API Overview Guide

9.2.0 Release

Table of Contents

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.

To maintain optimal performance and ensure security, use public and documented APIs only. Internal APIs are not intended for public general use and may be changed or removed without notice. Support cannot provide assistance for issues resulting from the uses of nonpublic features.

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 .Net Core. 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 .NET Core. C# and VB.NET have very different syntax elements, but Microsoft developed these languages simultaneously as part of a common .NET Core 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 Core 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 Core 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.

Developer Fundamentals

VB.Net and C#

The OneStream platform is based entirely on .Net Core 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.

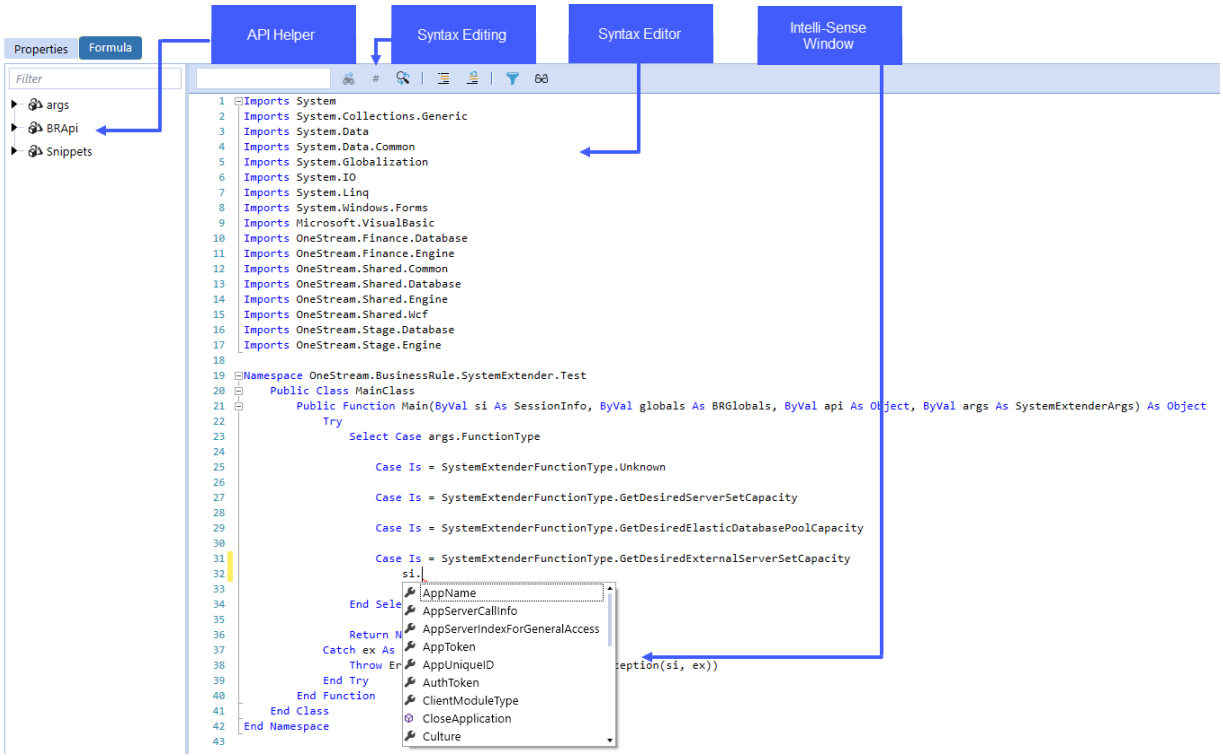
In-Solution Documentation

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 OneStream 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 .Net Core. 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 Solution application development capabilities and continues to evolve with each product release. Like the Data Management Engine, the Presentation Engine interacts with and can call the services of all other engines in the 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 (for example, Parser) to tell other engines (for example, Finance) to run their own APIs (for example, 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.

API Structure and Organization

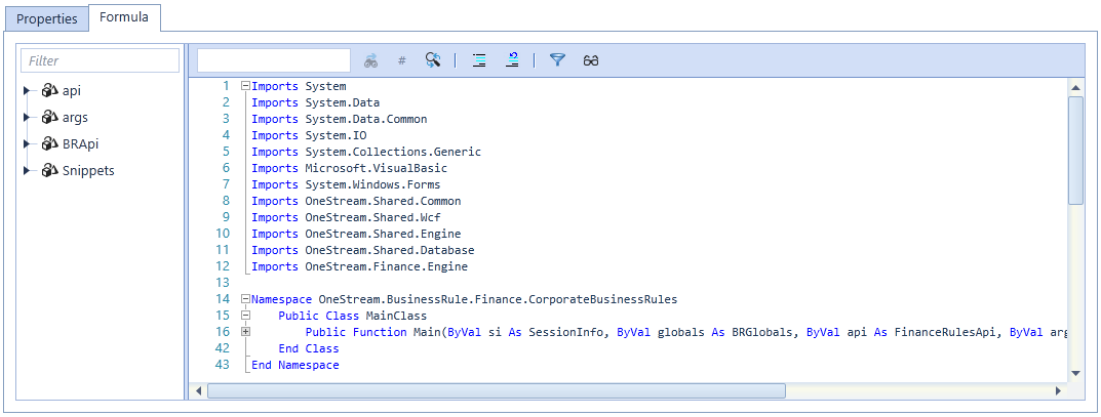
Namespaces

.Net Core 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, .Net Core 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:

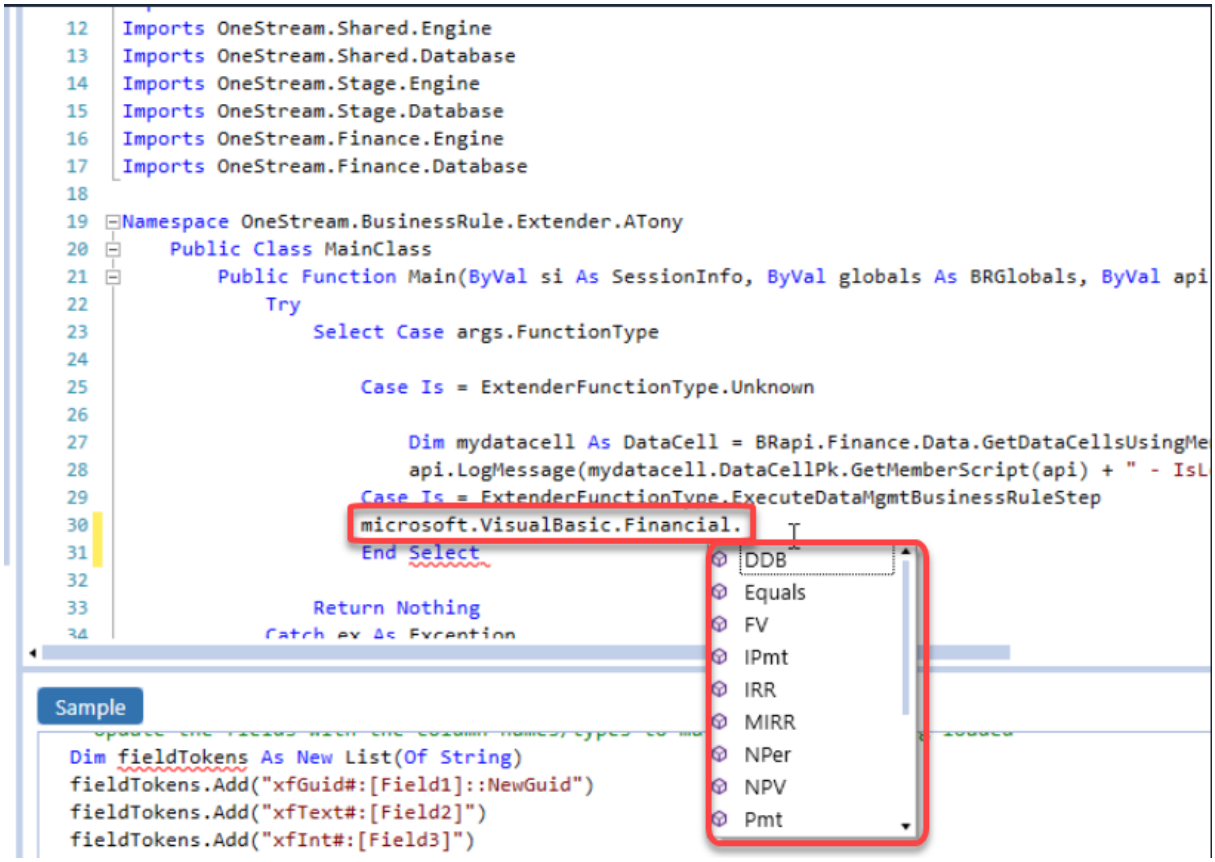
- Depreciation
- Present and future values
- Interest rates
- Rates of return
- Payments

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.



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 SystemExtenderFunctionType.GetDesiredElasticDatabasePoolCapacity

                    Case 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.SQLElasticPoolArgs.DatabaseAndPoolDTU.AzureElasticPoolDTU

        If (args.SQLElasticPoolArgs.AzureElasticPoolLevelMetricValues.DTUConsumptionPercent
> 90) Then
            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`

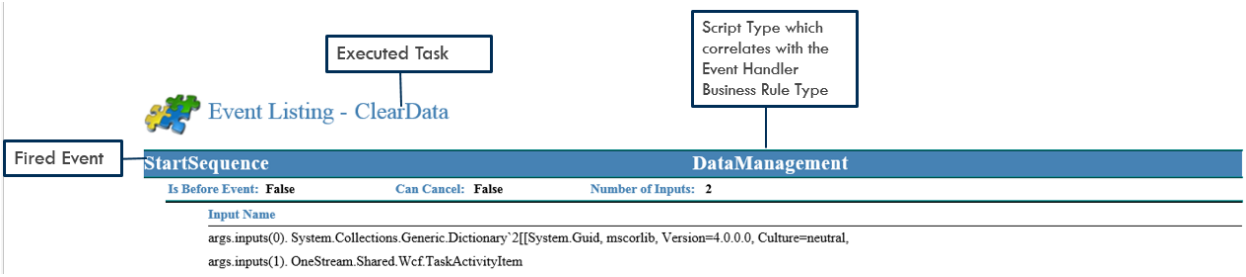
Workflow Event Handler

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

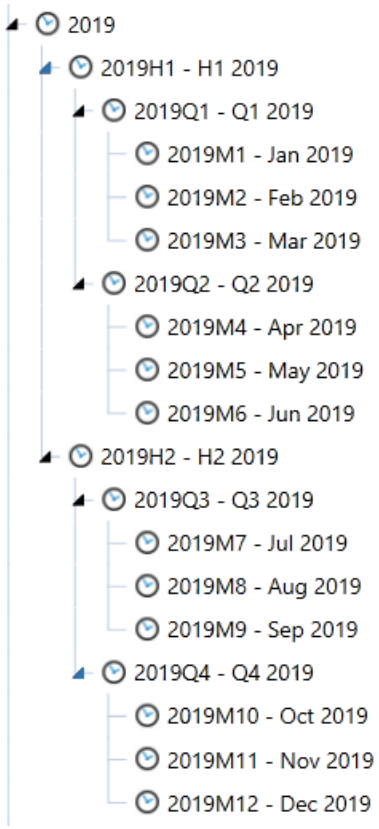
There are many functions that use MemberID as an integer to pass in as a property. These functions get the current POV of the specific Dimension member to perform a variety of tasks, such as:

- 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
```

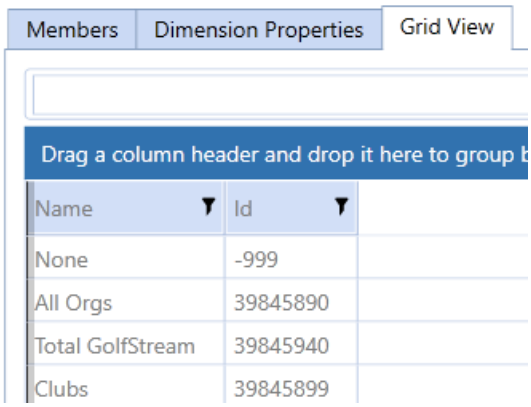
Example using `api.Pov.Time.MemberId` in a working formula:

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

'Execute Formula only if Current Year is Greater Than or Equal to 2023
If curYear >= 2023 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
```

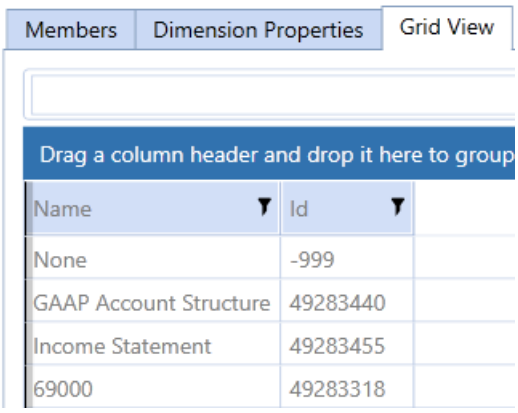
Example using api.Pov.Entity.MemberId in a working formula:

```
'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 =
("#EHousehold:C#Local:$#Actual:T#2019M1:W#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}")
```

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.

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)

'Execute Formula only if Current Year is Greater Than or Equal to 2023
If curYear >= 2023 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)

'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)

'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)

'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)

'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}")
```

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 $Actual:A#10000 and copy the cells to
$ActualCopy:A#CashCalc
Dim destinationInfo As ExpressionDestinationInfo = api.Data.GetExpressionDestinationInfo
("$#ActualCopy")
Dim sourceDataBuffer As DataBuffer = api.Data.GetDataBuffer
(DataApiScriptMethodType.Calculate, "$#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 =
("#EHousehold:C#Local:$#Actual:T#2019M1:W#Periodic:A#Dept_
Intersection:F#None:O#Forms:1#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

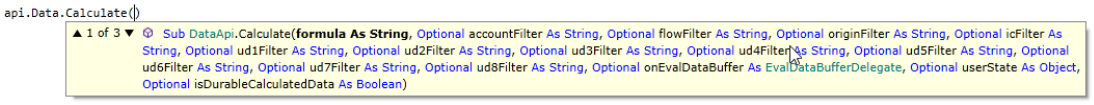
When writing a Member Formula or a Business Rule for a Stored Calculation, the new calculated numbers store data for that Cube, Entity, Parent, Cons, Scenario, and Time combination. For example, a Data Unit.

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 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.

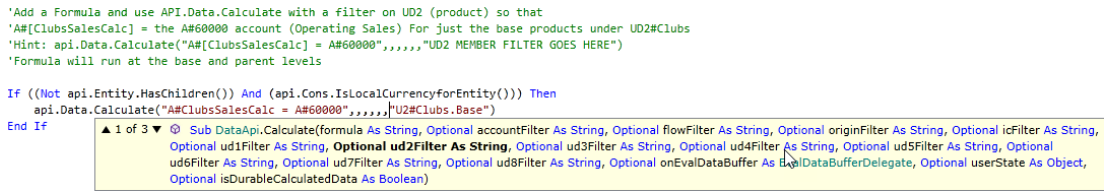
The Api.Data.Calculate function must abide by the data explosion rules, which means that the left side and the right side of the formulas are balanced with the same dimension values on both sides. If a Member is specified for a Dimension anywhere on the right side of the equation, you must explicitly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



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("$F[" & targetScenario & "]:T#[" & selectedTime & "] =
RemoveNoData($F[" & 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:

```
Private Sub OnEvalDataBuffer (ByVal api As FinanceRulesApi, ByVal evalName As String, ByVal
eventArgs As EvalDataBufferEventArgs)
    Try
        'Start with an 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 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
```

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.

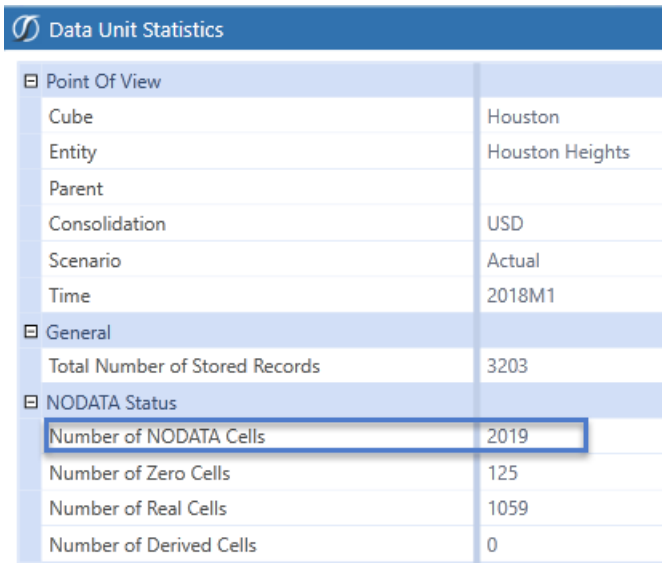
NODATA cells and 0 cells can be found using the following methods:

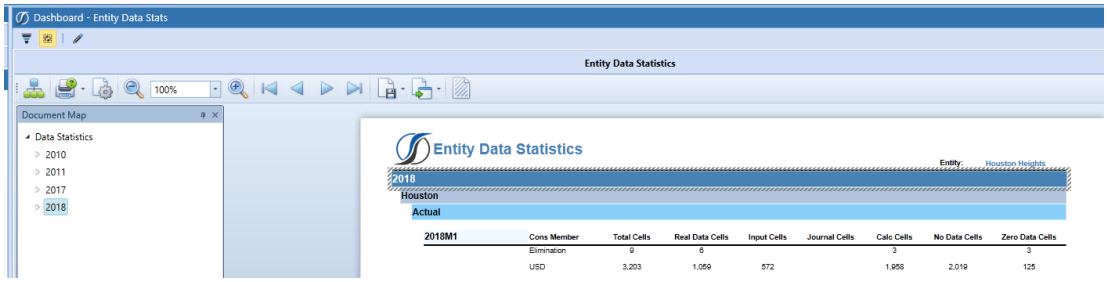
1. Review the Data Unit Statistics when you right-click on a cell in Cube View.
2. Review the Application Analysis Dashboard and check the Entity Data Statistics Report.

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 for 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

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 NoData 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#Non
e, F#None:O#Forms:I#None:U2#None:U3#None:U4#None:U5#None:U6#None:U7#None:U8#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,
(#EGlobal:W#YTD:A#RateAccount:C#USD:F#None:O#BeforeAdj:I#None:U1#None:U2#None:U3#None:U4#Non
e/12), E#Global:W#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("W#Periodic = $result")
```

DataFrame Methods

A DataFrame is an in-memory, tabular data structure designed for flexible data manipulation outside of the cube. It provides a powerful way to retrieve, transform, and analyze data from multiple sources, including SQL queries, Data Adapters, and dynamic system objects. DataFrames support operations similar to traditional tables but optimized for OneStream's analytics workflows, allowing large datasets to be processed efficiently. DataFrames can also be converted to legacy types such as DataTable or DataSet for compatibility with existing and legacy processes.

This topic covers the following:

- DataFrame NameSpace
- Basic DataFrame Functions
- Quickstart DataFrame Examples

For information about IDataFrame, see IDataFrame Interface.

DataFrame NameSpace

To use a DataFrame in a rule, make sure the `OneStream.Data.DataFrame` and `OneStream.Data.DataFrame.Abstractions` NameSpaces are included.

C#:

```
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;
```

VB:

```
Imports OneStream.Data.DataFrame
Imports OneStream.Data.DataFrame.Abstractions
```

Basic DataFrame Functions

The following BRApi.Database functions are available to create DataFrame objects: :

- **GetDataFrame**: Creates a DataFrame and populates it using a database query. See GetDataFrame.
- **CreateDataFramewithColumns**: Creates an empty table with column names. See CreateDataFramewithColumns.
- **CreateDataFrame**: Creates an empty table in memory that can be populated. See DataFrame Methods.
- **CreateEmptyDataFrame**: Creates an empty table. See CreateEmptyDataFrame.

GetDataFrame

`BRApi.Database.GetDataFrame(dbConn, dataFrameName, sqlStatement, dbParamsInfos, useCommandTimeoutLarge)`

Execute an SQL query and return a DataFrame instance. These parameters are used:

Parameter	Type	Description
dbConn	DbConnInfo	Database connection to run the SQL select statement against.
dataFrameName	string	Name of the DataFrame.
sqlStatement	string	SQL statement to be executed.
dbParamInfos (optional)	List<DbParamInfo>	A list of DbParamInfo objects that can be used to substitute SQL parameter names prefixed with @ using the associated parameter value.
useCommandTimeoutLarge	bool	Switch used to indicate if the standard system large timeout value should be used for the statement execution.

Example in C#

```
// get DataFrame using a query
var myFrame = BRApi.Database.GetDataFrame(dbConn, "myFrame", "SELECT * FROM MyTable",
useLargeTimeout);
var myFrame = BRApi.Database.GetDataFrame(dbConn, "myFrame", "SELECT * FROM MyTable where
E=@entity", myDbParamInfos, useLargeTimeout);
```

Example in VB:

```
' get DataFrame using a query
Dim myFrame = BRApi.Database.GetDataFrame(dbConn, "myFrame", "SELECT * FROM MyTable",
useLargeTimeout)
Dim myFrame = BRApi.Database.GetDataFrame(dbConn, "myFrame", "SELECT * FROM MyTable WHERE
E=@entity", myDbParamInfos, useLargeTimeout)
```

CreateDataFramewithColumns

`BRApi.Database.CreateDataFramewithColumns(dataFrameName, columnsInfo)`

Create a DataFrame with a specified set of columns. If the type of column data is not specified, it defaults to String. These parameters are used:

Parameter	Type	Description
dataFrameName	string	Name of the DataFrame.
columnsInfos	List<string> or Dictionary<string, XFDataType>	A list of column names used for the DataFrame, or a dictionary where keys are column names and values are XFDataType constants that specify the type of data for each column.

Example in C#:

```
// create empty DataFrame with columns
var myFrame = BRApi.Database.CreateDataFrameWithColumns("myFrame", colNames);
var myFrame = BRApi.Database.CreateDataFrameWithColumns("myFrame", colNamesAndTypesDict);
```

Example in VB:

```
' create empty DataFrame with columns
Dim myFrame2 = BRApi.Database.CreateDataFrameWithColumns("myFrame", colNames)
Dim myFrame2 = BRApi.Database.CreateDataFrameWithColumns("myFrame", colNamesAndTypesDict)
```

CreateEmptyDataFrame

`BRApi.Database.CreateEmptyDataFrame(dataFrameName)`

Produce an empty DataFrame with default properties. These parameters are used:

Parameter	Type	Description
dataFrameName	string	Name of the DataFrame.

Example in C#:

```
// create empty DataFrame
var myFrame = BRApi.Database.CreateEmptyDataFrame("myFrame") ;
```

Example in VB:

```
' create empty DataFrame
Dim myFrame1 = BRApi.Database.CreateEmptyDataFrame("myFrame")
```

Quickstart DataFrame Examples

The following snippets show a set of common techniques to work with DataFrames. Consult the class reference for the complete api description.

Convert DataFrame to Legacy Types

Use `toDataTable()`, `toDataTable()`, `toDataTable()`, and `toDataTable()` to convert data to a DataTable, XFDataTable, DataSet or RowsArray.

Example in C#:

```
// convert to DataTable
DataTable myDataTable = myDataFrame.toDataTable();

// convert to XFDataTable
XFDataTable myXFDataTable = myDataFrame.toXFDataTable();

// convert to DataSet
DataSet myDataSet = myDataFrame.toDataSet();

// convert to RowsArray
object[][] myNestedArray = myDataFrame.toRowsArray();
```

Example in VB:

```
' convert to DataTable
Dim DataTable As myDataTable = myDataFrame.toDataTable()

' convert to XFDataTable
Dim XFDataTable As myXFDataTable = myDataFrame.toXFDataTable()

' convert to DataSet
Dim DataSet As myDataSet = myDataFrame.toDataSet()

' convert to RowsArray
object[][] myNestedArray = myDataFrame.toRowsArray()
```

Get Metadata from DataFrame

```
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;

// metadata
int numColumns = myDataFrame.ColumnsCount;
int numRows = myDataFrame.RowCount;
string[] columnNames = myNewFrame.GetAllColumnNames();
```

Get DataFrame Column by Position

Get column data by name or ordinal position.

```
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;

// get columns by name or by ordinal position
DataFrameColumn myColumn = myNewFrame.GetColumn("Price");
DataFrameColumn myColumn = myNewFrame.GetColumn(3);
```

Assign Metadata to Columns

Use `addColumn()` to write metadata to columns.

Example in C#

```
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;

// columns carry metadata
Type colType = myColumn.Type;
string colName = myColumn.Name;

// define columns
myFrame.AddColumn(new DataFrameColumn<string>("colA"));
myFrame.AddColumn(new DataFrameColumn<int>("colB"));
```

Example in VB:

```
Imports OneStream.Data.DataFrame
Imports OneStream.Data.DataFrame.Abstractions

' columns carry metadata
Dim colType As Type = myColumn.Type
Dim colName As String = myColumn.Name

' define columns
myFrame.AddColumn(New DataFrameColumn(Of String)("colA"))
myFrame.AddColumn(New DataFrameColumn(Of Integer)("colB"))
```

Add and Retrieve Data

Use `addRow()` to write new rows with values to a DataFrame or `GetColumn()`, `GetValue()`, `GetValueAsString()`, or `GetColumnOrdinal()` to retieve data from a DataFrame.

Example in C#

```
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;

// add and retrieve data
object[] inValues = { "valueField1", 23};
myFrame.addRow(inValues);
for(int c = 0; c < df.ColumnsCount; c++){
 for(int r = 0; r < df.RowCount; r++){
  int myInt = (int) myFrame.GetColumn(c).GetValue(r)
      string myStr = (string) myFrame.GetColumn(c).GetValueAsString(r);
}};
int myInt = (int) myFrame.GetValue(0, myFrame.GetColumnOrdinal("colB"));
```

Example in VB:

```
Imports OneStream.Data.DataFrame
Imports OneStream.Data.DataFrame.Abstractions

' add and retrieve data
Dim inValues As Object() = {"valueField1", 23}
myFrame.addRow(inValues)
For c As Integer = 0 To df.ColumnsCount - 1
```

```
    For r As Integer = 0 To df.RowCount - 1
        Dim myInt As Integer = CType(myFrame.GetColumn(c).GetValue(r), Integer)
        Dim myStr As String = CType(myFrame.GetColumn(c).GetValueAsString(r), String)
    Next
Next
Dim myInt As Integer = CType(myFrame.GetValue(0, myFrame.GetColumnOrdinal("colB")), Integer)
```

DataFrame Conversion

DataFrames can be sliced through conversion.

Example in C#:

```
var myFrame = BRApi.Database.GetDataFrame(...)
var options = DataFrameConversionOptions()
options.MaxRows = 100
options.StartRow = 10
var myXfDataTable = myFrame.ToXFDataTable(options)
```

Example in VB:

```
Dim myFrame = BRApi.Database.GetDataFrame(...)
Dim options As New DataFrameConversionOptions()
options.MaxRows = 100
options.StartRow = 10
Dim myXfDataTable = myFrame.ToXFDataTable(options)
```

Logs

Use `ErrorLog.LogMessage()` to write DataFrame content to the log.

Example in C#:

```
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;
```

```
// Convert the first 10 rows to string for logging
BRApi.ErrorLog.LogMessage(si, "DataFrame Test Output", myFrame.Peek(10));
```

Example in VB:

```
Imports OneStream.Data.DataFrame
Imports OneStream.Data.DataFrame.Abstractions

' Convert the first 10 rows to string for logging
BRApi.ErrorLog.LogMessage(si, "DataFrame Test Output", myFrame.Peek(10))
```

Create DataFrame from Data Adapter

Example in C#:

```
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;

// Look up a Data Adapter and use its query - 9.1+
// note: requires PowerBI Connector support enabled
var myFrame = BRApi.Analytics.GetDataFrameForAdapter(si, isSystemLevel,
 "myWorkspace", "myAdapter", "myResultTable", customSubstVars, null);
```

Example in VB:

```
Imports OneStream.Data.DataFrame
Imports OneStream.Data.DataFrame.Abstractions

'Look up a Data Adapter and use its query - 9.1+
'note: requires PowerBI Connector support enabled
Dim myFrame = BRApi.Analytics.GetDataFrameForAdapter(si, isSystemLevel,
 "myWorkspace", "myAdapter", "myResultTable", customSubstVars, null)
```

Create a Dashboard DataSet Service

The following example implements a Dashboard DataSet using a few different techniques. The first and second datasets are generated by retrieving a table using `GetDataFrame()` and calling `ToDataSet()` to return a DataSet object. The last one creates a DataFrame using `CreateDataFrameWithColumns()` and then uses `ToDataTable()` to export a data table.

Example in C#:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using Microsoft.CSharp;
using OneStream.Finance.Database;
using OneStream.Finance.Engine;
using OneStream.Shared.Common;
using OneStream.Shared.Database;
using OneStream.Shared.Engine;
using OneStream.Shared.Wcf;
using OneStream.Stage.Database;
using OneStream.Stage.Engine;
using OneStreamWorkspacesApi;
using OneStreamWorkspacesApi.V800;
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;


namespace Workspace.__WsNamespacePrefix.__WsAssemblyName
{
    public class DatasetWithDataFrame : IWsasDataSetV800
    {
        public object GetDataSet(SessionInfo si, BRGlobals brGlobals, DashboardWorkspace
workspace, DashboardDataSetArgs args)
        {
            try
            {
                if ((brGlobals != null) && (workspace != null) && (args != null))
                {
                    if (args.DataSetName.XFEqualsIgnoreCase("HelloDataFrame")){

                        // simple query
                        bool useLargeCommandTimeout = false;
                        string frameName  = "MyOutFrame";
                        string sql = "SELECT * FROM Dim"; // list all Dimensions
```

```
                        using(DbConnInfo dbConn = BRApi.Database.CreateApplicationDbConnInfo
(si)){
                            // *** main call - execute SQL and return a dataframe ***
                            var myFrame = BRApi.Database.GetDataFrame(dbConn, frameName,
sql, useLargeCommandTimeout);
                            // once we get the dataframe, convert it to legacy type that the
rule expects
                            // Here you can use .ToDataTable() or .ToDataSet()
                            return myFrame.ToDataSet();
                        }

                    } else if(args.DataSetName.XFEqualsIgnoreCase("QueryWithParams")){
                        //  query with parameters

                        // basic settings
                        bool useLargeCommandTimeout = false;
                        string frameName  = "MyOutFrame";
                        string dimTypeStr = args.NameValuePairs.XFGetValue("Dim", "A");
                        bool isParent, isCube, isBRFN;
                        DimType dimTypeObj = DimType.GetItemFromAbbreviation(dimTypeStr,
                                                    out isParent, out isCube, out isBRFN);

                        // define parameters as names starting with @
                        string sql = "SELECT * FROM Dim WHERE DimTypeId = @dTypeId";

                        // associate your values to the param names
                        var myParams = new List<DbParamInfo>{
                            new DbParamInfo("dTypeId", dimTypeObj.Id)};

                        // execute
                        using(DbConnInfo dbConn = BRApi.Database.CreateApplicationDbConnInfo
(si)){
                            // *** main call - execute SQL and return a dataframe ***
                            var myFrame = BRApi.Database.GetDataFrame(dbConn, frameName,
sql, myParams, useLargeCommandTimeout);
                            // convert to expected legacy type
                            return myFrame.ToDataSet();
                        }
                    } else if(args.DataSetName.XFEqualsIgnoreCase("FromScratch")){

                        // specify type for each column (optional)
                        var columns = new Dictionary<string, XFDataType>{
                                {"Strings", XFDataType.Text},
                                {"Ints", XFDataType.Int32},
                                {"Bools", XFDataType.Boolean},
                                {"Datetimes", XFDataType.DateTime},
                                {"Decimals", XFDataType.Decimal},
                                {"Guids", XFDataType.Guid},
                                {"WeirdosAsString", XFDataType.WorkflowTrackingFrequency}
                            };
                        // create and populate dataframe
                        var myDf = BRApi.Database.CreateDataFrameWithColumns("Output Frame",
columns);
                        myDf.AddRow(new object[]{ "My Text ", 42, true, DateTime.Now, 10.3m,
Guid.Empty, WorkflowTrackingFrequency.HalfYearly });
                        // convert to expected legacy type
```

```
                        return myDf.ToDataTable();

                    }
                }

                return null;
            }
            catch (Exception ex)
            {
                throw new XFException(si, ex);
            }
        }
    }
}
```

Example in VB:

```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Imports Microsoft.VisualBasic
Imports OneStream.Finance.Database
Imports OneStream.Finance.Engine
Imports OneStream.Shared.Common
Imports OneStream.Shared.Database
Imports OneStream.Shared.Engine
Imports OneStream.Shared.Wcf
Imports OneStream.Stage.Database
Imports OneStream.Stage.Engine
Imports OneStream.Data.DataFrame
Imports OneStream.Data.DataFrame.Abstractions
Imports OneStreamWorkspacesApi
Imports OneStreamWorkspacesApi.V800

Namespace Workspace.__WsNamespacePrefix.__WsAssemblyName
    Public Class DatasetWithDataFrame
        Implements IWsasDataSetV800

        Public Function GetDataSet(ByVal si As SessionInfo, ByVal brGlobals As BRGlobals,
ByVal workspace As DashboardWorkspace, _
            ByVal args As DashboardDataSetArgs) As Object Implements
IWsasDataSetV800.GetDataSet
            Try

                If (brGlobals IsNot Nothing) AndAlso (workspace IsNot Nothing) AndAlso (args
IsNot Nothing) Then
                    ' simple query
```

```
                    If args.DataSetName.XFEqualsIgnoreCase("HelloDataFrame") Then
                        Dim useLargeCommandTimeout As Boolean = False
                        Dim frameName As String = "MyOutFrame"
                        Dim sql As String = "SELECT * FROM Dim"          ' List all
Dimensions

                        Using dbConn As DbConnInfo =
BRApi.Database.CreateApplicationDbConnInfo(si)
                            ' *** main call - execute SQL and return a dataframe ***
                            Dim myFrame = BRApi.Database.GetDataFrame(dbConn, frameName,
sql, useLargeCommandTimeout)
                            ' Once we get the dataframe, convert it to legacy type that the
rule expects
                            ' Here you can use .ToDataTable() or .ToDataSet()
                            Return myFrame.ToDataSet()
                        End Using

                    ElseIf args.DataSetName.XFEqualsIgnoreCase("QueryWithParams") Then
                        ' query with parameters

                        ' basic settings
                        Dim useLargeCommandTimeout As Boolean = False
                        Dim frameName As String = "MyOutFrame"
                        Dim dimTypeStr As String = args.NameValuePairs.XFGetValue("Dim",
"A")
                        Dim isParent, isCube, isBRFN As Boolean
                        Dim dimTypeObj As DimType = DimType.GetItemFromAbbreviation
(dimTypeStr, _
                                                                                isParent,
isCube, isBRFN)

                        ' define parameters starting with @
                        Dim sql As String = "SELECT * FROM Dim WHERE DimTypeId = @dTypeId"

                        ' associate your values to param names
                        Dim myParams = New List(Of DbParamInfo) From {
                            New DbParamInfo("dTypeId", dimTypeObj.Id)
                        }

                        ' execute
                        Using dbConn As DbConnInfo =
BRApi.Database.CreateApplicationDbConnInfo(si)
                            ' *** main call - execute SQL and return a dataframe ***
                            Dim myFrame = BRApi.Database.GetDataFrame(dbConn, frameName,
sql, myParams, useLargeCommandTimeout)
                            ' convert to expected legacy type
                            Return myFrame.ToDataSet()
                        End Using

                    ElseIf args.DataSetName.XFEqualsIgnoreCase("FromScratch") Then

                        ' specify type for each column (optional, defaults to string
otherwise)
                        Dim columns = New Dictionary(Of String, XFDataType) From {
                            {"Strings", XFDataType.Text},
                            {"Ints", XFDataType.Int32},
                            {"Bools", XFDataType.Boolean},
```

```
                            {"Datetimes", XFDataType.DateTime},
                            {"Decimals", XFDataType.Decimal},
                            {"Guids", XFDataType.Guid},
                            {"WeirdosAsString", XFDataType.WorkflowTrackingFrequency}
                        }
                        ' create and populate dataframe
                        Dim myDf = BRApi.Database.CreateDataFrameWithColumns("Output Frame",
columns)
                        myDf.AddRow(New Object() {"My Text ", 42, True, DateTime.Now, 10.3D,
Guid.Empty, WorkflowTrackingFrequency.HalfYearly})
                        ' convert to expected legacy type
                        Return myDf.ToDataTable()

                    End If
                End If

                Return Nothing
            Catch ex As Exception
                Throw New XFException(si, ex)
            End Try
        End Function

    End Class
End Namespace
```

Create a Dynamic Grid

The following example shows how to create a Dynamic Grid Service by:

- Using a custom table.
- Populating default values if the table does not exist.
- Writing data back to the table on edit.

Example in C#:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
```

```
using Microsoft.CSharp;
using OneStream.Finance.Database;
using OneStream.Finance.Engine;
using OneStream.Shared.Common;
using OneStream.Shared.Database;
using OneStream.Shared.Engine;
using OneStream.Shared.Wcf;
using OneStream.Stage.Database;
using OneStream.Stage.Engine;
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;
using OneStreamWorkspacesApi;
using OneStreamWorkspacesApi.V800;

namespace Workspace.__WsNamespacePrefix.__WsAssemblyName
{
    public class DynamicGridWithDataFrame : IWsasDynamicGridV800
    {
        private string customTable = "XXX_DataFrameSamples";

        public XFDynamicGridGetDataResult GetDynamicGridData(SessionInfo si, BRGlobals
brGlobals, DashboardWorkspace workspace, DashboardDynamicGridArgs args)
        {
            try
            {
                if ((brGlobals != null) && (workspace != null) && (args != null))
                {

                    // create our dataframe
                    var df = GetOrCreateCustomTable(si, brGlobals, workspace, args);

                    // small hack to force formatting on Decimal column
                    // ** only necessary if dataframe is manually populated and used with
Dynamic Grid
                    var amountColumnFormat = new XFDynamicGridColumnDefinition();
                    amountColumnFormat.ColumnName = "Amount";
                    amountColumnFormat.DataFormatString = "{0:C2}";  // currency, culture-
specific symbol, 2 decimals
                    //amountColumnFormat.DataFormatString = "{0:N2}"; // generic number, no
currency symbol, 2 decimals
                    var columnDefs = new List<XFDynamicGridColumnDefinition>
{amountColumnFormat};

                    // convert to XFDataTable to set extra properties for our grid

                    XFDataTable xfDt = df.ToXFDataTable();
                    xfDt.Columns[0].IsPrimaryKeyColumn = true;
                    xfDt.HasPrimaryKeyColumns = true;
                    XFDynamicGridGetDataResult result = new XFDynamicGridGetDataResult(
                        xfDt, columnDefs, DataAccessLevel.AllAccess );

                    return result;


                }
```

```
                return null;
            }
            catch (Exception ex)
            {
                throw new XFException(si, ex);
            }
        }

        public XFDynamicGridSaveDataResult SaveDynamicGridData(SessionInfo si, BRGlobals
brGlobals, DashboardWorkspace workspace, DashboardDynamicGridArgs args)
        {
            try
            {
                if ((brGlobals != null) && (workspace != null) && (args != null))
                {

                       DashboardDynamicGridGetDataArgs getDataArgs = args.GetDataArgs;

                    DashboardDynamicGridSaveDataArgs saveDataArgs = args.SaveDataArgs;

                    if ((getDataArgs == null) || (saveDataArgs == null))
                    {
                        return null;
                    }

                    List<XFEditedDataRow> editedDataRows = saveDataArgs.EditedDataRows;
                    if (editedDataRows != null)
                    {
                        using (DbConnInfo dbConn =
BRApi.Database.CreateApplicationDbConnInfo(si))
                        {
                            XFDataTableHelper.SaveRows(dbConn, "dbo", customTable,
args.SaveDataArgs.Columns, true,
                                editedDataRows, true, true, true);
                        }
                    }

                    XFDynamicGridSaveDataResult result = new XFDynamicGridSaveDataResult();
                    result.DataTable = GetDynamicGridData(si, brGlobals, workspace,
args)?.DataTable;
                    result.PageIndex = (getDataArgs.StartRowIndex / getDataArgs.PageSize);
                    result.IndexOfSelectedRowOnPage = 0;
                    result.SaveDataTaskResult = new XFDynamicGridSaveDataTaskResult()
                    {
                        IsOK = true,
                        ShowMessageBox = false,
                        Message = "Save Finished"
                    };

                    return result;

                }

                return null;
            }
            catch (Exception ex)
```

```
            {
                throw new XFException(si, ex);
            }
        }

        /** utility method - Retrieve the contents of our custom table, or create and
populate it with default values */
        private IDataFrame GetOrCreateCustomTable(SessionInfo si, BRGlobals brGlobals,
DashboardWorkspace workspace, DashboardDynamicGridArgs args)
        {
            string sqlCheck = @"SELECT
    CASE
        WHEN EXISTS (
            SELECT 1
            FROM INFORMATION_SCHEMA.TABLES
            WHERE TABLE_CATALOG = DB_NAME()
              AND TABLE_NAME = @customTableName
        ) THEN 1
        ELSE 0
END as TableExists";
            var sqlCheckParams = new List<DbParamInfo>{
                new DbParamInfo("customTableName", customTable)
            };
            using(DbConnInfo dbConn = BRApi.Database.CreateApplicationDbConnInfo(si)){

                // check if table already exists
                var dfCheck = BRApi.Database.GetDataFrame(dbConn, "tableCheck", sqlCheck,
sqlCheckParams, false);
                if(Convert.ToBoolean(dfCheck.GetValue<int>(0,0))){
                    // table exists, return contents
                    return BRApi.Database.GetDataFrame(dbConn, "tableData", $"SELECT * FROM
{customTable}", false);

                } else {
                    // table does not exist, create it
                    var df = BRApi.Database.CreateDataFrameWithColumns(customTable,
                        new Dictionary<string, XFDataType>{
                            {"SKU", XFDataType.Int32},
                            {"Item", XFDataType.Text},
                            {"Amount", XFDataType.Decimal},
                            {"Added On", XFDataType.DateTime},
                            {"Available", XFDataType.Boolean},
                            {"RefreshFreq", XFDataType.WorkflowTrackingFrequency}
                        }
                    );
                    string sqlCreate = BuildSqlCreate(df, "SKU");

                    var createResult = BRApi.Database.ExecuteSql(dbConn, sqlCreate, false);

                    // populate it in memory
                    df.AddRow(1, "Lightsaber (green)", 1000m, DateTime.Now, true,
WorkflowTrackingFrequency.Monthly);
                    df.AddRow(2, "Darth Helmet", 10000m, DateTime.Now, false,
WorkflowTrackingFrequency.Yearly);
                    df.AddRow(3, "Bounty Hunter JetPack", 500m, DateTime.Now, true,
WorkflowTrackingFrequency.Quarterly);
```

```
                    df.AddRow(4, "X-Wing", 200000000m, DateTime.Now, true,
WorkflowTrackingFrequency.AllTimePeriods);
                    df.AddRow(5, "Sleaveless Vest", 50m, DateTime.Now, false,
WorkflowTrackingFrequency.HalfYearly);

                    // convert it in order to use convenience calls for saving.

                    var dt = df.ToDataTable();
                    BRApi.Database.SaveCustomDataTable(si, "Application", customTable, dt,
true, false);
                    // return contents
                    return df;
                }
            }


        }

        /** utility method - given a DataFrame, generate a CREATE statement */
        private string BuildSqlCreate(IDataFrame df, string primaryColumn = null)
        {
            var sb = new StringBuilder();
            sb.AppendLine($"CREATE TABLE [{df.Name}] (");
            string primaryKeyCol = null;
            foreach(var col in df.Columns)
            {
                // probably incomplete mapping, but covers the basics
                string sqlType = col.Type == typeof(int) ? "INT"
                    : col.Type == typeof(string) ? "NVARCHAR(MAX)"
                    : col.Type == typeof(decimal) ? "DECIMAL(18,2)"
                    : col.Type == typeof(DateTime) ? "DATETIME"
                    : col.Type == typeof(bool) ? "BIT"
                    : "NVARCHAR(MAX)"; // default fallback

                sb.Append($"    [{col.Name}] {sqlType}");

                // If this is the primary key column and integer, set as IDENTITY and NOT
NULL
                if ((primaryColumn != null) && col.Name.XFEqualsIgnoreCase(primaryColumn))
                {
                    sb.Append(" IDENTITY(1,1) NOT NULL");
                    primaryKeyCol = col.Name;
                }
                else if (!col.Nullable)
                {
                    sb.Append(" NOT NULL");
                }
                sb.Append(",");
                sb.AppendLine();
            }
        // remove last ",\n" -- cross-platform way
        int trimLength = Environment.NewLine.Length + 1;
        sb.Remove(sb.Length - trimLength, trimLength );

            if (!string.IsNullOrEmpty(primaryKeyCol))
            {
```

```
                sb.AppendLine($",    CONSTRAINT [PK_{df.Name}] PRIMARY KEY
([{primaryKeyCol}])");
            }

            sb.Append(");");
            return sb.ToString();

        }

    }
}
```

Example in VB:

```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Imports Microsoft.VisualBasic
Imports OneStream.Finance.Database
Imports OneStream.Finance.Engine
Imports OneStream.Shared.Common
Imports OneStream.Shared.Database
Imports OneStream.Shared.Engine
Imports OneStream.Shared.Wcf
Imports OneStream.Stage.Database
Imports OneStream.Stage.Engine
Imports OneStream.Data.DataFrame
Imports OneStream.Data.DataFrame.Abstractions
Imports OneStreamWorkspacesApi
Imports OneStreamWorkspacesApi.V800

Namespace Workspace.__WsNamespacePrefix.__WsAssemblyName
Public Class DynamicGridWithDataFrame
    Implements IWsasDynamicGridV800

    Private customTable As String = "XXX_DataFrameSamples"

    Public Function GetDynamicGridData(ByVal si As SessionInfo, ByVal brGlobals As
BRGlobals, ByVal workspace As DashboardWorkspace, _
   ByVal args As DashboardDynamicGridArgs) As XFDynamicGridGetDataResult Implements
IWsasDynamicGridV800.GetDynamicGridData
        Try

            If (brGlobals IsNot Nothing) AndAlso (workspace IsNot Nothing) AndAlso (args
IsNot Nothing) Then
    ' create our dataframe -- see function further down
                Dim df = GetOrCreateCustomTable(si, brGlobals, workspace, args)
```

```
    ' small hack to force formatting on Decimal column
    ' ** only necessary if dataframe is manually populated and used
                Dim amountColumnFormat = New XFDynamicGridColumnDefinition()
                amountColumnFormat.ColumnName = "Amount"
                amountColumnFormat.DataFormatString = "{0:C2}"    ' currency, culture-
specific symbol, 2 decimals
    ' amountColumnFormat.DataFormatString = "{0:N2}";  ' generic number, no currency symbol,
2 decimals

                Dim columnDefs = New List(Of XFDynamicGridColumnDefinition) From {
                    amountColumnFormat
                }
    ' Convert to XFDataTable to set extra properties for our grid
    ' Hopefully unnecessary in future releases...
                Dim xfDt As XFDataTable = df.ToXFDataTable()
                xfDt.Columns(0).IsPrimaryKeyColumn = True
                xfDt.HasPrimaryKeyColumns = True
                Dim result As XFDynamicGridGetDataResult = New XFDynamicGridGetDataResult
(xfDt, columnDefs, DataAccessLevel.AllAccess)
                Return result
            End If

            Return Nothing
        Catch ex As Exception
            Throw New XFException(si, ex)
        End Try
    End Function

    Public Function SaveDynamicGridData(ByVal si As SessionInfo, ByVal brGlobals As
BRGlobals, ByVal workspace As DashboardWorkspace, _
  ByVal args As DashboardDynamicGridArgs) As XFDynamicGridSaveDataResult Implements
IWsasDynamicGridV800.SaveDynamicGridData
        Try

            If (brGlobals IsNot Nothing) AndAlso (workspace IsNot Nothing) AndAlso (args
IsNot Nothing) Then
                Dim getDataArgs As DashboardDynamicGridGetDataArgs = args.GetDataArgs
                Dim saveDataArgs As DashboardDynamicGridSaveDataArgs = args.SaveDataArgs

                If (getDataArgs Is Nothing) OrElse (saveDataArgs Is Nothing) Then
                    Return Nothing
                End If

                Dim editedDataRows As List(Of XFEditedDataRow) = saveDataArgs.EditedDataRows

                If editedDataRows IsNot Nothing Then

                    Using dbConn As DbConnInfo = BRApi.Database.CreateApplicationDbConnInfo
(si)
                        XFDataTableHelper.SaveRows(dbConn, "dbo", customTable,
args.SaveDataArgs.Columns, True, editedDataRows, True, True, True)
                    End Using
                End If

                Dim result As XFDynamicGridSaveDataResult = New XFDynamicGridSaveDataResult
()
                result.DataTable = GetDynamicGridData(si, brGlobals, workspace,
```

```
args)?.DataTable
                result.PageIndex = (getDataArgs.StartRowIndex / getDataArgs.PageSize)
                result.IndexOfSelectedRowOnPage = 0
                result.SaveDataTaskResult = New XFDynamicGridSaveDataTaskResult() With {
                    .IsOK = True,
                    .ShowMessageBox = False,
                    .Message = "Save Finished"
                }
                Return result
            End If

            Return Nothing
        Catch ex As Exception
            Throw New XFException(si, ex)
        End Try
    End Function

 ' ** utility method - Retrieve the contents of our custom table, or create and populate it
with default values.
    Private Function GetOrCreateCustomTable(ByVal si As SessionInfo, ByVal brGlobals As
BRGlobals, ByVal workspace As DashboardWorkspace, ByVal args As DashboardDynamicGridArgs) As
IDataFrame
        Dim sqlCheck As String = "SELECT
    CASE
        WHEN EXISTS (
            SELECT 1
            FROM INFORMATION_SCHEMA.TABLES
            WHERE TABLE_CATALOG = DB_NAME()
              AND TABLE_NAME = @customTableName
        ) THEN 1
        ELSE 0
END as TableExists"

        Dim sqlCheckParams = New List(Of DbParamInfo) From {
            New DbParamInfo("customTableName", customTable)
        }

        Using dbConn As DbConnInfo = BRApi.Database.CreateApplicationDbConnInfo(si)

   ' check if table already exists
            Dim dfCheck = BRApi.Database.GetDataFrame(dbConn, "tableCheck", sqlCheck,
sqlCheckParams, False)
            If Convert.ToBoolean(dfCheck.GetValue(Of Integer)(0, 0)) Then
    ' table exists, return contents
                Return BRApi.Database.GetDataFrame(dbConn, "tableData", $"SELECT * FROM
{customTable}", False)
            Else
    ' table does not exist yet, create it
                Dim df = BRApi.Database.CreateDataFrameWithColumns(customTable, New
Dictionary(Of String, XFDataType) From {
                    {"SKU", XFDataType.Int32},
                    {"Item", XFDataType.Text},
                    {"Amount", XFDataType.Decimal},
                    {"Added On", XFDataType.DateTime},
                    {"Available", XFDataType.Boolean},
                    {"RefreshFreq", XFDataType.WorkflowTrackingFrequency}
                })
```

```
                Dim sqlCreate As String = BuildSqlCreate(df, "SKU")
                Dim createResult = BRApi.Database.ExecuteSql(dbConn, sqlCreate, False)
    ' populate it in memory
                df.AddRow(1, "Lightsaber (green)", 1000D, DateTime.Now, True,
WorkflowTrackingFrequency.Monthly)
                df.AddRow(2, "Darth Helmet", 10000D, DateTime.Now, False,
WorkflowTrackingFrequency.Yearly)
                df.AddRow(3, "Bounty Hunter JetPack", 500D, DateTime.Now, True,
WorkflowTrackingFrequency.Quarterly)
                df.AddRow(4, "X-Wing", 200000000D, DateTime.Now, True,
WorkflowTrackingFrequency.AllTimePeriods)
                df.AddRow(5, "Sleaveless Vest", 50D, DateTime.Now, False,
WorkflowTrackingFrequency.HalfYearly)
    ' convert it in order to use convenience calls for saving
                Dim dt = df.ToDataTable()
                BRApi.Database.SaveCustomDataTable(si, "Application", customTable, dt, True,
False)
    ' return contents
                Return df
            End If
        End Using
    End Function

 ' utility method - given a DataFrame, generate a CREATE statement
    Private Function BuildSqlCreate(ByVal df As IDataFrame, ByVal Optional primaryColumn
As String = Nothing) As String
        Dim sb = New StringBuilder()
        sb.AppendLine($"CREATE TABLE [{df.Name}] (")
        Dim primaryKeyCol As String = Nothing

        For Each col In df.Columns
   ' probably incomplete mapping, but covers the basics well
            Dim sqlType As String = _
    If(col.Type = GetType(Integer), "INT", _
     If(col.Type = GetType(String), "NVARCHAR(MAX)", _
      If(col.Type = GetType(Decimal), "DECIMAL(18,2)", _
       If(col.Type = GetType(DateTime), "DATETIME", _
        If(col.Type = GetType(Boolean), "BIT", _
         "NVARCHAR(MAX)"))))) ' default fallback
            sb.Append($"    [{col.Name}] {sqlType}")

   ' If this is the primary key column and integer, set as IDENTITY and NOT NULL
            If (primaryColumn IsNot Nothing) AndAlso col.Name.XFEqualsIgnoreCase
(primaryColumn) Then
                sb.Append(" IDENTITY(1,1) NOT NULL")
                primaryKeyCol = col.Name
            ElseIf Not col.Nullable Then
                sb.Append(" NOT NULL")
            End If

            sb.Append(",")
            sb.AppendLine()
        Next

  ' remove last ",\n" -- cross-platform way
  Dim trimLength as Integer = Environment.NewLine.Length + 1
```

```
  sb.Remove(sb.Length - trimLength, trimLength )

        If Not String.IsNullOrEmpty(primaryKeyCol) Then
            sb.AppendLine($",    CONSTRAINT [PK_{df.Name}] PRIMARY KEY ([{primaryKeyCol}])")
        End If

        sb.Append(");")
        Return sb.ToString()
    End Function
End Class
End Namespace
```

Create a Table View

The following example creates a TableView Service using a DataFrame

Example in C#

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Globalization;
using System.IO;
using System.Linq;
using Microsoft.CSharp;
using OneStream.Finance.Database;
using OneStream.Finance.Engine;
using OneStream.Shared.Common;
using OneStream.Shared.Database;
using OneStream.Shared.Engine;
using OneStream.Shared.Wcf;
using OneStream.Stage.Database;
using OneStream.Stage.Engine;
using OneStream.Data.DataFrame;
using OneStream.Data.DataFrame.Abstractions;
using OneStreamWorkspacesApi;
using OneStreamWorkspacesApi.V800;

/**
 Sample TableView implemented with DataFrame.
**/

namespace Workspace.__WsNamespacePrefix.__WsAssemblyName
{
 public class TableViewWithDataFrame : IWsasTableViewV800
 {
```

```
        public TableView GetTableView(SessionInfo si, BRGlobals brGlobals,
DashboardWorkspace workspace, string tableViewName, Dictionary<string, string>
customSubstVars, Dictionary<string, string> nameValuePairs)
        {
            try {
                if ((brGlobals != null) && (workspace != null)) {
                    if (tableViewName.XFEqualsIgnoreCase("DataFrame")) {
      using(DbConnInfo dbConn = BRApi.Database.CreateApplicationDbConnInfo(si)){
       // retrieve your data
       IDataFrame df = BRApi.Database.GetDataFrame(dbConn,
             "MyTableView", "SELECT * FROM Dim", false);
       // build the tableview
       TableView tv = new TableView();
       // set up headers
       foreach(var col in df.Columns){
        // set basic info
        var tvCol = new TableViewColumn();
        tvCol.Name = col.Name;
        tvCol.Value = col.Name;
        tvCol.IsHeader = true;
        // set the datatype
        tvCol.DataType = XFDataTypeHelper.GetXFDataTypeFromDotNetDataType(si, col.Type);
        // add to table
        tv.Columns.Add(tvCol);
       }
// Loop through rows to populate table
// Note how we keep track of the row index:
//   values belong to *columns*, so to find them we ask a column to "go down X cells".
//  Alternatively, we could keep track of the column position too,
//   and use df.GetValue(colIndex, rowIndex), but it's more work.
       foreach(var item in df.Rows.Select((row, index) => new {row, index})){
        var tvr = new TableViewRow();
        foreach(var col in df.Columns){
   // TableViews want strings everywhere, so we use GetValueAsString.
   // In other situation, we could use GetValueAsObject and cast it to the right type.
         var tvrc = tv.CreateColumn(col.Name, col.GetValueAsString(item.index), false,
true);
         tvr.Items.Add(col.Name, tvrc);
        }
        tv.Rows.Add(tvr);
       }
       return tv;
      }
                    }
                }

                return null;
            }
            catch (Exception ex)
            {
                throw new XFException(si, ex);
            }
        }

        public List<string> TableViewGetCustomSubstVarsInUse(SessionInfo si, BRGlobals
brGlobals, DashboardWorkspace workspace, string tableViewName, Dictionary<string, string>
```

```
custSubstVarsAlreadyResolved)
        {
            try
            {
                if ((brGlobals != null) && (workspace != null))
                {
                    if (tableViewName.XFEqualsIgnoreCase("DataFrame"))
                    {
                    }
                }

                return null;
            }
            catch (Exception ex)
            {
                throw new XFException(si, ex);
            }
        }

  public bool SaveTableView(SessionInfo si, BRGlobals brGlobals, DashboardWorkspace
workspace, string tableViewName, TableView tableView)
        {
            try
            {
                if ((brGlobals != null) && (workspace != null) && (tableView != null))
                {
                    if (tableViewName.XFEqualsIgnoreCase("MyTableView"))
                    {
                    }
                }

                return false;
            }
            catch (Exception ex)
            {
                throw new XFException(si, ex);
            }
        }
 }
}
```

Example in VB:

```
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.Linq
Imports Microsoft.VisualBasic
```

```
Imports OneStream.Finance.Database
Imports OneStream.Finance.Engine
Imports OneStream.Shared.Common
Imports OneStream.Shared.Database
Imports OneStream.Shared.Engine
Imports OneStream.Shared.Wcf
Imports OneStream.Stage.Database
Imports OneStream.Stage.Engine
Imports OneStream.Data.DataFrame
Imports OneStream.Data.DataFrame.Abstractions
Imports OneStreamWorkspacesApi
Imports OneStreamWorkspacesApi.V800

Namespace Workspace.__WsNamespacePrefix.__WsAssemblyName
Public Class TableViewWithDataFrame
    Implements IWsasTableViewV800


    Public Function GetTableView(ByVal si As SessionInfo, ByVal brGlobals As BRGlobals,
ByVal workspace As DashboardWorkspace, ByVal tableViewName As String, ByVal customSubstVars
As Dictionary(Of String, String), _
  ByVal nameValuePairs As Dictionary(Of String, String)) As TableView Implements
IWsasTableViewV800.GetTableView
        Try

            If (brGlobals IsNot Nothing) AndAlso (workspace IsNot Nothing) Then

                If tableViewName.XFEqualsIgnoreCase("DataFrame") Then

                    Using dbConn As DbConnInfo = BRApi.Database.CreateApplicationDbConnInfo
(si)
      ' retrieve your data
                        Dim df As IDataFrame = BRApi.Database.GetDataFrame(dbConn, _
             "MyTableView", "SELECT * FROM Dim", False)
      ' build the tableview
                        Dim tv As TableView = New TableView()
      ' set up headers
                        For Each col In df.Columns
       ' set basic info
                            Dim tvCol = New TableViewColumn()
                            tvCol.Name = col.Name
                            tvCol.Value = col.Name
                            tvCol.IsHeader = True
       ' set datatype
                            tvCol.DataType =
XFDataTypeHelper.GetXFDataTypeFromDotNetDataType(si, col.Type)
       ' add to table
                            tv.Columns.Add(tvCol)
                        Next
  ' Loop through rows To populate table
  ' Note how we keep track Of the row index:
  '   values belong To *columns*, so t' find them we ask a column to "go down X cells".
  '  Alternatively, we could keep track of the column position too,
  '   and use df.GetValue(colIndex, rowIndex), but it's more work.
                        For Each item In df.Rows.[Select](Function(row, index) New With
{row, index
                        })
```

```
                            Dim tvr = New TableViewRow()

                            For Each col In df.Columns
  '  TableViews want strings everywhere, so we use GetValueAsString.
  '  In other situations, we could use GetValueAsObject and cast it to the right type.
                                Dim tvrc = tv.CreateColumn(col.Name, col.GetValueAsString
(item.index), False, True)
                                tvr.Items.Add(col.Name, tvrc)
                            Next

                            tv.Rows.Add(tvr)
                        Next

                        Return tv
                    End Using
                End If
            End If

            Return Nothing
        Catch ex As Exception
            Throw New XFException(si, ex)
        End Try
    End Function


    Public Function TableViewGetCustomSubstVarsInUse(ByVal si As SessionInfo, ByVal
brGlobals As BRGlobals, ByVal workspace As DashboardWorkspace, ByVal tableViewName
As String, ByVal custSubstVarsAlreadyResolved As Dictionary(Of String, String)) _
   As List(Of String) Implements IWsasTableViewV800.TableViewGetCustomSubstVarsInUse
        Try

            If (brGlobals IsNot Nothing) AndAlso (workspace IsNot Nothing) Then

                If tableViewName.XFEqualsIgnoreCase("DataFrame") Then
    ' implement this as you need
                End If
            End If

            Return Nothing
        Catch ex As Exception
            Throw New XFException(si, ex)
        End Try
    End Function

    Public Function SaveTableView(ByVal si As SessionInfo, ByVal brGlobals As BRGlobals,
ByVal workspace As DashboardWorkspace, ByVal tableViewName As String, _
  ByVal tableView As TableView) As Boolean Implements IWsasTableViewV800.SaveTableView
        Try

            If (brGlobals IsNot Nothing) AndAlso (workspace IsNot Nothing) AndAlso
(tableView IsNot Nothing) Then

                If tableViewName.XFEqualsIgnoreCase("DataFrame") Then
    ' implement this as you need
                End If
            End If
```

```
            Return False
        Catch ex As Exception
            Throw New XFException(si, ex)
        End Try
    End Function
End Class
End Namespace
```

IDataFrame Interface

This interface represents an in-memory, tabular data structure composed of named columns and row values.

For information about DataFrame Methods, including NameSpaces, basic functions, and examples, see DataFrame Methods.

Properties and Indexer

Property	Type	Description
ColumnsCount	int	Number of columns.
Name	string?	Optional name/identifier for the DataFrame.
RowCount	int	Number of rows in the frame. This should be preferred to `.Rows.Count`, since the latter will force looping through all rows, affecting performance.

Property	Type	Description
Columns	IEnumerable<IDataFrameColumn>	Enumerable of column descriptors/objects in column ordinal order (zero-based count).
Rows	IEnumerable<object[]>	Enumerable of rows; each row is represented as an array of objects in column order (zero-based count).
this[int columnIndex]	object	Returns a representative value for the given column index.

Methods

Property	Type	Description
AddColumn(IDataFrameColumn dataFrameColumn)	int	Adds a column; returns the column ordinal of the newly added column.

Property	Type	Description
AddColumn(IDataFrameColumn dataFrameColumn, bool bypassColumnShrink)	int	Adds a column with an option to bypass any column-capacity shrinking/resizing logic. Returns the column ordinal.
AddRow(params object[] items)	void	Appends a single row. items should match the number and order of Columns.
AddRowsFromColumnRowArrayAsync(object[][] items)	Task	Asynchronously appends multiple rows from a jagged array. Expected shape is a 2D array, where outer (first) keys specify columns and inner (second) keys specify rows.
Append(IDataFrame dataFrame)	void	Appends all rows/columns from another IDataFrame into this one.

Property	Type	Description
GetAllColumnsNames()	string[]	Returns an array of all column names in ordinal order.
GetColumn(int columnOrdinal)	IDataFrameColumn	Returns the IDataFrameColumn at the specified zero-based ordinal.
GetColumnOrdinal(string name)	int	Returns the zero-based ordinal of the column with the specified name.
GetRowArray(int rowIndex)	object[]	Returns the row at rowIndex as an object[] in column order.
GetValue<T>(int columnOrdinal, int rowIndex)	T	Returns the value at the specified column and row, cast to T. May throw an invalid cast exception if the stored value cannot be converted to T.

Property	Type	Description
GetValueAsObject(int columnOrdinal, int rowIndex)	object	Returns the value as object.
Optimize()	void	Triggers optimizations (e.g., compacting storage, trimming buffers, compressing sparse data). Use to reduce memory footprint or improve subsequent read performance. Use after creating the object manually.
Peek(int maxRecords = 1000)	string	Produces a human-readable preview (string) of up to maxRecords rows. Useful for debugging/logging.
GetParquetStreamAsync()	Task<MemoryStream>	Serializes the frame to Parquet format and returns a MemoryStream containing the content.

Property	Type	Description
GetParquetByteArrayAsync()	Task<byte[]?>	Serializes to Parquet and returns the bytes, or null if serialization is not possible or fails.
ToDataSet()	DataSet	Converts the frame to a DataSet.
ToDataTable()	DataTable	Converts to a single DataTable.
ToRowsArray()	object[][]	Returns all rows as a 2D jagged array (object[][]) — each inner array is a row.
Split(int maxRowsPerDataFrame)	IDataFrame[]	Splits the frame into multiple IDataFrame segments, each with up to maxRowsPerDataFrame rows. Useful for batching/parallel processing.

Property	Type	Description
ToXFDataTable(this IDataFrame dataFrame, string tableName = null)	XFDataTable	Converts a DataFrame to an XFDataTable using default options. Table name defaults to "DataFrame" if not provided.
ToXFDataTable(this DataFrame dataFrame, Dictionary<string,string> columnMappings, string tableName = null)	XFDataTable	Converts a DataFrame with explicit column name mapping. columnMappings must be a map of source column-name -> XF column-name. If a source name is not present, the original name is used. Value types default to String if not specified.

Property	Type	Description
ToXFDataTable(this DataFrame dataFrame, DataFrameConversionOptions options)	XFDataTable	Converts a DataFrame controlling conversion to exclude columns, rename columns, override column data types, mark primary/read-only columns, limit rows, set start-row, and apply per-column value transformations. If options is null, a default DataFrameConversionOptions is created and TableName defaults to the DataFrame name or "DataFrame".

DataFrameConversionOptions

This class is a wrapper to specify options for converting DataFrames.

Property	Type	Description
TableName	string	Custom table name (default string.Empty; conversion will choose the DataFrame name or "DataFrame" if left empty).
MaxRows	int	Maximum amounts of rows to convert. Defaults to 0, which means "all rows".
StartRow	int	Convert rows starting from the rows at the specified position. Defaults to 0.

Property	Type	Description
ExcludeColumns	HashSet<string>	Columns to exclude.
ColumnNameMappings	Dictionary<string,string>	Rename column names from source keys to values.
DataTypeMappings	Dictionary<string,XFDataType>	Override XF data types in converted output.
PrimaryKeyColumns	HashSet<string>	Columns flagged as primary keys.
ReadOnlyColumns	HashSet<string>	Columns flagged read-only.

Property	Type	Description
ValueTransformations	Dictionary<string, Func<T, TResult>>	Per-column transform functions. The passed function will receive a value of type T and output a value of type TResult for each element in the column specified as key.

IDataFrameColumn

This interface represents a single named column in an in-memory DataFrame. It is typically accessed through the collection.

Columns found on IDataFrame instances.

Property	Type	Description
Name	string	Column name (used as the key when converting to other tabular formats).

Property	Type	Description
Type	Type	CLR type representing the stored values (nullable wrappers are usually unwrapped by callers).
Length	int	Number of values in the column (may be less than the parent frame's RowCount for sparse or streaming implementations).
GetValueAsObject(int rowIndex)	object	Returns the raw object stored at the specified zero-based row index, or null if missing/invalid.