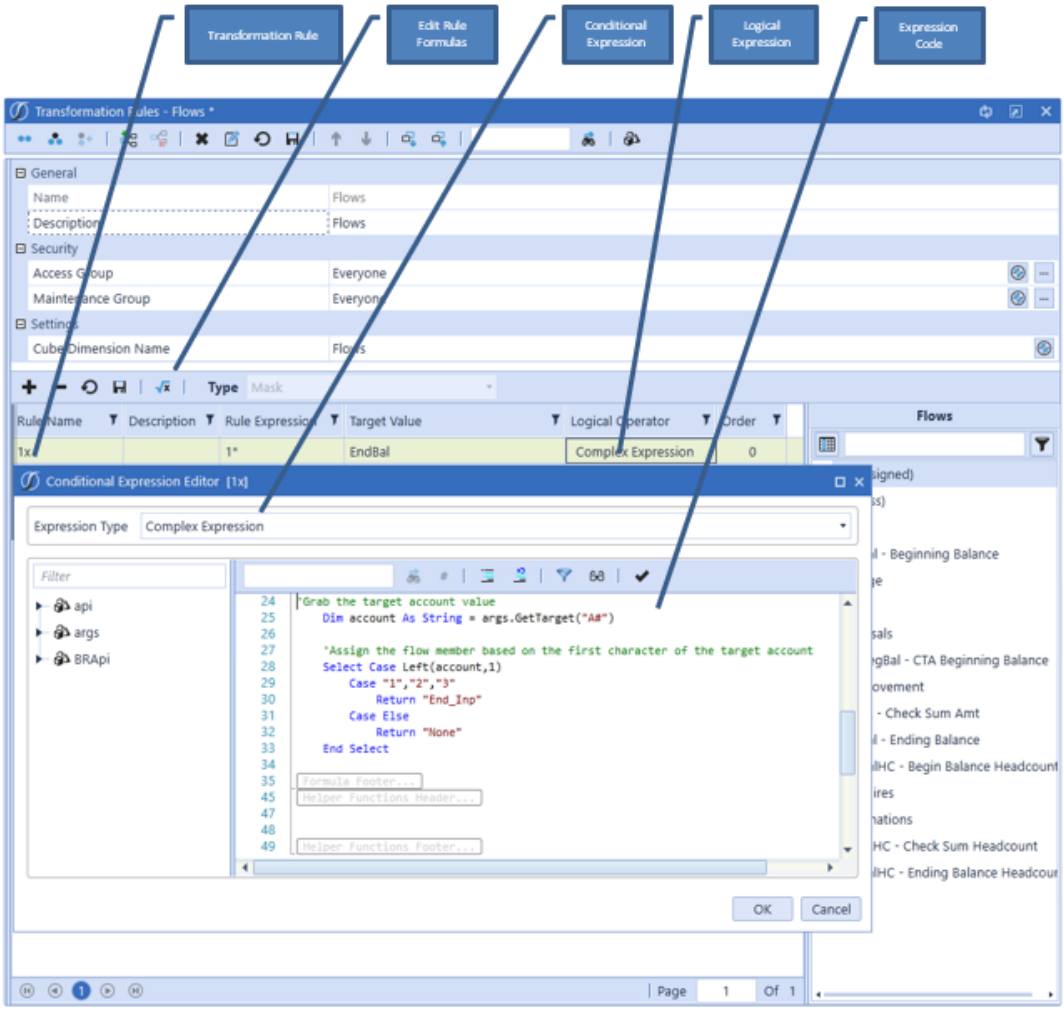
Apply Complex Expressions to a Derivative Rule by selecting the individual Derivative Rule requiring custom logic and setting the *Logical Operator*. Clicking the *Edit Rule Formulas* toolbar button opens the Logical Expression Editor dialog and allows the user to either select a *Shared Derivative Business Rule*, write a *Complex Expression,* or use a *Pre-Built Expression*. Both Shared Derivative Business Rules and Derivative Complex Expressions result in the exact same compiled Business Rule code. The exception is a Complex Expression is only executed for the rule to which it is applied and a Shared Derivative Rule is shared and can be called by many rules.



Using Complex Expressions in a Conditional Transformation Rule

Apply Complex Expressions to a Transformation Rule by selecting the individual Transformation Rule requiring conditional logic and setting the *Logical Operator*. Clicking the *Edit Rule Formulas* toolbar button opens the Logical Expression Editor dialog and allows the user to either select a *Shared Conditional Business Rule* or write a *Complex Expression*. Both Shared Conditional Business Rules and Conditional Complex Expressions result in the exact same compiled Business Rule code. The exception is a Complex Expression is only executed for the rule to which it is applied and a Shared Conditional Rule is shared and can be called by many rules.

> **NOTE:** Shared Conditional Business Rules and Complex Expressions cannot be applied to One-To-One Transformation Rule Types. One-To-One Transformation Rules are executed during the parsing process and therefore are completely processed prior to the conditional mapping process.

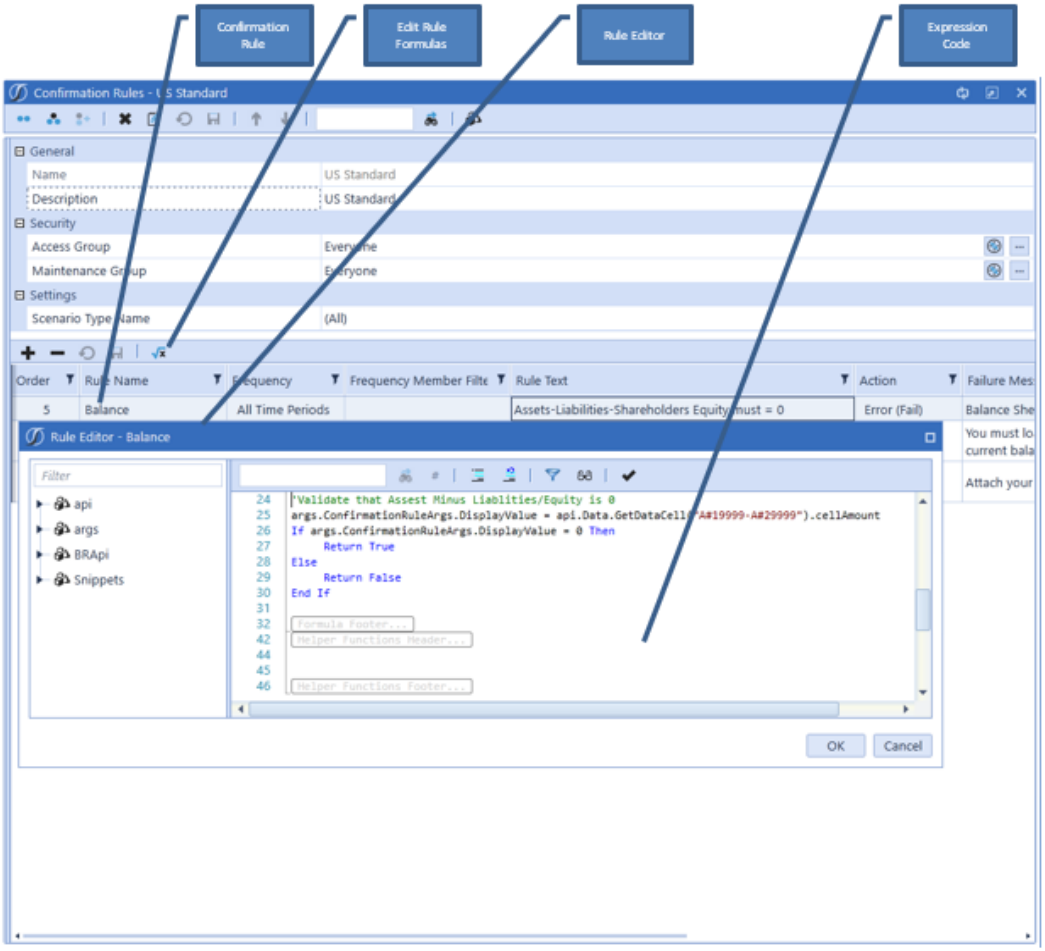


Confirmation Rules

Confirmation Rules are called by the Data Quality Engine and Finance Engine. Apply Complex Expressions to Confirmation Rules by selecting the individual Confirmation Rule and clicking the

Edit Rule Formulas toolbar button. This button opens the Rule Editor dialog and allows the user to write a Complex Expression containing the Confirmation Rule logic. A Confirmation Rule is only written on the specific rule to which it applies. Confirmation rules do not have an equivalent Shared Business Rule because each Confirmation Rule requires specific logic.

> **TIP:** Shared Finance Business Rules can be called from a Confirmation Rule. Create standard helper functions in a Shared Finance Business Rule and call them from a specific Confirmation Rule creating some reusable logic and improving the overall Confirmation Rule infrastructure maintenance (*see Business Rule Organization and Referencing in Business Rules*).



Business Rule Types

Finance

Finance Business Rules are used to generate multi-Dimensional calculations. These Business Rules are written as Shared Business Rules and applied to a Cube or Member Formulas.

Invoking Engine
Finance

API Object Type
FinanceAPI

Args Object Type
FinanceRulesApi

These contain multiple child objects that are populated based on how the rule type is called.

- FinanceRulesApi.MemberListHeadersArgs
- FinanceRulesApi.MemberListArgs
- FinanceRulesApi.DataCellArgs
- FinanceRulesApi.FXRateArgs
- FinanceRulesApi.ConfirmationRuleArgs
- FinanceRulesApi.CalculateArgs
- FinanceRulesApi.DrillDownArgs

Common Usage

The list below details the common use cases that apply to Finance Business Rules:

- Stored Calculation of a Member Value
- Dynamic Calculation of a Member Value
- Programmatic Member Filters
- Scenario Copy Logic
- Allocation Logic
- Conditional No Input Rules
- Custom Consolidation Logic *(Shared Business Rule only)*
- Custom Translation Logic *(Shared Business Rule only)*

- Custom Elimination Logic *(Shared Business Rule only)*
- Confirmation Rule Logic
- Custom Calculations *(Done via Dashboard Parameter Components)*

Parser

Parser Business Rules are used to evaluate and/or modify field values being 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.

Invoking Engine
Stage

API Object Type
ParserDimension

Args Object Type
ParserArgs

Common Usage

The list below details the common use cases that apply to Parser Business Rules.

- Custom Parsing Logic
- Field Value Concatenation
- Field Value Bypassing
- Evaluate Field other than Current Field being Parsed

Connector

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.

Invoking Engine
Stage

API Object Type
Transformer

Args Object Type
ConnectorArgs

Common Usage

The list below details the common use cases that apply to Connector Business Rules.

- Source System Connection Logic
- Source System Field List Logic
- Source System GetData Logic
- Source System DrillBack Logic

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.

Invoking Engine
Stage

API Object Type
Transformer

Args Object Type
ConditionalRuleArgs

Common Usage

The list below details the common use cases that apply to Conditional *(mapping)* Business Rules.

- Evaluate Source Values and Conditional Map Target
- Evaluate Other Mapped Value and Conditional Map Target

DerivativeRule

Derivative Rules *(derive data prior to mapping)* are used to evaluate and/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.

Invoking Engine
Stage

API Object Type
Transformer

Args Object Type
DerivativeRuleArgs

Common Usage

The list below details the common use cases that apply to Derivative *(derived data)* Business Rules.

- Calculate Mathematical Expressions
- Lookup Value from Transformation Cache for use in Calculations
- Lookup Value from Cube for use in Calculations
- Source System Check Rule Logic *(validation rules on source data)*

Cube View Extender

Cube View Extender Rules are used to apply advanced Cube View formatting to any Cube View Report. Using custom formatting allows the Cube View design to go beyond the standard Cube View formatting properties and provides flexibility for specific formatting needs. The Extender Rule is used in conjunction with the Custom Report Formatting properties on the Cube View under General Settings|Report Tab.

Invoking Engine
Presentation

API Object Type
No specific API *(used General BRApi)*

Args Object Type

CubeView

CubeViewExtenderFunctionType

CubeViewExtenderReport

CustomSubVars

FunctionType

Common Usage

- 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
  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
- Customize the Cube View Header
  - Control the Left, Right, Center Subtitle widths
  - Control the font size of Title and Subtitles
- Customize the date display
- Customize bottom text alignment
- Apply Conditional Formatting
  Format cells based on their contents. Change the text color of a value in order to effectively hide the result.
- Customized Report row and column formatting such as borders, background and text colors and alignment

DashboardDataSet

DashboardDataSet 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 or C#. These Business Rules are written as Shared Business Rules and applied to Dashboard Data Adapters or Dashboard Parameters.

Invoking Engine
Presentation

API Object Type

No specific API (used General BRApi)

Args Object Type
DashboardDataSetArgs

Common Usage

The list below details the common use cases that apply to DashboardDataSet Business Rules.

- Combine Different Types of Data for a Report
- Build Programmatic Data Queries *(e.g., analytic plus SQL)*
- Conditionally Build Data Query Reports
- Conditionally Build Data Query Parameters

DashboardExtender

DashboardExtender Rules are used to perform a variety of tasks associated with custom Dashboards and MarketPlace Solutions. These Business Rules can be thought of as multi-purpose rules and make up the majority of the code written in a MarketPlace Solution. In addition, they are written as Shared Business Rules and applied to Application Dashboard Parameter Components (Buttons, Combo Boxes, etc.).

Invoking Engine
Presentation

API Object Type
No Specific API *(uses General BRApi)*

Args Object Type
DashboardExtenderArgs

Common Usage

The list below details the common use cases that apply to DashboardExtender Business Rules.

- Execute a Task when the User Clicks 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
- These rules are basically limited to the imagination of the developer

DashboardStringFunction

DashboardStringFunction (reference as XFBR) Rules are used to process conditional Dashboard Parameters. These rules inspect and alter a Dashboard Parameter value using the full syntax capabilities of VB.Net or C#. DashboardStringFunctions are written as Shared Business Rules and called by using a XFBR(BusinessRuleName, FunctionName, UserParam=[UserValue]) specification anywhere a standard Dashboard Parameter is used.

Invoking Engine
Presentation

API Object Type
No Specific API *(uses General BRApi)*

Args Object Type DashboardStringFunctionArgs

Common Usage

The list below details the common use cases that apply to DashboardStringFunction (i.e., conditional Parameters) Business Rules.

- Evaluate a Dashboard Parameter and conditionally return another Value
- Evaluate a Cube View Parameter and conditionally return another Value
- This Business Rule can be substituted anywhere a Dashboard Parameter is used in order to evaluate the Supplied Parameter value and return a different value

Extender

Extender Rules are the most generalized type of Business Rule in the platform. Use these to write a simple utility function or a specific helper function called as part of a Data Management Job. These Business Rules are written as Shared Business Rules and executed directly from the code editor, a data management job or the Finance Engine during an external Dimension request *(i.e., read Dimension Members from an external list).*

Invoking Engine Business Rule, Data Management, Finance

API Object Type No Specific API *(uses General BRApi)*

Args Object Type

ExtenderArgs

This contains multiple child objects that are populated based on how the rule type is called.

- ExtenderArgs.DataMgmtArgs
- ExtenderArgs.ExternalDimSourceArgs

Common Usage

The list below details the common use cases that apply to Extender Business Rules.

- Create a General Helper Rule for Administrators Only
- Create Data Management Business Rule Step Logic
- Create a Query to fill an External Dimension List

Organizing and Referencing Business Rules

The Business Rule framework provided organizes business rules to maximize their reuse. You can link business rules and reference one business rule from another. You can also link and call external DLLs from a business rule. This section describes how to reference a shared business rule and an external DLL from another business rule.

Defining a Reference to a Shared Business Rule

When you create a shared business rule is created, its public members can be referenced and run by other shared and item specific business rules. Creating a shared or referenced business rule lets you:

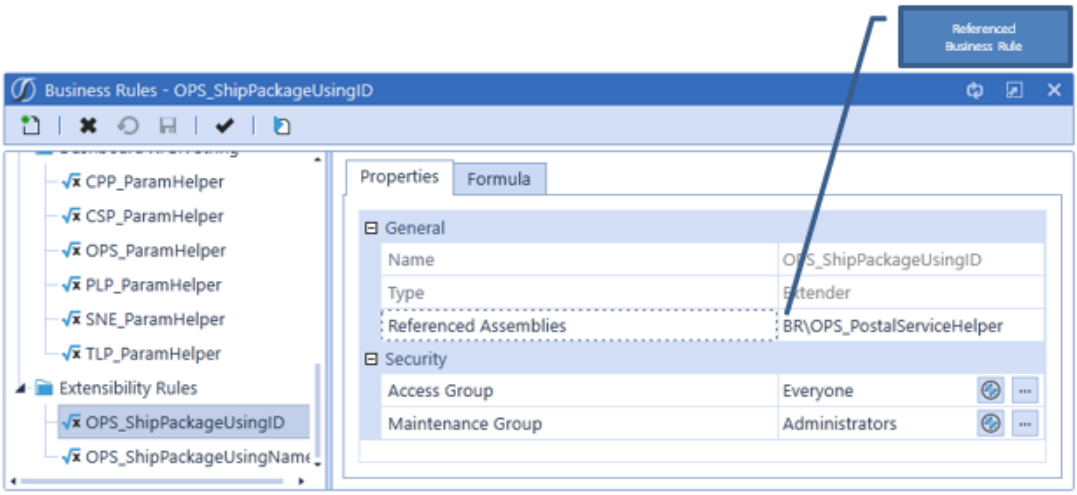
- Create a list of shared constant values.
- Create a set of standard helper functions.
- Centralize the maintenance of shared logic.

Reference Syntax

This section defines the syntax required to reference a shared business rule from another shared or item specific business rule.

Shared business rules referencing other shared business rules
To create a reference from one shared business rule to another, go to the rule calling a Public Method of another shared business rule and make a declaration in the Referenced Assemblies property. The syntax requires a BR\ prefix and the business rule name to reference. A rule may reference either a VB.NET or C# rule.

> **TIP:** Reference multiple business rules by creating a comma-separated list of reference statements.



Syntax

BR\<business rule name to reference>

Example (Single Reference)

BR\OPS_PostalServiceHelper

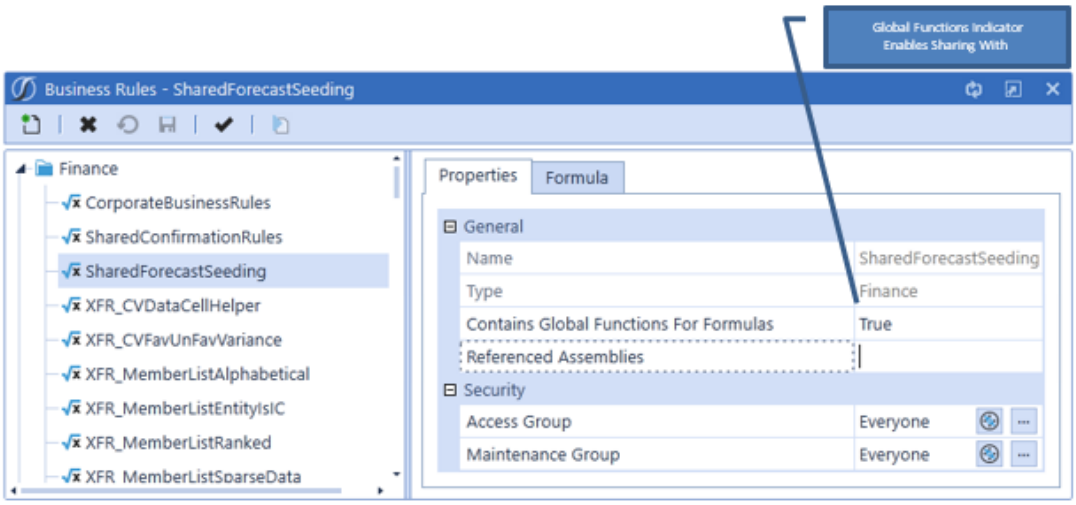
Example (Multiple References)

BR\OPS_PostalServiceHelper; BR\CPP_SolutionHelper

Referencing a Shared Business Rule From an Item Specific Business Rule

Finance, Parser, ConditionalRule and DeriviativeRule shared business rules have equivalent item specific business rules. When you create a shared business rule, set the *Contains Global Functions For Formulas* property to *True* to make the rule available to I\item specific business rules. Item specific business rules do not have a *Referenced Assemblies* property so can only reference shared rules of the same engine type with the *Contains Global Functions For Formulas* property set to *True.*

In the example below, the SharedForecastSeeding rule can be called from any other Finance rule because its *Contains Global Functions For Formulas* property is *True.*



NOTE: If a Finance business rule has *Contains Global Functions For Formulas* set to *True*, changes to the business rule have a metadata status impact and change the Calculation Status to *OK, MC*. This dependency must occur because a global rule can be used by a member formula calculation which can impact the status of the Finance Engine's data *(analytic / Cube data)*.

Using a Code Declaration

Once a reference is made to a shared business rule, its Public Methods *(Functions / Subs)* can be called. To access the Public Methods, declare an instance of the rule in the code using the Business Rule's fully qualified Namespace. This creates an object variable that references the shared business rule calls its Public Methods.

Example Declaration

'Declaring an object variable to reference a shared business rule.

```
Dim opsHelper As New OneStream.BusinessRule.DashboardExtender.OPS_
PostalServiceHelper.MainClass
```

Example Usage

'Executing a function on the Reference business rule object variable

```
Dim desc As String opsHelper.GetFieldFromID(si, "Dashboard", "Name",
dashName, "Description")
```

Referencing an External .Net DLL

Developers can build and reference custom Microsoft .Net DLLs from shared business rules. These are written in either VB.Net or C#. Custom, encapsulated business logic can be protected within an external DLL written in Microsoft Visual Studio.

Create a DLL referenced by a business rule to:

- Protect domain specific intellectual property *(hide value programming logic).*
- Separate code with dependencies on other programs *(system integration wrappers).*
- Complex logic requiring development tools only available within Microsoft Visual Studio *(Web Service Discovery and Interface Development).*

Installing and Configuring DLLs

Perform these tasks to enable an external DLL to be referenced from a shared business rule.

1. Specify the BusinessRuleAssemblyFolder located in the Application Server configuration file. This folder should be shared by all application servers. The folder must be accessible via the *Account Credentials* used to configure the IIS Application Pool on the application server.

   This setup is a best practice, but not required. Alternatively, you can reference the external DLL from a folder on each application server. When the DLL is updated, copy it to a standard folder on each application server.

2. Identify or create the external DLL to be called and copy it to BusinessRuleAssemblyFolder. When a business rule runs and an external DLL reference with the XF\ prefix is found in the Referenced Assemblies property of the rule, the application server looks in the BusinessRuleAssemblyFolder specified in the application server configuration file to find the DLL to reference.

3. Add a reference specification to the DLL in the **Referenced Assemblies** property of the business rules using it.

Reference Specification

This section defines the syntax required to reference an external DLL using the shared business rule's *Referenced Assemblies* property. There are three methods to reference an external DLL.

Method 1
This method uses the *XF* prefix to create a reference to an external DLL located in the *BusinessRuleAssemblyFolder* folder which is specified in the application server configuration file.

Syntax
XF\<External DLL Name to Reference>

Example (Single Reference)
XF\ExternalCode.DLL

Example (Multiple References)
XF\ExternalCode1.DLL;XF\ExternalCode2.DLL

Method 2
This method uses the file system path *C:\DLLFolderName* to create a reference to an external DLL on each application server.

> **NOTE:** The same folder path and DLL must exist on all application servers. This method is not a best practice for custom business logic DLLs because it increases maintenance.

You can use a file system path to reference an external DLL that already exists on an application server, as part of the operating system or as an installed component.

Syntax
C:\DLLFolderName\<External DLL Name to Reference>

Example (Single Reference)
C:\DLLFolderName\ExternalCode.DLL

Example (Multiple References)
C:\DLLFolderName\ExternalCode1.DLL; C:\DLLFolder\ExternalCode2.DLL

Code Declaration

Once a reference is made to an External DLL from a shared business rule, the Public Methods *(Functions / Subs)* of that external DLL can be called. To access the shared business rule's Public Methods, declare an Import to the Namespaces defined by the DLL, then create an instance of the desired class to use in the code.

Example Import

Imports YourNamespace.SubNamespace

Example Declaration

‘Declaring an object variable to reference a class on the external DLL

Dim extHelper As New YourClass

Example Usage

‘Executing a Function on the external DLL

Dim desc As String extHelper.YourFunciton(“SomeParameter”)

Method 3
This method uses a Windows environment variable to create a reference to an external DLL. All standard Windows paths are supported and the name is determined by .NET.

Syntax
%System%\DLLName.DLL

Example
%userprofile%\documents\WindowsBase.DLL

API Structure and Organization

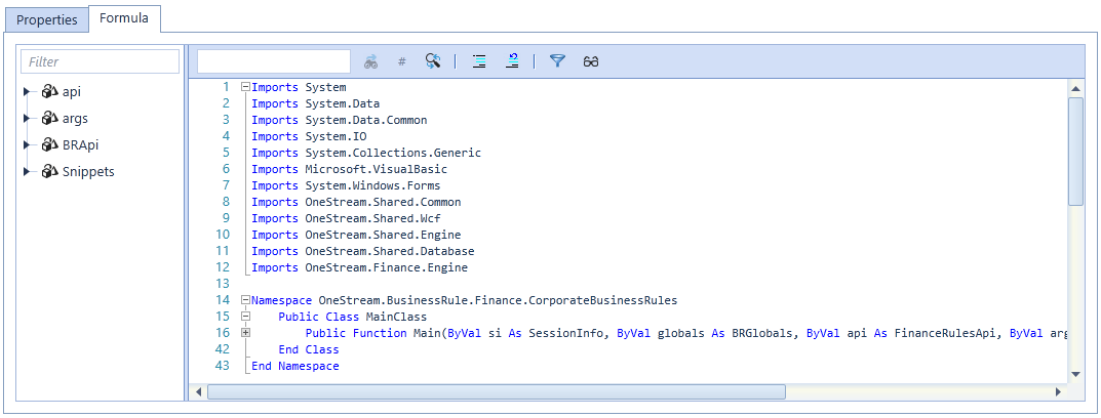
Namespaces

The Microsoft .Net Framework organizes code libraries into subject areas called Namespaces. The process begins with identifying the Namespaces *(libraries)* required for the procedure being created. Namespaces provide distinction to the objects and methods that exist in a code library. As a best practice, Namespaces typically start with the name of the company that created the code library.This prevents naming conflicts for objects that share a common name, but were created by different software providers.

In an effort to keep coding syntax as terse as possible, the .Net Framework allows the user to specify common Namespaces to use at the top of a Business Rule. These lines are preceded by the key word *Imports.* Adding Imports Statements prevents having to type an object's fully qualified name within a Namespace.

All Business Rules are prepopulated with both the commonly used Microsoft Namespaces as well as the OneStream specific Namespaces. For example, adding the statement *Imports System.Math* to a Business Rule enables access to objects in the *System.Math* Namespace. Instead of typing *System.Math.Round(100.05,0),* type *Round(100.05,0).*

The example below shows the Namespace references used in a standard Extensibility Rule.



Namespaces Defined

OneStream is a large and sophisticated software platform and consequently a great deal of effort went into organizing the code base into a hierarchical set of Namespaces. This section defines the Namespace hierarchy and explains the primary purpose of the code libraries in each Namespace. It is important to understand structure and meaning of the platform Namespaces because most API methods accept and return objects defined within specific Namespaces. By understanding the structure of the Namespace hierarchy, developers can browse for objects using intelli-sense in the syntax editor.

Namespace Hierarchy

The hierarchy below denotes the platform Namespaces and the object libraries contained within them. This hierarchy is explored from within the Business Rule syntax editor by typing *OneStream.* and navigating through the intelli-sense popup lists. This technique helps find objects to pass into an API function, objects returned from an API function, or common helper classes available in the platform.

```
OneStream (Root Namespace)
OneStream.BusinessRule
OneStream.BusinessRule.Finance
OneStream.BusinessRule.Parser
OneStream.BusinessRule.Connector
OneStream.BusinessRule.ConditionalRule
OneStream.BusinessRule.DerivativeRule
OneStream.BusinessRule.DashboardDataSet
OneStream.BusinessRule.DashboardExtender
OneStream.BusinessRule.DashboardStringFunction
OneStream.BusinessRule.Extender
OneStream.Client
OneStream.Client.SharedUI
OneStream.Client.SharedUI.FinanceMsgStrings
OneStream.Client.SharedUI.FinanceUIStrings
OneStream.Client.SharedUI.GeneralMsgStrings
OneStream.Client.SharedUI.GeneralUIStrings
OneStream.Client.SharedUI.StageMsgStrings
OneStream.Client.SharedUI.StageUIStrings
OneStream.Client.SharedUI.StringResourceFileType
OneStream.Client.SharedUI.StringResourceHelper
```

```
OneStream.Client.SharedUI.XFStrings
OneStream.Finance
OneStream.Finance.Engine
OneStream.Finance.Engine.DataApi
OneStream.Finance.Engine.EvalDataBufferDelegate
OneStream.Finance.Engine.FinanceRulesApi
OneStream.Finance.Engine.IAccountApi
OneStream.Finance.Engine.ICalcStatusApi
OneStream.Finance.Engine.IConsApi
OneStream.Finance.Engine.ICubesApi
OneStream.Finance.Engine.IDimensionsApi
OneStream.Finance.Engine.IEntityApi
OneStream.Finance.Engine.IFlowApi
OneStream.Finance.Engine.IFunctionsApi
OneStream.Finance.Engine.IFxRatesApi
OneStream.Finance.Engine.IMembersApi
OneStream.Finance.Engine.IPovApi
OneStream.Finance.Engine.IScenarioApi
OneStream.Finance.Engine.ITimeApi
OneStream.Finance.Engine.IUDApi
OneStream.Finance.Engine.IViewApi
OneStream.Finance.Engine.IWorkflowApi
OneStream.Stage
OneStream.Stage.Engine
OneStream.Stage.Engine.Parser
OneStream.Stage.Engine.ParserDimension
OneStream.Stage.Engine.TransformerDataCache
OneStream.Stage.Engine.Transformer
OneStream.Stage.Engine.TransformerDimension
OneStream.Stage.Engine.TransformRuleCache
OneStream.Shared
OneStream.Shared.Engine
OneStream.Shared.Engine.ExternalWcfClient
OneStream.Shared.Engine.TaskActivityStepWrapperItem
OneStream.Shared.Database
OneStream.Shared.Database.DbConnInfo
OneStream.Shared.Common
```

```
OneStream.Shared.Common.(Various Constants, Helper Classes & Data Transfer Objects ‘DTO’ )
OneStream.Shared.Wcf
OneStream.Shared.Wcf.(Various Constants & Data Transfer Objects ‘DTO’)
```

Microsoft Financial Calls

Financial calls are part of the Microsoft.VisualBasic namespace, and can be used to for calculations such as:

- Depreciation
- Present and future values
- Interest rates
- Rates of return
- Payments

These functions are available to anyone with access to Business Rules. They can be explored within the Business Rule syntax editor by typing Microsoft.VisualBasic.Financial then navigating through the intelli-sense popup lists.

To view all methods from the Microsoft.Visual Basic Financial class used in a Business Rule:

1. Navigate to the Business Rule Editor:
    a. In the OneStream Software application, click the **Application** tab.
    b. Under Tools, click **Business Rules**.
    c. Expand the appropriate Business Rules category or click **Search** on the toolbar.
2. Click the **Formula** tab.
3. In the editor window, type **Microsoft.Visualbasic.Financial**.

A list of methods displays.

```
12  Imports OneStream.Shared.Engine
13  Imports OneStream.Shared.Database
14  Imports OneStream.Stage.Engine
15  Imports OneStream.Stage.Database
16  Imports OneStream.Finance.Engine
17  Imports OneStream.Finance.Database
18
19  Namespace OneStream.BusinessRule.Extender.ATony
20      Public Class MainClass
21          Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api
22              Try
23                  Select Case args.FunctionType
24
25                      Case Is = ExtenderFunctionType.Unknown
26
27                          Dim mydatacell As DataCell = BRapi.Finance.Data.GetDataCellsUsingMe
28                          api.LogMessage(mydatacell.DataCellPk.GetMemberScript(api) + " - IsL
29                      Case Is = ExtenderFunctionType.ExecuteDataMgmtBusinessRuleStep
30                      microsoft.VisualBasic.Financial.
31                      End Select
32
33                  Return Nothing
34              Catch ex As Exception

Sample
Dim fieldTokens As New List(Of String)
fieldTokens.Add("xfGuid#:[Field1]::NewGuid")
fieldTokens.Add("xfText#:[Field2]")
fieldTokens.Add("xfInt#:[Field3]")

DDB
Equals
FV
IPmt
IRR
MIRR
NPer
NPV
Pmt
```

See Business Rules for more information.

In-Solution Development

In-solution development is the process of creating OneStream Business Rules to deliver domain specific solutions. This means that all Business Rules are executed within the application server process space. The code written is only executed on the application servers where OneStream is deployed.

Developing within the application server environment enables solution developers to focus on the business problem instead of common programming concerns. The platform takes care of managing connections, moving data between application tiers, and load balancing server activities.

In some cases, in-solution development is seen as a limitation because the developer is restricted to coding within the application server tier. However, in most cases the efficiency and quality gained by developing within the platform out ways any limitations imposed by coding at the application server tier.

Custom Development

Custom development refers to stand alone application development that interacts with the platform at the web server tier.

Custom Web Development

The platform has the ability to display web pages within a custom Dashboard. This allows completely custom web applications to surface within the OneStream solution. OneStream can pass information about the user's POV and Workflow as URL Parameters enabling the custom web application to act as part of an integrated solution.

With this capability, developers are free to create and incorporate any solution they can imagine.

Using System Tools

System Business Rules

System Extender Business Rules are used in coordination with Azure Server Sets for elastic scalability at the Azure Database and Server Sets level. Server and eDTU scaling can be accomplished manually or via System Business Rules. If System Business Rules is selected as a Scaling Type, then OneStream will call a user-defined System Extender Business Rule to determine if scaling is needed. The user is responsible for implementing the scaling function and returning the proper scaling object to OneStream. This can be accomplished by adding a System Extender Business Rule and assigning it appropriately.

Under each Case statement, these rules and related Args and BRApis can be used to check the current Server Set capacity, query metrics about a Server Set or Azure Database and impact the volume of Server Sets or level of Azure Database deployed.

Refer to the *Installation and Configuration Guide* under *Azure Database Connection Settings* and *Server Sets* for where to refer to these Business Rules. Example starting point of empty System Extender Business Rule upon creation:

```
Namespace OneStream.BusinessRule.SystemExtender.Test
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As Object, ByVal args As SystemExtenderArgs) As Object
            Try
                Select Case args.FunctionType

                    Case Is = SystemExtenderFunctionType.Unknown

                    Case Is = SystemExtenderFunctionType.GetDesiredServerSetCapacity

                    Case Is = SystemExtenderFunctionType.GetDesiredElasticDatabasePoolCapacity

                    Case Is = SystemExtenderFunctionType.GetDesiredExternalServerSetCapacity

                End Select

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

Sample System Business Rule

Metrics data is passed to this function to help the user determine whether the server or database needs to be scaled or not. Depending on what is being scaled, different metric data is passed in. For server scaling, Environment metrics and Scale Set metrics are passed in to help determine scaling. For database scaling, Environment metrics and SQL Server Elastic Pool metrics are passed in to help determine scaling.

```
Select Case args.FunctionType

    Case Is = SystemExtenderFunctionType.Unknown

    Case Is = SystemExtenderFunctionType.GetDesiredScaleSetCapacity
        Dim systemExtenderScaleSetResult As New SystemExtenderScaleSetResult
        systemExtenderScaleSetResult.Capacity = args.ScaleSetArgs.CurrentScaleSetCapacity

        If (args.ScaleSetArgs.ScaleSetMetricValues.AvgCPUUtilization > 50) Then
            systemExtenderScaleSetResult.Capacity = args.ScaleSetArgs.CurrentScaleSetCapacity + 1
        End If

        Return systemExtenderScaleSetResult

    Case Is = SystemExtenderFunctionType.GetDesiredElasticDatabasePoolCapacity
        Dim systemExtenderSQLServerElasticPoolResult As New SystemExtenderSQLServerElasticPoolResult
        systemExtenderSQLServerElasticPoolResult.AzureElasticPoolDTU = args.SQLServerElasticPoolArgs.DatabaseAndEPoolDTU.AzureElasticPoolDTU

        If (args.SQLServerElasticPoolArgs.AzureElasticPoolLevelMetricValues.DTUConsumptionPercent > 90)
            systemExtenderSQLServerElasticPoolResult.AzureElasticPoolDTU = 1600
        End If

        Return systemExtenderSQLServerElasticPoolResult

    Case Is = SystemExtenderFunctionType.GetDesiredExternalScaleSetCapacity

End Select
```

Database

The Database screen allows System Administrators to view all of OneStream's database tables and provides tools for managing stored data and other information.

Tables

This gives read-only access to all data tables in the database and can be used for tasks such as trying to debug issues without having access to the database, or deletion logging.

Tools

Database Tools allow System Administrators to manage the database.

Data Records

Enter a Member Filter in order to view data for the entire system.

Event Listing

Event Handler Business Rules

WCF Event Handler
This allows 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 such as 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 e-mail an auditor about every metadata change as it happens.

Transformation Event Handler
This can be run at various points from Import through Load. 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
This can be run before, during, or after a Journal operation such as Submission, Approval, or Post. Available operations:

SubmitJournal

ApproveJournal

RejectJournal

PostJournal

UnpostJournal

StartUpdateJournalWorkflow

EndUpdateJournalWorkflow

FinalizeUpdateJournalWorkflow

Save Data Event Handler
This is run in order to track all save events in an application.

Forms Event Handler
This can be run before, during, or after an operation such as Form Save. Available operations:

SaveForm

CompleteForm

RevertForm

StartUpdateFormWorkflow

EndUpdateFormWorkflow

FinalizeUpdateFormWorkflow

Data Quality Event Handler
This can be run before, during, or after data quality events like Confirmation and Certification. Available operations:

StartProcessCube

Calculate

Translate

Consolidate

EndProcessCube

FinalizeProcessCube

PrepareICMatch

StartICMatch

PrepareICMatchData

EndICMatch

StartConfirm

EndConfirm

FinalizeConfirm

SaveQuestionResponse

StartSetQuestionairreState

SaveQuestionairreState

EndSetQuestionairreState

StartSetCertifyState

SaveCertifyState

EndSetCertifyState

FinalizeSetCertifyState

Data Management Event Handler
This can be run before or after a Data Management Sequence or Step runs. Available operations:

StartSequence

ExecuteStep

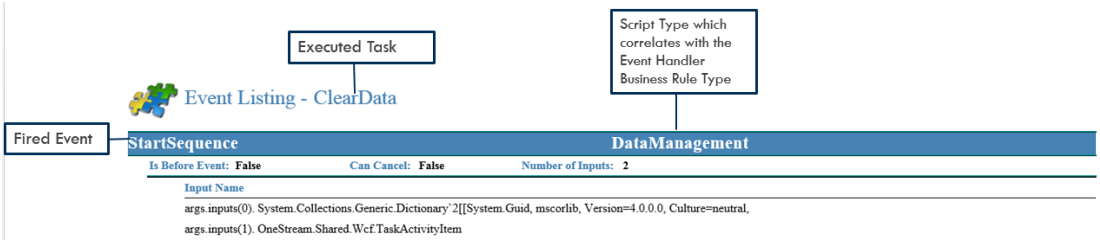
EndSequence

Workflow Event Handler
This can be run before or after a Workflow execution step. Available operations:

UpdateWorkflowStatus
WorkflowLock
WorkflowUnlock

Event Firing Sequences

OneStream fires a series of events when completing tasks via Event Handler Business Rules. The example below explains how to read the table which provides the firing sequence when running a specific task.



Clear Cube Data

StartSequence — **DataManagement**

Is Before Event: **False** Can Cancel: **False** Number of Inputs: **2**

Input Name
args.inputs(0). System.Collections.Generic.Dictionary`2[[System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral,
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

ExecuteStep — **DataManagement**

Is Before Event: **True** Can Cancel: **False** Number of Inputs: **2**

Input Name
args.inputs(0). OneStream.Finance.Engine.DataMgmtStepMetadataInfo
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

SaveCubeData — **SaveData**

Is Before Event: **True** Can Cancel: **True** Number of Inputs: **0**

Input Name
args.inputs(0). SAVE DATA EVENT IS USED FOR DEBUG ONLY

UpdateWorkflowStatus — **Workflow**

Is Before Event: **True** Can Cancel: **True** Number of Inputs: **7**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: **False** Can Cancel: **True** Number of Inputs: **7**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

ExecuteStep — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). OneStream.Finance.Engine.DataMgmtStepMetadataInfo

ExecuteStep — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

EndSequence — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). System.Collections.Generic.Dictionary`2[[System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral,
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

Clear Stage Data

StartSequence — **DataManagement**

Is Before Event: **False** Can Cancel: **False** Number of Inputs: **2**

Input Name
args.inputs(0). System.Collections.Generic.Dictionary`2[[System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral,
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

ExecuteStep — **DataManagement**

Is Before Event: **True** Can Cancel: **False** Number of Inputs: **2**

Input Name
args.inputs(0). OneStream.Finance.Engine.DataMgmtStepMetadataInfo
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

SaveCubeData — **SaveData**

Is Before Event: **True** Can Cancel: **True** Number of Inputs: **0**

Input Name
args.inputs(0). SAVE DATA EVENT IS USED FOR DEBUG ONLY

UpdateWorkflowStatus — **Workflow**

Is Before Event: **True** Can Cancel: **True** Number of Inputs: **7**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: **False** Can Cancel: **True** Number of Inputs: **7**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

ExecuteStep — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). OneStream.Finance.Engine.DataMgmtStepMetadataInfo

ExecuteStep — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

EndSequence — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). System.Collections.Generic.Dictionary`2[[System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral,
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

Execute Data Management

StartSequence — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). System.Collections.Generic.Dictionary`2[[System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral,
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

ExecuteStep — **DataManagement**

Is Before Event: True | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). OneStream.Finance.Engine.DataMgmtStepMetadataInfo
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

ExecuteStep — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). OneStream.Finance.Engine.DataMgmtStepMetadataInfo
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

EndSequence — **DataManagement**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). System.Collections.Generic.Dictionary`2[[System.Guid, mscorlib, Version=4.0.0.0, Culture=neutral,
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem

Import Data Connection

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

SaveCubeData — **SaveData**

Is Before Event: True | Can Cancel: True | Number of Inputs: 0

Input Name
args.inputs(0). SAVE DATA EVENT IS USED FOR DEBUG ONLY

StartLoadIntersect — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode

StartLoadIntersect — **Transformation**

Is Before Event: **True** | Can Cancel: **False** | Number of Inputs: **5**

Input Name
args.inputs(4). System.Guid

EndLoadIntersect — **Transformation**

Is Before Event: **False** | Can Cancel: **False** | Number of Inputs: **5**

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: **True** | Can Cancel: **True** | Number of Inputs: **7**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: **False** | Can Cancel: **True** | Number of Inputs: **7**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String

UpdateWorkflowStatus — **Workflow**

Is Before Event: **False** | Can Cancel: **True** | Number of Inputs: **7**

Input Name
args.inputs(6). System.Guid

FinalizeLoadIntersect — **Transformation**

Is Before Event: **False** | Can Cancel: **False** | Number of Inputs: **5**

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

Import Excel File

StartParseAndTransform	Transformation

Is Before Event: False Can Cancel: False Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

InitializeTransformer	Transformation

Is Before Event: True Can Cancel: True Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

InitializeTransformer	Transformation

Is Before Event: False Can Cancel: True Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ParseSourceData	Transformation

Is Before Event: True Can Cancel: False Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

InitializeExcelRangeLayout — **Transformation**

Is Before Event: **True** | Can Cancel: **False** | Number of Inputs: **2**

Input Name
args.inputs(0). OneStream.Stage.Engine.Parser
args.inputs(1). OneStream.Shared.Engine.StageRangeContent

InitializeExcelRangeLayout — **Transformation**

Is Before Event: **False** | Can Cancel: **False** | Number of Inputs: **2**

Input Name
args.inputs(0). OneStream.Stage.Engine.Parser
args.inputs(1). OneStream.Shared.Engine.StageRangeContent

ParseSourceData — **Transformation**

Is Before Event: **False** | Can Cancel: **False** | Number of Inputs: **4**

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ProcessDerivedRules — **Transformation**

Is Before Event: **True** | Can Cancel: **False** | Number of Inputs: **4**

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ProcessDerivedRules — **Transformation**

Is Before Event: **False** | Can Cancel: **False** | Number of Inputs: **4**

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes

ProcessDerivedRules — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(3). System.Guid

ProcessTransformRules — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ProcessTransformRules — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

DeleteData — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

DeleteData — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String

DeleteData — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

DeleteRuleHistory — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

DeleteRuleHistory — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

WriteTransformedData — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

WriteTransformedData — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer

WriteTransformedData — **Transformation**

Is Before Event: **False** Can Cancel: **False** Number of Inputs: **4**

Input Name
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

SummarizeTransformedData — **Transformation**

Is Before Event: **True** Can Cancel: **False** Number of Inputs: **4**

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

SummarizeTransformedData — **Transformation**

Is Before Event: **False** Can Cancel: **False** Number of Inputs: **4**

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

CreateRuleHistory — **Transformation**

Is Before Event: **True** Can Cancel: **False** Number of Inputs: **4**

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

CreateRuleHistory — **Transformation**

Is Before Event: **False** Can Cancel: **False** Number of Inputs: **4**

Input Name

CreateRuleHistory — Transformation

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

EndParseAndTransform — Transformation

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

UpdateWorkflowStatus — Workflow

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — Workflow

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String

UpdateWorkflowStatus — Workflow

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

FinalizeParseAndTransform — Transformation

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

Import Text File

StartParseAndTransform — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

InitializeTransformer — **Transformation**

Is Before Event: True | Can Cancel: True | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

InitializeTransformer — **Transformation**

Is Before Event: False | Can Cancel: True | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ParseSourceData — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ParseSourceData — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ProcessDerivedRules — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ProcessDerivedRules — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ProcessTransformRules — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

ProcessTransformRules — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

DeleteData — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

DeleteData — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

DeleteRuleHistory — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

DeleteRuleHistory — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

WriteTransformedData — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

WriteTransformedData — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

SummarizeTransformedData — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

SummarizeTransformedData — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

CreateRuleHistory — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

CreateRuleHistory — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

EndParseAndTransform — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

FinalizeParseAndTransform — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Stage.Engine.Transformer
args.inputs(1). System.String
args.inputs(2). OneStream.Shared.Common.TransformLoadMethodTypes
args.inputs(3). System.Guid

Process Form

CompleteForm — **Forms**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.XFFormEx
args.inputs(1). System.Boolean
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Common.WorkflowStatusTypes

CompleteForm — **Forms**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.XFFormEx
args.inputs(1). System.Boolean
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Common.WorkflowStatusTypes

CompleteForm — **Forms**

Is Before Event: True | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.XFFormEx
args.inputs(1). System.Boolean
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Common.WorkflowStatusTypes

CompleteForm — **Forms**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.XFFormEx
args.inputs(1). System.Boolean
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Common.WorkflowStatusTypes

StartUpdateFormWorkflow — **Forms**

Is Before Event: False | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.InputFormsProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean

EndUpdateFormWorkflow — **Forms**

Is Before Event: False | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.InputFormsProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(6). System.Guid

Process Journal

SubmitJournal		Journals
Is Before Event: True	Can Cancel: False	Number of Inputs: 2

Input Name
args.inputs(0). System.Guid
args.inputs(1). OneStream.Shared.Wcf.JournalEx

SubmitJournal		Journals
Is Before Event: False	Can Cancel: False	Number of Inputs: 2

Input Name
args.inputs(0). System.Guid
args.inputs(1). OneStream.Shared.Wcf.JournalEx

FinalizeSubmitJournal		Journals
Is Before Event: False	Can Cancel: False	Number of Inputs: 1

Input Name
args.inputs(0). System.Guid

ApproveJournal		Journals
Is Before Event: True	Can Cancel: False	Number of Inputs: 2

Input Name
args.inputs(0). System.Guid
args.inputs(1). OneStream.Shared.Wcf.JournalEx

ApproveJournal		Journals
Is Before Event: False	Can Cancel: False	Number of Inputs: 2

Input Name
args.inputs(0). System.Guid
args.inputs(1). OneStream.Shared.Wcf.JournalEx

FinalizeApproveJournal		Journals
Is Before Event: False	Can Cancel: False	Number of Inputs: 1

Input Name
args.inputs(0). System.Guid

PostJournal — **Journals**

Is Before Event: True | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). System.Guid
args.inputs(1). OneStream.Shared.Wcf.JournalEx

SaveCubeData — **SaveData**

Is Before Event: True | Can Cancel: True | Number of Inputs: 0

Input Name
args.inputs(0). SAVE DATA EVENT IS USED FOR DEBUG ONLY

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

PostJournal — **Journals**

Is Before Event: False | Can Cancel: False | Number of Inputs: 2

Input Name
args.inputs(0). System.Guid
args.inputs(1). OneStream.Shared.Wcf.JournalEx

FinalizePostJournal — **Journals**

Is Before Event: False | Can Cancel: False | Number of Inputs: 1

Input Name
args.inputs(0). System.Guid

StartUpdateJournalWorkflow — **Journals**

Is Before Event: False | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.InputJournalsProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean

EndUpdateJournalWorkflow — **Journals**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.InputJournalsProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.JournalsAndTemplatesForWorkflow

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

FinalizeUpdateJournalWorkflow — **Journals**

Is Before Event: False | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.InputJournalsProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean

Process Workflow

StartValidateTransform — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.ValidationTransformationProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid

ValidateDimension — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(4). System.Guid

SetEventRules — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.ValidationTransformationProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). System.Guid

EndValidateTransform — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.ValidationTransformationProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus		Workflow
Is Before Event: False	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

FinalizeValidateTransform		Transformation
Is Before Event: False	Can Cancel: False	Number of Inputs: 4

Input Name
args.inputs(0). OneStream.Shared.Wcf.ValidationTransformationProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). System.Guid

StartValidateIntersect		Transformation
Is Before Event: True	Can Cancel: False	Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.ValidateIntersectionProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

UpdateWorkflowStatus		Workflow
Is Before Event: True	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes

UpdateWorkflowStatus		Workflow
Is Before Event: True	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus		Workflow
Is Before Event: False	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

EndValidateIntersect		Transformation
Is Before Event: False	Can Cancel: False	Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.ValidateIntersectionProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

UpdateWorkflowStatus		Workflow
Is Before Event: True	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes

UpdateWorkflowStatus		Workflow
Is Before Event: True	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus		Workflow
Is Before Event: False	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

FinalizeValidateIntersect		Transformation
Is Before Event: False	Can Cancel: False	Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.ValidateIntersectionProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

UpdateWorkflowStatus		Workflow
Is Before Event: True	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

SaveCubeData — **SaveData**

Is Before Event: True | Can Cancel: True | Number of Inputs: 0

Input Name
args.inputs(0). SAVE DATA EVENT IS USED FOR DEBUG ONLY

StartLoadIntersect — **Transformation**

Is Before Event: True | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

EndLoadIntersect — **Transformation**

Is Before Event: False | Can Cancel: False | Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

UpdateWorkflowStatus		Workflow
Is Before Event: True	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus		Workflow
Is Before Event: False	Can Cancel: True	Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

FinalizeLoadIntersect		Transformation
Is Before Event: False	Can Cancel: False	Number of Inputs: 5

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

StartLoadIntersect		Transformation
Is Before Event: True	Can Cancel: False	Number of Inputs: 5

StartLoadIntersect — **Transformation**

Is Before Event: **True** Can Cancel: **False** Number of Inputs: **5**

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

EndLoadIntersect — **Transformation**

Is Before Event: **False** Can Cancel: **False** Number of Inputs: **5**

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: **True** Can Cancel: **True** Number of Inputs: **7**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: **False** Can Cancel: **True** Number of Inputs: **7**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes

UpdateWorkflowStatus — **Workflow**

Is Before Event: **False** Can Cancel: **True** Number of Inputs: **7**

Input Name
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

FinalizeLoadIntersect — **Transformation**

Is Before Event: **False** Can Cancel: **False** Number of Inputs: **5**

Input Name
args.inputs(0). OneStream.Shared.Wcf.LoadCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Wcf.LoadDataMode
args.inputs(4). System.Guid

StartProcessCube — **DataQuality**

Is Before Event: **False** Can Cancel: **False** Number of Inputs: **3**

Input Name
args.inputs(0). OneStream.Shared.Wcf.ProcessCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). OneStream.Shared.Wcf.TaskActivityItem

Consolidate — **DataQuality**

Is Before Event: **True** Can Cancel: **False** Number of Inputs: **3**

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem
args.inputs(2). OneStream.Shared.Wcf.DataUnitInfo

Consolidate — **DataQuality**

Is Before Event: False | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem
args.inputs(2). OneStream.Shared.Wcf.DataUnitInfo

NoCalculate — **DataQuality**

Is Before Event: True | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem
args.inputs(2). OneStream.Shared.Wcf.DataUnitInfo

NoCalculate — **DataQuality**

Is Before Event: True | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.TaskActivityItem
args.inputs(2). OneStream.Shared.Wcf.DataUnitInfo

EndProcessCube — **DataQuality**

Is Before Event: False | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.ProcessCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). OneStream.Shared.Wcf.TaskActivityItem

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes

UpdateWorkflowStatus — **Workflow**

Is Before Event: True | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

UpdateWorkflowStatus — **Workflow**

Is Before Event: False | Can Cancel: True | Number of Inputs: 7

Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

FinalizeProcessCube — **DataQuality**

Is Before Event: False | Can Cancel: False | Number of Inputs: 3

Input Name
args.inputs(0). OneStream.Shared.Wcf.ProcessCubeProcessInfo
args.inputs(1). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(2). OneStream.Shared.Wcf.TaskActivityItem

Finance Functions APIs

Member ID

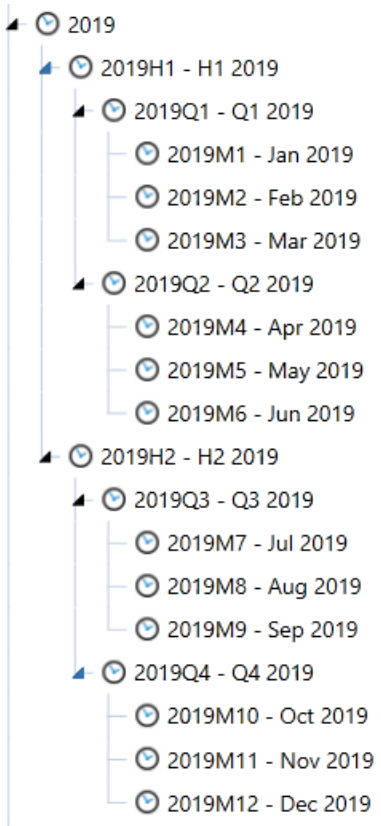
There are many functions that use MemberID as an integer to pass in as a property. These functions get the current POV of the specific Dimension member to perform a variety of tasks, such as:

- Get Current Year based on Time POV
  - Example: Api.Time.GetYearFromId(api.Pov.Time.MemberId)
- Get Text field value from Entity POV
  - Example: Api.Entity.Text(api.Pov.Entity.MemberId, 1)
- Get Account Type based on current Account POV
  - Example: Api.Account.GetAccountType(api.Pov.Account.MemberId)

When working with formulas and calculations, it is better to work with MemberId versus Member Name.

Api.Pov.Time.MemberId

Api.Pov.Time.MemberId is obtained from the Time Member Id for the current POV being executed during the calculation. The Time.MemberId is stored as an unique integer to represent a single Time member. The uniqueness is determined by the combination of the Year and Period.



H1 = 001

Q1 = 002

M1 = 003

M2 = 004

M3 = 005

Q2 = 006

M4 = 007

M5 = 008

M6 = 009

H2 = 010

Q3 = 011

M7 = 012

M8 = 013

M9 = 014

Q4 = 015

M10 = 016

M11 = 017

M12 = 018

The Time MemberId is constructed like this: 2019003000

The api.Pov.Time.MemberId is used as a property in many functions. Here are some of the most common functions:

- api.Time.GetYearFromId
- api.Time.GetPeriodNumFromId
- api.Time.GetNumDaysInTimePeriod
- api.Time.AddTimePeriods
- api.Time.AddYears

Api.Pov.Time.MemberId Usage

Example using api.Pov.Time.MemberId:

```
Dim timeId As Integer = api.Pov.Time.MemberId
    BRApi.ErrorLog.LogMessage(si, "TimeId = " & timeId)
```

ErrorLog result:

```
TimeId = 2018003000
```

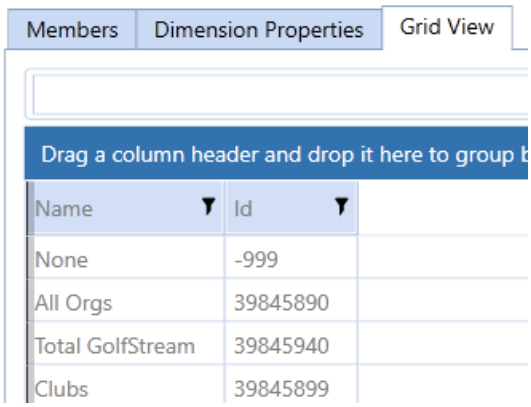
Example using api.Pov.Time.MemberId in a working formula:

```
'Get Current Year as Integer Based on Current POV TimeId
Dim curYear As Integer = api.Time.GetYearFromId(api.Pov.Time.MemberId)
                                 Function ITimeApi.GetYearFromId(Optional timeId As Integer) As Integer

'Execute Formula only if Current Year is Greater Than or Equal to 2018
If curYear >= 2018 Then
    'Only Run for Base Entities and at Local Currency
    If (Not api.Entity.HasChildren() And (api.Cons.IsLocalCurrencyforEntity())) Then
        api.Data.Calculate("A#CashCalc = A#10000")
    End If
End If
```

Api.Pov.Entity.MemberId

Api.Pov.Entity.MemberId is obtained from the Entity Member Id for the current Entity POV being executed during the calculation. The Entity.MemberId is stored as a unique integer to represent a single Entity member. The Entity Member Id is also found using the Grid View in the Entity Dimension Library.



Api.Pov.Entity.MemberId is used as a property in many functions. Here are some of the most common functions:

- Get Local Currency Id for current Entity POV.
    - Example: api.Entity.GetLocalCurrencyId(api.Pov.Entity.MemberId)
- Get Local Currency Cons Member Name for current Entity POV.

- Example:

  api.Entity.GetLocalCurrencyConsMember(api.Pov.Entity.MemberId).Name

- Get value in Text Field for Dimension Members prior to executing formula calculation.
  - Example: api.Entity.Text(api.Pov.Entity.MemberId, 1)
- Get Percent Consolidation for Parent Child Relationship and specific to user localization. Can also determine by Scenario Type and Time.
  - Example: api.Entity.PercentConsolidation(api.Pov.Entity.MemberId, api.Pov.Parent.MemberId, api.Pov.ScenarioTypeId, api.Pov.Time.MemberId).XFToStringForFormula
- Get Percent Ownership for Parent Child Relationship and specific to user localization. Can also determine by Scenario Type and Time.
  - Example: api.Entity.PercentOwnership(api.Pov.Entity.MemberId, api.Pov.Parent.MemberId, api.Pov.ScenarioTypeId, api.Pov.Time.MemberId).XFToStringForFormula

Api.Pov.Entity.MemberId Usage

Example using api.Pov.Entity.MemberId:

```
Dim entityId As Integer = api.Pov.Entity.MemberId
    BRApi.ErrorLog.LogMessage(si, "EntityId = " & entityId)
```

ErrorLog Result:

```
EntityId = 29360129
```

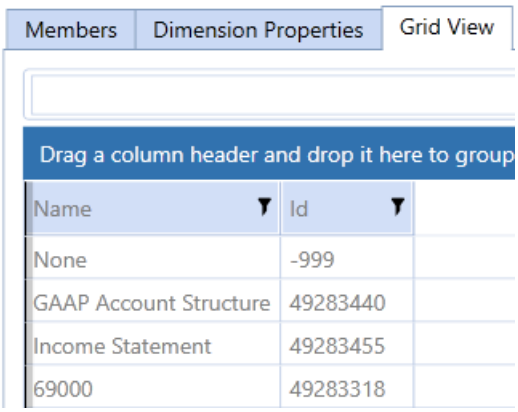
Example using api.Pov.Entity.MemberId in a working formula:

```
'Get Text Value in Entity Text 1 Field for Current Entity POV
Dim entityText As String = api.Entity.Text(api.Pov.Entity.MemberId, 1)

'Only Run For Base Entities And at Local Currency
If (Not api.Entity.HasChildren() And (api.Cons.IsLocalCurrencyforEntity())) Then
    'Execute Formula if Entity has NA in the Entity Text 1 Field
    If entityText.XFEqualsIgnoreCase("NA") Then
        api.Data.Calculate("A#CashCalc = A#10000")
    End If
End If
```

Api.Pov.Account.MemberId

Api.Pov.Account.MemberId is obtained from the Account Member Id for the current Account POV being executed during the calculation. The Account.MemberId is stored as a unique integer to represent a single Account member. The Account Member Id is also found using the Grid View in the Account Dimension Library.



Api.Pov.Account.MemberId is used as a property in many functions. Here are some of the most common functions:

- Get Account Type based on current Account POV
    - Example: api.Account.GetAccountType(api.Pov.Account.MemberId)
- Get value in Text Field for Dimension Members prior to executing formula calculation
    - Example: api.Account.Text(api.Pov.Account.MemberId, 1)

Api.Pov.Account.MemberId Usage

Example using api.Pov.Account.MemberId :

```
Dim accountType As AccountType = api.Account.GetAccountType(api.Pov.Account.MemberId)
    BRApi.ErrorLog.LogMessage(si, "AccountType = " & accountType.ToString)
```

ErrorLog Result:

```
AccountType = Revenue
```

Example using api.Pov.Account.MemberId in a working formula:

```
'Get Account Type of Account and Use Specific FX Rate Type for Specific Account Types. Used in FinanceFunctionType.FXRate or Dynamic Calc
Dim accountType As String = api.Account.GetAccountType(api.Pov.Account.MemberId).ToString
Dim rateType As String = "ClosingRate"

If accountType = "Asset" Then

    Dim rate As Decimal = api.FxRates.GetCalculatedFxRate(rateType, api.Pov.Time.MemberId, args.FxRateArgs.SourceCurrencyId, args.FxRateArgs.DestCurrencyId)
    Return New FxRateResult(rate)

End If
```

Dimension Primary Key - DimPk

DimPk is known as Dimension Primary Key. This is a unique primary key that is assigned to Dimensions when they are created. It is a combination of the DimTypeId and the DimId.

DimPk is commonly used to identify which Dimension should be used when checking for members as base members or descendants in a specific Dimension. DimPk is commonly used in the following functions:

- Get Dimension Primary Key of a Specific Dimension
  - Example: api.Dimensions.GetDim("UD1DimName").DimPk
- Check if it is a Base Member of a Specific Ancestor
  - Example: api.Members.IsBase(dimPk, ancestorMemberId, baseMemberId, dimDisplayOptions)
- Get Base Members of Parent from GetMember
  - Example: api.Members.GetBaseMembers(api.Pov.UD1Dim.DimPk, parent.MemberId, Nothing)

DimPK Usage

Example using DimPK :

```
Dim dimPK As DimPk = api.Dimensions.GetDim("CostCenters").DimPk
    BRapi.ErrorLog.LogMessage(si, "DimPk for CostCenters = " & dimPK.ToString)
```

ErrorLog Result:

```
DimPk for CostCenters = DimTypeId: 9, DimId: 17
```

Example using api.Pov.UD1Dim.DimPk in a working formula:

```
'Retrieve Base Members of Services in UD1 to Use in GetDataCell Loop
Dim parent As Member = api.Members.GetMember(DimType.UD1.Id, "Services")
Dim serviceNames As List(Of Member) = api.Members.GetBaseMembers(api.Pov.UD1Dim.DimPk, parent.MemberId, Nothing)

'Loop through all the Service Base Members
If Not serviceNames Is Nothing Then
    For Each serviceName As Member In serviceNames
        'GetDataCell for All Service Base Members as String and Decimal
        Dim serviceNameCellString As String = ("E#Houston:C#Local:S#Actual:T#2019M1:V#Periodic:A#Dept_Intersection:F#None:O#Forms:I#None:U1#" & serviceName.Name & ":
        Dim serviceNameCell As Decimal = api.Data.GetDataCell(serviceNameCellString).CellAmount
    Next
End If
```

Dimension Type Id

Dimension Type Id is a property of DimPk. The Dimension Type Id is a unique integer Id that is assigned to a Dimension. The DimTypeId is found in the Dim table and the DimTypeId represents each Dimension.

- Entity = 0
- Scenario = 2
- Account = 5
- Flow = 6
- UD1 = 9
- UD2 = 10
- UD3 = 11
- UD4 = 12
- UD5 = 13
- UD6 = 14
- UD7 = 15
- UD8 = 16

The DimTypeId is used in various functions. DimTypeId is most commonly used with the GetMember or GetMemberId functions where the first property in the function is DimTypeId. In this case, GetMember and GetMemberId needs to know which Dimension Id to use for the member the function is looking for.

- Get a specific Member in a specific Dimension
    - Example: api.Members.GetMember(DimType.Account.Id, "AcctMemberName")
- Get Member Id for a specific Member in a specific Dimension
    - Example: api.Members.GetMemberId(DimType.Account.Id, "AcctMemberName")

DimTypeID Usage

Example using DimTypeId :

```
Dim dimTypeId As Integer = DimType.Account.Id
    BRApi.ErrorLog.LogMessage(si, "DimTypeID for Account = " & dimTypeId.ToString)
```

ErrorLog Result:

```
DimTypeID for Account = 5
```

Example using DimType.Account.Id in a working formula:

```
'Get Cash Account Member and Store as a Variable to Pass into Api.Data.Calculate
Dim acctMember As Member = api.Members.GetMember(DimType.Account.Id, "10000")
api.Data.FormulaVariables.SetMemberVariable("variableAccount",acctMember)
api.Data.Calculate("A#CashCalc= A$variableAccount * 100")
```

Data Unit Dimension POV

Stored calculations run based on the Data Unit POV. The Data Unit Dimension consists of Cube, Entity, Parent, Consolidation, Time, and Scenario.

Because stored calculations run off Data Unit Dimensions, these Dimensions are used as part of If Statements to execute calculations on conditions. The Data Unit Dimensions should not be used as destination data buffers, and should not be used on the left hand side of the equation in a api.Data.Calculate formula.

Account related Dimensions such as Account, Flow, and UD's are not available at run-time of the calculations. Therefore, they cannot be used in the If Statements for stored calculations. However, they are available for Dynamic Calculations.

Run for POV and Check Member Names for Data Unit Dimensions Before Executing Calculation:

- If api.Pov.Cube.Name.XFEqualsIgnoreCase("CubeName") Then
- If api.Pov.Entity.Name.XFEqualsIgnoreCase("EntityName") Then
- If api.Pov.Scenario.Name.XFEqualsIgnoreCase("ScenarioName") Then
- If api.Pov.Cons.Name.XFEqualsIgnoreCase("USD") Then

Data Unit Dimension POV Usage

Example using api.Pov.Entity.Name :

```
Dim entityPovName As String = api.Pov.Entity.Name
    BRApi.ErrorLog.LogMessage(si, "Entity Pov Name = " & entityPovName)
```

ErrorLog Result:

```
Entity Pov Name = Houston Heights
```

Example using api.Pov.Entity.Name in a working formula:

```
'Only Run Calculation For Houston Heights
If api.Pov.Entity.Name.XFEqualsIgnoreCase("Houston Heights") Then
    api.Data.Calculate("A#CashCalc = A#10000")
End If
```

```
'Only Run Calculation For Houston Heights
Dim entityPovName As String = api.Pov.Entity.Name

If entityPovName.XFEqualsIgnoreCase("Houston Heights") Then
    api.Data.Calculate("A#CashCalc = A#10000")
End If
```

Time Functions

The following APIs are some of the most common time functions:

- api.Time.GetYearFromId
- api.Time.GetPeriodNumFromId
- api.Time.GetNumDaysInTimePeriod
- api.Time.AddTimePeriods
- api.Time.AddYears

Api.Time.GetYearFromId

This function gets the year from the current POV Time Id. It evaluates the year and then introduces logic to execute the formula.

```
'Get Current Year as Integer Based on Current POV TimeId
Dim curYear As Integer = api.Time.GetYearFromId(api.Pov.Time.MemberId)
                                Function ITimeApi.GetYearFromId(Optional timeId As Integer) As Integer

'Execute Formula only if Current Year is Greater Than or Equal to 2018
If curYear >= 2018 Then
    'Only Run for Base Entities and at Local Currency
    If (Not api.Entity.HasChildren() And (api.Cons.IsLocalCurrencyforEntity())) Then
        api.Data.Calculate("A#CashCalc = A#10000")
    End If
End If
```

Api.Time.GetPeriodNumFromId

This function gets the period number from the current POV Time Id. The period is static and is configured with either months or weeks followed by the period number. For example: M1 – M12 or W1 – W54. It evaluates the period number and then introduces logic to execute the formula.

Api.Time.GetPeriodNumFromId Usage

Example using api.Time.GetPeriodNumFromId :

```
'Get Current Period As Integer Based on Current POV TimeId
Dim curPeriod As Integer = api.Time.GetPeriodNumFromId(api.Pov.Time.MemberId)
    BRApi.ErrorLog.LogMessage(si, "Period Number = " & curPeriod)
```

ErrorLog Result:

```
Period Number = 1
```

Example using api.Time.GetPeriodNumFromId in a working formula:

```
'Get Time Member Id to Get Year and Period
Dim timeId As Integer = api.Pov.Time.MemberId

'Get Current Year As Integer Based On Current POV TimeId
Dim curYear As Integer = api.Time.GetYearFromId(api.Pov.Time.MemberId)

'Get Current Period As Integer Based on Current POV TimeId
Dim curPeriod As Integer = api.Time.GetPeriodNumFromId(api.Pov.Time.MemberId)
                                   Function ITimeApi.GetPeriodNumFromId(Optional timeId As Integer) As Integer

'Execute Formula only if Current Year is Greater Than or Equal to 2018
'AND Current Period Number is Greater Than or Equal to 1
If curYear >= 2018 And curPeriod >= 1 Then
    'Only Run for Base Entities and at Local Currency
    If (Not api.Entity.HasChildren() And (api.Cons.IsLocalCurrencyforEntity())) Then
        api.Data.Calculate("A#CashCalc = A#10000")
    End If
End If
```

Api.Time.GetNumDaysInTimePeriod

This function gets the number of days from the current POV Time Id. The number of days are already programmed depending on the month that is selected. It evaluates the number of days for a period and then introduces logic to execute the formula.

Api.Time.GetNumDaysInTimePeriod Usage

Example using api.Time.GetNumDaysInTimePeriod:

```
'Get Current Number of Days in Time Period
Dim numDays As Integer = api.Time.GetNumDaysInTimePeriod(api.Pov.Time.MemberId)
    BRApi.ErrorLog.LogMessage(si, "Number of Days in Period = " & numDays)
```

ErrorLog Result:

```
Number of Days in Period = 31
```

Example using api.Time.GetNumDaysInTimePeriod in a working formula:

```
'Get Time Member Id to Get Year and Period
Dim timeId As Integer = api.Pov.Time.MemberId

'Get Current Year As Integer Based On Current POV TimeId
Dim curYear As Integer = api.Time.GetYearFromId(api.Pov.Time.MemberId)

'Get Current Period As Integer Based on Current POV TimeId
Dim curPeriod As Integer = api.Time.GetPeriodNumFromId(api.Pov.Time.MemberId)

'Get Current Number of Days in Time Period
Dim numDays As Integer = api.Time.GetNumDaysInTimePeriod(api.Pov.Time.MemberId)
                                 Function ITimeApi.GetNumDaysInTimePeriod(Optional timeId As Integer) As Integer

'Execute Formula only if Current Year is Greater Than or Equal to 2018
'AND Current Period Number is Greater Than or Equal to 1
'AND Number of Days is Greater Than or Equal to 30 Days
If (curYear >= 2018 And curPeriod >= 1 And numDays >= 30) Then
    'Only Run for Base Entities and at Local Currency
    If (Not api.Entity.HasChildren() And (api.Cons.IsLocalCurrencyforEntity())) Then
        api.Data.Calculate("A#CashCalc = A#10000")
    End If
End If
```

Api.Time.AddTimePeriods

This function adds time periods to the current POV Time Id. It passes that data to different functions like GetPeriodNumFromId and then introduces logic to execute the formula.

Api.Time.AddTimePeriods Usage

Example using api.Time.AddTimePeriods:

```
'Get Current Time Member Id, Add 2 Periods, and Ok to Span Years
'Example:  Current Time Member Id = 2018003000.  Add 2 Periods, Then Member Id = 2018005000
Dim addTime As Integer = api.Time.AddTimePeriods(api.Pov.Time.MemberId, 2, True)
    BRApi.ErrorLog.LogMessage(si, "Add Time Periods = " & addTime)
```

ErrorLog Result:

```
Add Time Periods = 2018005000
```

Example using api.Time.AddTimePeriods in a working formula:

```
'Get Time Member Id to Get Year and Period
Dim timeId As Integer = api.Pov.Time.MemberId

'Get Current Time Member Id, Add 2 Periods, and Ok to Span Years
'Example:  Current Time Member Id = 2018003000.  Add 2 Periods, Then Member Id = 2018005000
Dim addTime As Integer = api.Time.AddTimePeriods(api.Pov.Time.MemberId, 2, True)
                                 Function ITimeApi.AddTimePeriods(timeId As Integer, numTimePeriodsToAdd As Integer, okToSpanYears As Boolean) As Integer

'Get Period from Add Time Period and Pass in GetPeriodNumFromId
Dim periodNum As Integer = api.Time.GetPeriodNumFromId(addTime)

'Execute Formula Only in Mar Period
If periodNum = 3 Then
    'Only Run for Base Entities and at Local Currency
    If (Not api.Entity.HasChildren() And (api.Cons.IsLocalCurrencyforEntity())) Then
        api.Data.Calculate("A#CashCalc = A#10000")
    End If
End If
```

Api.Time.AddYears

This function adds years to the current POV Time Id. It passes that data to different functions like GetYearFromId or GetPeriodNumFromId and then introduces logic to execute the formula.

Api.Time.AddYears Usage

Example using api.Time.AddYears:

```
'Get Current Time Member Id and Add 2 Years
'Example:  Current Time Member Id = 2018003000.  Add 2 Years, Then Member Id = 2020003000
Dim addYears As Integer = api.Time.AddYears(api.Pov.Time.MemberId, 2)
    BRApi.ErrorLog.LogMessage(si, "Added 2 Years To Current Time POV = " & addYears)
```

ErrorLog Result:

```
Added 2 Years To Current Time POV = 2020003000
```

Example using api.Time.AddYears in a working formula:

```
'Get Current Time Member Id and Add 2 Years
'Example:  Current Time Member Id = 2018003000.  Add 2 Years, Then Member Id = 2020003000
Dim addYears As Integer = api.Time.AddYears(api.Pov.Time.MemberId, 2)
```

Function ITimeApi.AddYears(timeId As Integer, numYearsToAdd As Integer) As Integer

```
'Get Year from addYears and Pass it in for GetYearFromId function
Dim futureYear As Integer = api.Time.GetYearFromId(addYears)

'Execute Formula Only in Year 2020
If futureYear = 2020 Then
    'Only Run for Base Entities and at Local Currency
    If (Not api.Entity.HasChildren() And (api.Cons.IsLocalCurrencyforEntity())) Then
        api.Data.Calculate("A#CashCalc = A#10000")
    End If
End If
```

Using Member Functions for Calculations

Calculation Member functions are commonly used through the Finance Api's for accessing general information for any specified Member within a dimension. The Member functions allow a rule writer to identify members, get member information, and identify base and parent members to execute within Member Formulas and Business Rules.

The following are some of the most common Member functions for calculations:

- GetMember
- GetMemberID
- GetBaseMembers

GetMember

This function gets a specific dimension member. It is used for different functions like api.Data.FormulaVariables, GetBaseMembers function, custom member lists, and when working with Member Id within data buffers for processes like custom consolidation.

GetMember Usage

Example using GetMember:

```
Dim getMember As Member = api.Members.GetMember(DimType.Account.Id, "10000")
    BRapi.ErrorLog.LogMessage(si, "Member Property Info = " & getMember.ToString)
```

ErrorLog Result:

```
Member Property Info = DimTypeId: 5, MemberId: 39845888,
Name: 10000, Description: Petty Cash, DimId: 38
```

Example using GetMember in a working formula:

```
'Get Cash Account Member and Store as a Variable to Pass into Api.Data.Calculate
Dim acctMember As Member = api.Members.GetMember(DimType.Account.Id, "10000")
api.Data.FormulaVariables.SetMemberVariable("variableAccount",acctMember)
api.Data.Calculate("A#CashCalc= A$variableAccount * 100")
```

GetMemberId

This function gets a specific dimension member Id. This technique is commonly used when working with source Data Buffers where the cells for a specific member Id need to be changed.

GetMemberID Usage

Example using GetMemberId:

```
Dim getMemberId As Integer = api.Members.GetMemberId(DimType.Account.Id, "10000")
    BRapi.ErrorLog.LogMessage(si, "Member Id for 10000 = " & getMemberId.ToString)
```

ErrorLog Result:

```
Member Id for 10000 = 39845888
```

Example using GetMemberId in a working formula:

```
'Get Member Id for CashCalc Account
Dim cashCalcId As Integer = api.Members.GetMemberId(DimType.Account.Id, "CashCalc")

'Create a data buffer with the cells from S#Actual:A#10000 and copy the cells to S#ActualCopy:A#CashCalc
Dim destinationInfo As ExpressionDestinationInfo = api.Data.GetExpressionDestinationInfo("S#ActualCopy")
Dim sourceDataBuffer As DataBuffer = api.Data.GetDataBuffer(DataApiScriptMethodType.Calculate, "S#Actual:A#10000", destinationInfo)
'Check that the source Data Buffer exists
If Not sourceDataBuffer Is Nothing Then
    'Create a new result data buffer for data cells
    Dim resultDataBuffer As DataBuffer = New DataBuffer()
    'Loop through source data cells from the source data buffer
    For Each sourceCell As DataBufferCell In sourceDataBuffer.DataBufferCells.Values
        'Only get source cells that have data
        If (Not sourceCell.CellStatus.IsNoData) Then

            'Copy the cell from 10000 - Petty Cash to CashCalc Account in ActualCopy Scenario
            'The source data buffer contains source data cells with 10000 - Petty Cash AccountId
            'Change the source Account Id for 10000 - Petty Cash with the CashCalc Account Id
            Dim resultCell As New DataBufferCell(sourceCell)
            resultCell.DataBufferCellPk.AccountId = cashCalcId
            resultDataBuffer.SetCell(api.DbConnApp.SI, resultCell)

        End If
    Next
    'Set Destination Data Buffer with new Data Buffer with new cells including the CashCalc AccountId
    api.Data.SetDataBuffer(resultDataBuffer, destinationInfo)

End If
```

GetBaseMembers

This function gets base members from a specific parent member. It is commonly used when working with Member Lists as part of FinanceFunctionType.MemberList, or to get base members to loop through specific dimensions for api.Data.GetDataCell.

GetBaseMembers Usage

Example using GetBaseMembers:

```
'Retrieve Base Members of Services in UD1 to Use in GetDataCell Loop
Dim parent As Member = api.Members.GetMember(DimType.UD1.Id, "Services")
Dim serviceNames As List(Of Member) = api.Members.GetBaseMembers(api.Pov.UD1Dim.DimPk, parent.MemberId, Nothing)

'Loop through all the Service Base Members
If Not serviceNames Is Nothing Then
    For Each serviceName As Member In serviceNames
        BRapi.ErrorLog.LogMessage(si, "List of Base Members = " & serviceName.ToString)
```

ErrorLog Result:

List of Base Members = DimTypeId: 9, MemberId: 17825805, Name: GroundsMaint, Description: Ground Maintenance, DimId: 17
List of Base Members = DimTypeId: 9, MemberId: 17825797, Name: EquipMaint, Description: Equipment Maintenance, DimId: 17
List of Base Members = DimTypeId: 9, MemberId: 17825804, Name: GolfPros, Description: Golf Pro Staff, DimId: 17
List of Base Members = DimTypeId: 9, MemberId: 17825814, Name: ProShop, Description: ProShop Retail, DimId: 17

Example using GetBaseMembers in a working formula:

```
'Retrieve Base Members of Services in UD1 to Use in GetDataCell Loop
Dim parent As Member = api.Members.GetMember(DimType.UD1.Id, "Services")
Dim serviceNames As List(Of Member) = api.Members.GetBaseMembers(api.Pov.UD1Dim.DimPk, parent.MemberId, Nothing)

'Loop through all the Service Base Members
If Not serviceNames Is Nothing Then
    For Each serviceName As Member In serviceNames
'       'GetDataCell for All Service Base Members as String, Decimal, and for International Rule Writing
        Dim serviceNameCellString As String = ("E#Houston:C#Local:S#Actual:T#2019M1:V#Periodic:A#Dept_Intersection:F#None:O#Forms:I#None:U1#" & serviceName.Name & ":U2#UD1Default:
        Dim serviceNameCell As Decimal = api.Data.GetDataCell(serviceNameCellString).CellAmount
        Dim serviceNameCellText As String = serviceNameCell.ToString("G", CultureInfo.InvariantCulture)

        'Check cell amount for intersection and then introduce logic based on cell amount
        'Use Data Buffer logic or api.Data.Calculate with SetDataBufferVariable function when in loop
    Next
End If
```

Writing Stored Calculations

When writing a Member Formula or a Business Rule for a Stored Calculation, the new calculated numbers store data for that Cube, Entity, Parent, Cons, Scenario, and Time combination. For example, a Data Unit.

Return is never seen in a Member Formula for Formula Pass. Instead of being returned, many numbers are calculated and stored. When running a Calculation, Translation, or Consolidation, it calls the Member Formula once for an entire Data Unit. OneStream does not tell with which Account, Flow, or User Defined the numbers are being saved.

Initially, this may be confusing because Member Formulas are often written in an account's Formula property, and administrators believe OneStream 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 OneStream calls that formula, it is currently calculating a Data Unit and will initialize the API engine with only the Data Unit Dimensions.

Basic stored formulas are commonly used via the Api.Data.Calculate api function. Api.Data.Calculate is used in three different ways:

- Api.Data.Calculate using Formula as String, Overload Functions, Eval Function, and IsDurableCalculatedData

```
api.Data.Calculate()
    ▲ 1 of 3 ▼  Sub DataApi.Calculate(formula As String, Optional accountFilter As String, Optional flowFilter As String, Optional originFilter As String, Optional icFilter As
    String, Optional ud1Filter As String, Optional ud2Filter As String, Optional ud3Filter As String, Optional ud4Filter As String, Optional ud5Filter As String, Optional
    ud6Filter As String, Optional ud7Filter As String, Optional ud8Filter As String, Optional onEvalDataBuffer As EvalDataBufferDelegate, Optional userState As Object,
    Optional isDurableCalculatedData As Boolean)
```

- Api.Data.Calculate using Formula as String and IsDurableCalculatedData

```
api.Data.Calculate()
    ▲ 2 of 3 ▼  Sub DataApi.Calculate(formula As String, isDurableCalculatedData As Boolean)
```

- Api.Data.Calculate using Formula as String and Eval Function

```
api.Data.Calculate()
    ▲ 3 of 3 ▼  Sub DataApi.Calculate(formula As String, onEvalDataBuffer As EvalDataBufferDelegate, Optional userState As Object)
```

Overload Function

The most common function is Api.Data.Calculate, which sets the value of one or more dimension values (left side of formula) equal to another (right side). Final arguments (optional) are added to the formula for Overload Functions, Evals, and Durable Data.

The Api.Data.Calculate function must abide by the data explosion rules, which means that the left side and the right side of the formulas are balanced with the same dimension values on both sides. If a Member is specified for a Dimension anywhere on the right side of the equation, you must explicitly specify something for that Dimension on the left side of the equation.

This variation of the Api.Data.Calculate provides Member Filters (all optional) which can be used to filter the results before saving them to the target or destination. This function is the most powerful of the Api.Data.Calculate functions as it allows you to filter intersections. In addition, the Eval function adds the ability to filter down the number of individual data cells processed by data cell attributes such as CellAmount or CellStatus.

This function is commonly used to filter the source data buffer by base members of an Account related dimension. For example, A#Sales may be the source data buffer but the need for all products is not required for the calculation. Instead, A#Sales may need to be calculated by the base members of Clubs. By using Clubs.Base for A#Sales, the A#Sales data buffer has been reduced to only include Clubs.Base.

Api.Data.Calculate Usage

Example using Overload Function in a working formula:

```
'Add a Formula and use API.Data.Calculate with a filter on UD2 (product) so that
'A#[ClubsSalesCalc] = the A#60000 account (Operating Sales) For just the base products under UD2#Clubs
'Hint: api.Data.Calculate("A#[ClubsSalesCalc] = A#60000",,,,,,"UD2 MEMBER FILTER GOES HERE")
'Formula will run at the base and parent levels

If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyforEntity())) Then
    api.Data.Calculate("A#ClubsSalesCalc = A#60000",,,,,,"U2#Clubs.Base")
End If
```

```
▲ 1 of 3 ▼  Sub DataApi.Calculate(formula As String, Optional accountFilter As String, Optional flowFilter As String, Optional originFilter As String, Optional icFilter As String,
            Optional ud1Filter As String, Optional ud2Filter As String, Optional ud3Filter As String, Optional ud4Filter As String, Optional ud5Filter As String, Optional
            ud6Filter As String, Optional ud7Filter As String, Optional ud8Filter As String, Optional onEvalDataBuffer As EvalDataBufferDelegate, Optional userState As Object,
            Optional isDurableCalculatedData As Boolean)
```

IsDurableCalculatedData

This variation of Api.Data.Calculate lets you define whether data is durable or not. Durable data is not cleared automatically when a Data Unit is re-calculated. It can only be cleared by calling api.Data.ClearCalculatedData with the clearDurableCalculatedData Boolean property set to True. As part of the standard Calculation sequence that runs during a Calculate or Consolidate, Durable data will be ignored from processing the clear, unless the clear is specifically defined within the Business Rule or Member Formula.

The most common reason to set the IsDurableCalculatedData to True is for seeding purposes. As part of the first seeding, the goal may be to seed from one Scenario to another just once and never seed it again. In this case, the seeded data should not be cleared at any point during the Calculate or Consolidate process. This technique is commonly used in Budget or Forecast processes where you are executing the seeding through a Dashboard. The formula may be applied as a FinanceFunctionType.CustomCalculate or a FinanceFunctionType.Calculate in a Business Rule.

IsCurableCalculatedData Usage

Example using IsDurableCalculatedData in a working formula:

```
Case Is = FinanceFunctionType.CustomCalculate

    'Define a unique Function Name that will be passed into Custom Calculate process
    If args.CustomCalculateArgs.FunctionName.XFEqualsIgnoreCase("CopyScenario") Then

        'Declare variables that will be passed into api.Data.Calculate.
        'Selected values from parameters will be passed into api.Data.Calculate formula
        Dim selectedTime As String = args.CustomCalculateArgs.NameValuePairs("SelectedTime")
        Dim sourceScenario As String = args.CustomCalculateArgs.NameValuePairs("SourceScenario")
        Dim targetScenario As String = args.CustomCalculateArgs.NameValuePairs("TargetScenario")

        'Only run for base entities and local currency
        If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyforEntity())) Then
            'Using api.Data.Calculate function with formula and IsDurableCalculatedData set to TRUE As Boolean.
            'Can use filters as well. Use RemoveNoData function or EVAL to eliminate processing data cells with NODATA
            api.Data.Calculate("S#[" & targetScenario & "]:T#[" & selectedTime & "] = RemoveNoData(S#[" & sourceScenario & "]:T#[" & selectedTime & "])", True)
        End If

    End If
```

Eval Function

Eval has an advanced capability that lets you get at the individual Data Cells in any Data Unit created while processing an api.Data.Calculate script. It allows Eval() to be wrapped around a subset of the formula's math in order to evaluate the Data Buffer that was just created by running that math.

Prior to the 5.0 version and the introduction of the RemoveNoData function, Eval was commonly used to evaluate individual data cells in a source data buffer to process based on cell amount or cell status. Evaluating the number of No Data Cells for a Data Unit is an important factor for performance and calculation efficiencies.

Eval was initially an important function to evaluate individual data cells but it has been replaced with newer techniques such as GetDataBuffer and GetDataBufferUsingFormula, and looping through cells within the data buffer, as well as the Remove functions.

Eval Function Usage

Example using Eval in a working formula:

```
Formula Footer...
Helper Functions Header...
Private Sub OnEvalDataBuffer (ByVal api As FinanceRulesApi, ByVal evalName As String, ByVal eventArgs As EvalDataBufferEventArgs)
    Try

        'Start with and empty list of result cells.
        'Good practice - Clear out DataBufferResults before executing

        eventArgs.DataBufferResult.DataBufferCells.Clear()

        'Loop over the source cells and assign a bonus % based on level
        For Each sourceCell As DataBufferCell In eventArgs.DataBuffer1.DataBufferCells.Values

            'Only get source cells that have data and are greater than or equal to 0
            If (Not sourceCell.CellStatus.IsNoData) And (sourceCell.CellAmount >= 0.00) Then
                'Create new data buffer cells with the filtered data cells
                Dim resultCell As New DataBufferCell(sourceCell)
                    'Condition if Level is greater than or equal to 1 and less than 2
                    If (sourceCell.CellAmount >= 1.00) And (sourceCell.CellAmount < 2.00) Then
                        'Return 10% to multiply by Salary or A#50200
                        resultCell.CellAmount = 0.10

                        'Condition if Level is greater than or equal to 2 and less than 3
                    Else If (sourceCell.CellAmount >= 2.00) And (sourceCell.CellAmount < 3.00) Then
                        'Return 20% to multiply by Salary or A#50200
                        resultCell.CellAmount = 0.20

                        'Condition if Level is greater than or equal to 3 and less than 4
                    Else If (sourceCell.CellAmount >= 3.00) And (sourceCell.CellAmount < 4.00) Then
                        'Return 30% to multiply by Salary or A#50200
                        resultCell.CellAmount = 0.30

                    Else 'All other conditions
                        'Return 5% to multiply by Salary or A#50200
                        resultCell.CellAmount = 0.05

                    End If
                    'Set the final results of the data cells for the Data Buffer
                    eventArgs.DataBufferResult.SetCell(api.SI, resultcell, False)

            End If
        Next

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

Helper Functions Footer...
```

Summary

The Api.Data.Calculate is the easiest and simplest way to write a formula as a Member Formula or a Business Rule. The construction of an Api.Data.Calculate formula must be balanced on each side of the formula with the appropriate dimensions to prevent data explosion. There are three different ways to use the Api.Data.Calculate function: Formula with Overload, Formula with IsDurableCalculatedData, and Formula with Eval.

From a performance perspective:

1. Never use the Api.Data.Calculate in a loop when using variables.
2. Use Remove functions whenever possible especially for sparse data models with lots of NODATA cells.
3. GetDataBuffer and GetDataBufferUsingFormula may have a better performance impact. Try replacing Api.Data.Calculate when doing math with GetDataBuffer math. In some cases, performance is better by using GetDataBuffer functions in place of Api.Data.Calculate.

Remove Functions

Remove Functions were introduced in the 5.0 release. They replaced the reasons to use the Eval function. The basic need of the Eval function was to evaluate the individual data cells within a source data buffer to apply logic for processing. In many cases, OneStream did not want to process data cells in source data buffers that had a Cell Status of NODATA or Cell Amount = 0. With the 5.0 release, functions do that without the need for writing additional logic.

The **RemoveNoData** and **RemoveZeros** functions provide the ability to not process individual data cells within a source data buffer. They wrap the Remove() around a subset of the formula to prevent processing of individual data cells from within a source data buffer. Remove functions are used in Member Formulas or Business Rules.

Remove functions are used for performance reasons. Data Units may contain a great amount of NODATA data cells or 0 value data cells. These cells could be needlessly processed during calculation execution if these functions are not used in a Api.Data.Calculate formula.

RemoveZeros

RemoveZeros is used to remove data cells with a cell amount of 0 from the source data buffer. In addition, this function removes data cells with a cell status of NODATA from the source data buffer. It is important to evaluate if the 0s are needed for the Api.Data.Calculate formula during calculation execution.

RemoveNoData

RemoveNoData removes data cells with a cell status of NODATA ONLY from the source data buffer. Unlike the RemoveZeros function, this function does not remove data cells with a cell amount of 0.

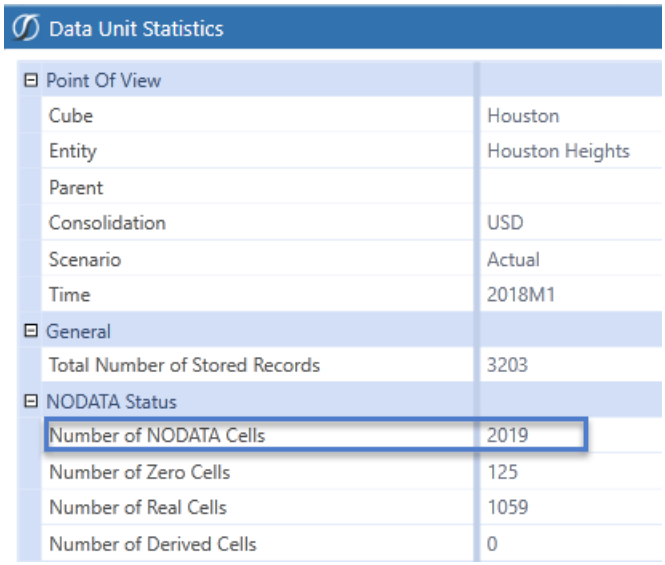
NODATA cells and 0 cells can be found using the following methods:

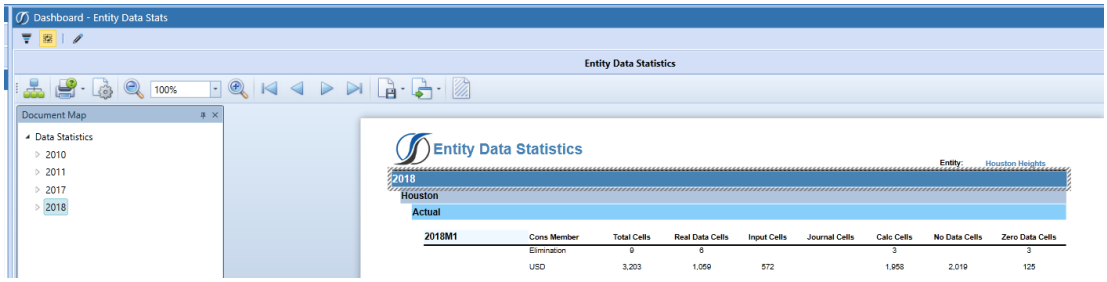
1. Review the Data Unit Statistics when you right-click on a cell in Cube View.
2. Review the Application Analysis Dashboard and check the Entity Data Statistics Report.

This is based on the Data Unit and Entity Data Statistics. There may be many Member Formulas and Business Rules that are driving data creation. Therefore, all formulas would need to be evaluated to determine whether these Remove functions are used. The higher the percentage ratio of NODATA cells to Total Number of Stored Records, the more important it is to use these Remove functions.

Example = 3,203 Stored Records with 2,019 of those Stored Records as NODATA cells. Nearly 65% of the Data Unit has NODATA cells to process which causes extra calculation time.

The Review functions can be found in Key Functions under Snippets.





Remove Functions Usage

Example using RemoveZeros in a working formula:

```
'Declare variable To Get period number From the current time period
Dim curMonth As Integer = api.Time.GetPeriodNumFromId(api.Pov.Time.MemberId)
'Run for Entity Base Members Only
If (Not api.Entity.HasChildren()) Then
    'Check to see if current month is M1.
    'If so, pull Ending Balances From M12 prior year.  We are using F#None for this exercise
    'If M2 - M12, pull Ending Balances or F#None from prior period in current year
    'Only run the calculation for Balance Sheet base accounts
    'Remove data cells with cell amount of 0 and cell status of NoData
    If curMonth = 1 Then
        api.Data.Calculate("F#BegBalCalcRemove= RemoveZeros(F#None:T#PovPriorYearM12)","A#[Balance Sheet].Base")
    Else
        api.Data.Calculate("F#BegBalCalcRemove = RemoveZeros(F#BegBalCalc:T#PovPrior1)","A#[Balance Sheet].Base")
    End If
End If
```

Example using RemoveNoData in a working formula:

```
'Declare variable to get period number from the current time period
Dim curMonth As Integer = api.Time.GetPeriodNumFromId(api.Pov.Time.MemberId)
'Run for Entity Base Members Only
If (Not api.Entity.HasChildren()) Then
    'Check to see if current month is M1.
    'If so, pull Ending Balances From M12 prior year.  We are using F#None for this exercise
    'If M2 - M12, pull Ending Balances or F#None from prior period in current year
    'Only run the calculation for Balance Sheet base accounts
    'Remove data cells with cell status of NoData ONLY
    If curMonth = 1 Then
        api.Data.Calculate("F#BegBalCalcRemove= RemoveNoData(F#None:T#PovPriorYearM12)","A#[Balance Sheet].Base")
    Else
        api.Data.Calculate("F#BegBalCalcRemove = RemoveNoData(F#BegBalCalc:T#PovPrior1)","A#[Balance Sheet].Base")
    End If
End If
```

GetDataBuffer Functions

A Member Script may not be defined for the Api.Data.Calculate function because multiple Data Cells, which seem completely unrelated to each other, are being processed and none of the Dimension Members are constant. For those situations, use the GetDataBuffer and SetDataBuffer functions.

GetDataBuffer and SetDataBuffer are more fundamental than using an Eval function. They allow you to read numbers using a Member Script, process or modify each cell in the result, and then save the changes. Common GetDataBuffer functions include:

- GetDataBuffer
- GetDataBufferForCustomShareCalculation
- GetDataBufferForCustomElimCalculation
- GetDataBufferUsingFormula
- SetDataBuffer

When using api.Data.Calculate functions, it is important to know which Member a formula is attached to. For example, if the formula starts with Api.Data.Calculate(“A#Sales1 = …”), put the formula in the Sales1 account Member’s Formula setting.

However, when using GetDataBuffer functions, the formula may not be writing to a specific Member. Every Data Cell saved is possibly written to a different dimension member. In this case, the logic can be developed in a Business Rule and could be created as a Sub routine to execute throughout Finance Business Rules.

GetDataBuffer Function

GetDataBuffer retrieves a Data Unit’s values during a particular consolidation, calculation, or translation. When using GetDataBuffer, this is equivalent to the source data buffer or to the right side of the equation for Api.Data.Calculate. Depending on which GetDataBuffer function you are using, three or four properties can be used.

For the basic GetDataBuffer, three properties are used:

- ScriptMethodType As DataApiScriptMethodType
- SourceDataBufferScript As String
- ExpressionDestinationInfo As ExpressionDestinationInfo

The scriptMethodType typically uses the Calculate option for DataApiScriptMethodType.

The sourceDataBufferScript is equivalent to the right side of the equation for the Api.Data.Calculate.

The expressionDestinationInfo is equivalent to the left side of the equation for the Api.Data.Calculate. Frequently, this gets manipulated using the Dimension Id, passing in the Dimension Member Id for the data buffer primary key.

The GetDataBuffer can be used in various ways, and is not limited to the following:

1. Use Data Buffers to perform Data Buffer math. In some cases, this can perform better than an Api.Data.Calculate.
2. Use GetDataBuffer in place of Api.Data.Calculate to use in Sub routines which execute code and instructions, are stored in memory, and are used within Functions throughout Finance Business Rules.

GetDataBuffer Usage

Example using GetDataBuffer with Data Buffer Math in a working formula:

```
'Alternate way to api.Data.Calculate("A#DataBufferMath:UD2#None = A#60999:UD2#Top - A#54500:UD2#Top").  May have better performance impact.

'Run only for Local Currency and Base Entities
If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyforEntity())) Then

    'Declare Variable for Destination Buffer
    Dim destinationInfo As ExpressionDestinationInfo = api.Data.GetExpressionDestinationInfo("A#DataBufferMath:UD2#None")

    'Get Source Data Buffer for Net Sales
    Dim netSales As DataBuffer = api.Data.GetDataBuffer(DataApiScriptMethodType.Calculate,"RemoveNoData(A#60999:UD2#Top)", destinationInfo)

    'Get Source Data Buffer for Operating Expenses
    Dim operatingExpenses As DataBuffer = api.Data.GetDataBuffer(DataApiScriptMethodType.Calculate,"RemoveNoData(A#54500:UD2#Top)", destinationInfo)

    'Create New Data Buffer With the End Result of Net Sales - Operating Expenses
    Dim dataBufferExample As DataBuffer = (netSales - operatingExpenses)

    'Set the Destination Data Buffer
    api.Data.SetDataBuffer(dataBufferExample, destinationInfo)

End If
```

Example using GetDataBuffer and SetDataBuffer in Business Rule Using Sub Routine in a working formula:

```
Case Is = FinanceFunctionType.Calculate

    'Execute Sub Routine in the Function to Return Results
    Me.CalculateBonusUsingGetDataBuffer(api)
```

```
Private Sub CalculateBonusUsingGetDataBuffer(ByVal api As FinanceRulesApi)
    Try
        'Define Destination Data Buffer or left side of the equation
        'Will copy to A#Bonus while processing the data buffer in memory
        Dim destinationInfo As ExpressionDestinationInfo = api.Data.GetExpressionDestinationInfo("")
        'Read the numbers for A#Salary into a source Data Buffer
        Dim sourceDataBuffer As DataBuffer = api.Data.GetDataBuffer(DataApiScriptMethodType.Calculate, "A#Salary", destinationInfo)

        'Check to make sure the source Data Buffer exists
        If Not sourceDataBuffer Is Nothing Then
            'Create a new data buffer for the result cells
            Dim resultDataBuffer As DataBuffer = New DataBuffer()

            'Loop over the source cells in the source Data Buffer
            For Each sourceCell As DataBufferCell In sourceDataBuffer.DataBufferCells.Values
                'Only process cells that have data and cell amount that is greater than 0
                If ((Not sourceCell.CellStatus.IsNoData) And (sourceCell.CellAmount > 0.00)) Then
                    'Create new data buffer cells from the filtered source cells from source Data Buffer
                    Dim resultCell As New DataBufferCell(sourceCell)

                    'Define A#Bonus as the target account to copy to
                    'Multiply data cell amounts by 5%
                    'Set the manipulated data cells to the data buffer
                    resultCell.DataBufferCellPk.AccountId = api.Members.GetMemberId(DimType.Account.Id, "Bonus")
                    resultCell.CellAmount = sourceCell.CellAmount * 0.05
                    resultDataBuffer.SetCell(api.SI, resultCell)

                End If
            Next

            'Save the results to the destination data buffer
            api.Data.SetDataBuffer(resultDataBuffer, destinationInfo)

        End If

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

Unbalanced Math Functions

Unbalanced Math Functions

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. The four Unbalanced Math functions are:

- AddUnbalanced
  - Example: api.Data.Calculate("A#TargetAccount = AddUnbalanced (A#OperatingSales, A#DriverAccount:U2#Global, U2#Global)")
- SubtractUnbalanced
  - Example: api.Data.Calculate("A#TargetAccount = SubtractUnbalanced (A#OperatingSales, A#DriverAccount:U2#Global, U2#Global)")
- MultiplyUnbalanced
  - Example: api.Data.Calculate("A#TargetAccount =MultiplyUnbalanced (A#OperatingSales, A#DriverAccount:U2#Global, U2#Global)")
- DivideUnbalanced
  - Example: api.Data.Calculate("A#TargetAccount =DivideUnbalanced (A#OperatingSales, A#DriverAccount:U2#Global, U2#Global)")

When using Unbalanced Math functions, 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 important 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).

Often, these functions would be used when one source data buffer is doing math with a specific data cell intersection. This could be a rate, driver, or some data cell input.

Unbalanced Math Functions Usage

Example using MultiplyUnbalanced in a working formula:

```
'Calculate Salary (A#50200) times Bonus Percent to create Bonus number.  Use MultiplyUnbalanced formula to calculate.
'Use a Technique to Not Process No Data Cells and 0 Data Cells for Salary account
'1st property is the data buffer with the least dimensions and matches dimensionality of destination data buffer. Follow Data Explosion rules
'2nd Property is the data buffer with the most dimensions
'3rd Property is the list of account related dimensions that make it unbalanced

'Run for only Base Entities and Local Currency
If ((Not api.Entity.HasChildren()) And (api.Cons.IsLocalCurrencyforEntity())) Then
    api.Data.Calculate("A#BonusUnbalanced = MultiplyUnbalanced(RemoveZeros(A#50200), A#BonusPercent:F#None:O#Forms:I#None:U2#None:U3#None:U4#None:U5#None:U6#None:U7#None:U8#None, F#None:O#Forms:I#None:U2#None:
End If
```

GetDataBufferUsingFormula Function

The GetDataBufferUsingFormula function uses an entire math expression to calculate a final data buffer. GetDataBufferUsingFormula can perform the same data buffer math as Api.Data.Calculate, but the result is assigned to a variable, where Api.Data.Calculate actually saves the calculated data.

GetDataBufferUsingFormula calculates multiple source data buffers first. Then, the result of the math is stored in memory using a Formula Variable. Finally, the Formula Variable is used anywhere within the Member Formula or Business Rule. This function is commonly used during rule writing for Planning Business Rules using MultiplyUnbalanced, DivideUnbalanced, Trailing functions such as trailing 12 months, and Allocations.

When using GetDataBufferUsingFormula, FilterMembers and RemoveMembers are used in conjunction to shrink down dimensional members in the source Data Buffer.

FilterMembers

FilterMembers 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.

GetDataBufferUsingFormula Usage

Example using GetDataBufferUsingFormula in a working formula:

```
'Alternate way to api.Data.Calculate("A#DataBufferMathUsingFormula:UD2#None = A#60999:UD2#Top - A#54500:UD2#Top").  May have better performance impact using
'GetDataBufferUsingFormula

    'Standard GetDataBufferUsingFormula formula
    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#60999:UD2#Top) - RemoveNoData(A#54500:UD2#Top)")
        'Set Data Buffer Variable to pass into api.Data.Calculate formula.  Can be used for multiple instances of api.Data.Calculate
        'Create a unique name to name the Data Buffer as a Formula Variable
        api.Data.FormulaVariables.SetDataBufferVariable("dataBufferExample", dataBufferExample, False)
        'Pass variable into api.Data.Calculate using a $
        'Can pass this variable to other api.Data.Calculate, GetDataBufferUsingFormula, or Sub routines
        api.Data.Calculate("A#DataBufferMathUsingFormula:UD2#None = $dataBufferExample")

    End If
```

Example using GetDataBufferUsingFormula with FilterMembers and MultipleUnbalanced in a working formula:

```
'Use Data Buffer Using Formula to filter specific members
'1st argument inside () is the starting data buffer
'2nd argument is the filter based on the starting data buffer
'Can continue to add filters separated by a comma
Dim salesExp As DataBuffer = api.Data.GetDataBufferUsingFormula("RemoveZeros(FilterMembers(A#All,A#TotalExp.Base))")

'Set Data Buffer Variable to pass salesExp to any other formula
api.Data.FormulaVariables.SetDataBufferVariable("salesExp", salesExp, False)

'Use MultiplyUnbalanced to multiply all Expense Accounts by a specific data cell intersection and divide by 12
'1st argument is Formula Variable multiplied by 2nd argument which is an individual data cell intersection
'3rd argument is the dimensions that make it unbalanced
Dim result As DataBuffer = api.Data.GetDataBufferUsingFormula("MultiplyUnbalanced($salesExp, (E#Global:V#YTD:A#RateAccount:C#USD:F#None:O#BeforeAdj:I#None:U1#None:U2#None:U3#None:U4#None/12), E#Global:V#YTD:C#USD:F#None:O#BeforeAdj

'Set Data Buffer Variable to pass result to any other member formula
api.Data.FormulaVariables.SetDataBufferVariable("result", result, True)

'Calculate using Data Buffer Variable.  Can do additional math inside api.Data.Calculate
api.Data.Calculate("V#Periodic = $result")
```