API Overview Guide

6.6.0 Release

Table of Contents

Introduction 1

Development Technologies 2

- Programming Language 2
- User Interface Technology 2
- Server Technology 2
- Database Technology 3
  - OneStream API Details and Database Documentation 3

Developer Fundamentals 4

- VB.Net 4
- In-Solution Documentation 4
- Business Rules Editor Overview 4
  - Helpful Resources 5
  - Microsoft VB.Net Learning 5
  - Microsoft Visual Basic 5

Platform Engines 6

- Workflow Engine 6
- Stage Engine 6
- Finance Engine 6
- Data Quality Engine 7

Data Management Engine 7

Presentation Engine 7

BRApi 8

Business Rules 9

Anatomy of a Business Rule 9

Business Rule Definition 9

Business Rule Classifications 11

Event Handler Business Rules 12

Complex Expressions 15

Business Rule Types 20

Business Rule Organization and Referencing 29

API Structure and Organization 35

Namespaces 35

Namespaces Defined 36

Namespace Hierarchy 36

In-Solution Development 38

Custom Development 39

System Tools 40

System Business Rules 40

Database 41

Tables 41
Tools 41
Data Records 41
Client API Listing 42
Client API Object Hierarchy 42
PowerShell 46
Event Listing 50
Event Handler Business Rules 50
Event Firing Sequences 53

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.

Development Technologies

Programming Language

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

User Interface Technology

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

Server Technology

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

Database Technology

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

OneStream API Details and Database Documentation

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

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

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

Developer Fundamentals

VB.Net

The OneStream platform is based entirely on the Microsoft .Net Framework as is the Business Rules engine. Therefore, VB.Net is 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 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 is a full featured programming language. In the end, these capabilities result in a more reliable Business Rule code.

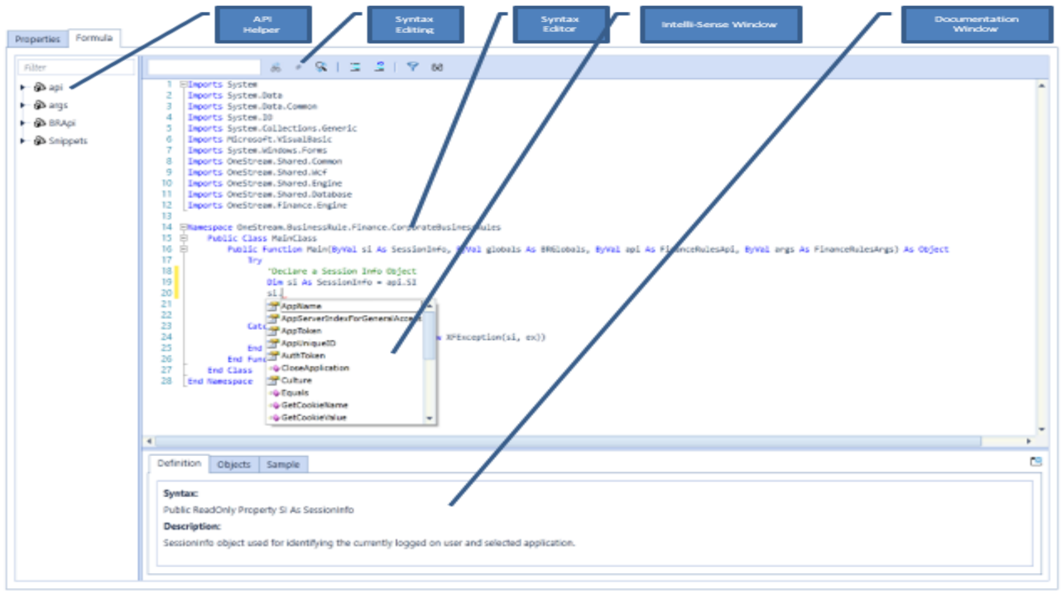
In-Solution Documentation

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

Business Rules Editor Overview

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

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



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

Namespace OneStream.BusinessRule.Finance.CorporateBusinessRules
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As FinanceRulesApi, ByVal args As FinanceRulesArgs) As Object
            Try
                'Declare a Session Info Object
                Dim si As SessionInfo = api.SI
                si.


            Catc                                              w XFException(si, ex))

            End
        End Func
    End Class
End Namespace
```

Syntax:
Public ReadOnly Property SI As SessionInfo
Description:
SessionInfo object used for identifying the currently logged on user and selected application.

Helpful Resources

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

Microsoft VB.Net Learning

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

Microsoft Visual Basic

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

Platform Engines

The platform is comprised of multiple processing engines. These engines have distinct responsibilities with respect to system processing and consequently they expose different API interfaces to the Business Rules they call. This section provides a brief overview of each engine in the platform and describes the engine's core responsibilities.

Workflow Engine

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

Stage Engine

The Stage Engine performs the task of sourcing and transforming external data into valid analytic data points. The main responsibility of this engine is to read source data *(files or systems)* and parse the information into a tabular format. This allows the data to be transformed or mapped to valid Members defined by the Finance Engine. The Stage Engine is an in-memory, multi-threaded engine that provides the opportunity to interact with source data as it is being parsed and transformed. In addition to parsing and transforming data, the Stage Engine also has a sophisticated calculation that enables data to be derived and evaluated based on incoming source data. The Stage Engine provides quality services to source data by validating, mapping, and executing Derivative Check Rules.

Finance Engine

The Finance Engine is an in-memory financial analytic engine. The main responsibility of this engine is to enrich and aggregate base data cells into consolidated multi-Dimensional information. The Finance Engine provides the opportunity to define sophisticated financial calculations through centralized Business Rules as well as member specific Business Rules *(Member Formulas).* It works concurrently with the Stage Engine to validate incoming intersections and works with the Data Quality Engine to execute Confirmation Rules which are used to validate analytic data values.

Data Quality Engine

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

Data Management Engine

The Data Management Engine provides task automation services to the platform. This engine executes batches of commands that are organized into sequences which contain steps. Steps represent entry points or mechanisms to execute features of other engines. For example, the Clear Data Step uses the services of the Finance Engine. In addition, the Data Management Engine has the ability to execute a Business Rule Step which executes a custom Business Rule as part of a Data Management Sequence. This is an incredibly powerful capability because it provides the ability to string together any combination of predefined processing steps with custom Business Rule steps.

Presentation Engine

The Presentation Engine provides extensive data visualization services to platform. The Presentation Engine is made up of the following component engines: Cube View Engine, Dashboard Engine, Parameter Engine, Book Engine and Extensible Document Engine. The Presentation Engine is responsible for managing and delivering content to the end user as well as providing a development environment for custom user interface elements. This engine enables OneStream MarketPlace application development capabilities and continues to evolve with each product release. Like the Data Management Engine, the Presentation Engine interacts with and can call the services of all other engines in the product.

BRApi

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

Business Rules

Anatomy of a Business Rule

This section provides a detailed explanation of the following:

- Business Rule structure and fundamentals
- Business Rule Classifications
- Specific Business Rule Types
- Business Rule organization
- OneStream Business Rule framework
- Best practices for Business Rule architecture

Business Rule Definition

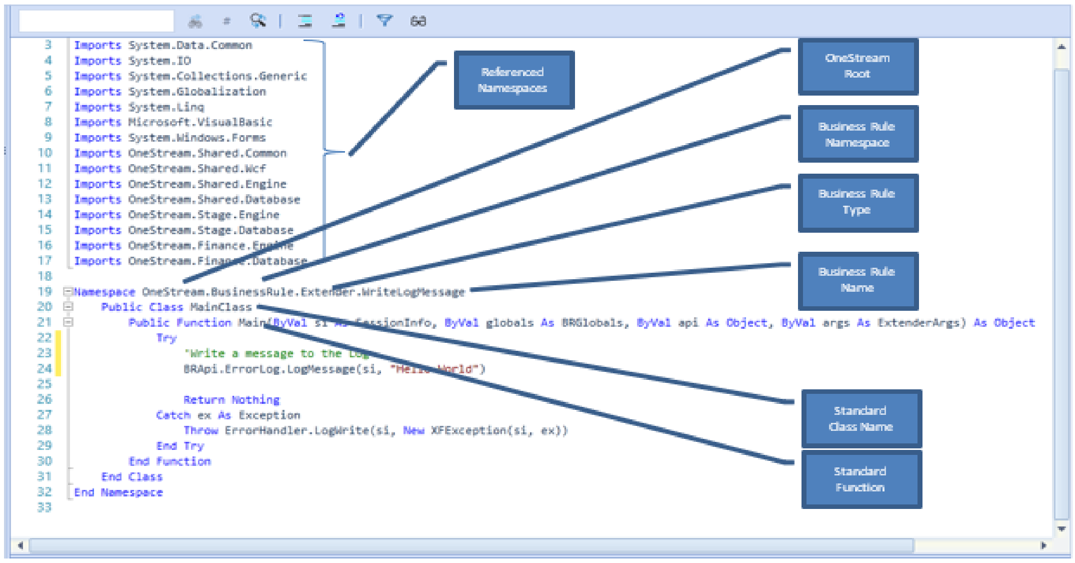
A Business Rule is a VB.Net Class meaning each OneStream Business Rule is an independent object encapsulating VB.Net code. A Business Rule can be as simple as a one-line call to write a log message, or it can be a full code library containing other custom VB.Net Classes, Methods and Properties.

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

Predefined Object Names

- Namespace: `OneStream.BusinessRule.<Business Rule Type>.<Unique Business Rule Name>`
- Class: `MainClass;`
- Function: `Main`

Example Business Rule Structure



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

Namespace OneStream.BusinessRule.Extender.WriteLogMessage
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As Object, ByVal args As ExtenderArgs) As Object
            Try
                'Write a message to the Log
                BRApi.ErrorLog.LogMessage(si, "Hello World")

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

Function Prototypes

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

`Public Function Main`

`(`

`ByVal si As SessionInfo,` Connection Object Required to use API

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

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

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

`)`

`As Object`

Business Rule Classifications

OneStream provides Classifications for business logic organization. At the core, all business logic is delivered and executed as compiled VB.Net code. This means no matter what type of business logic is used, there is a consistency in the syntax and compilation process. The reason for different Classifications has to do with when and how the business logic is invoked and how the Business Rule is scoped. There are two broad Business Rule Classifications: Shared Business Rules and Item Specific Business Rules. Each engine in the system may support one or both Business Rule Classifications. Whenever a processing sequence is executed within the platform, the particular engine(s) involved evaluates how/what business logic is associated with the process. This may include Shared Business Rules *(Named and Event Handlers)* as well as Item Specific Business Rules (*Member Formulas, Logical Expressions, and Confirmation Rules*).

Finance Engine Example

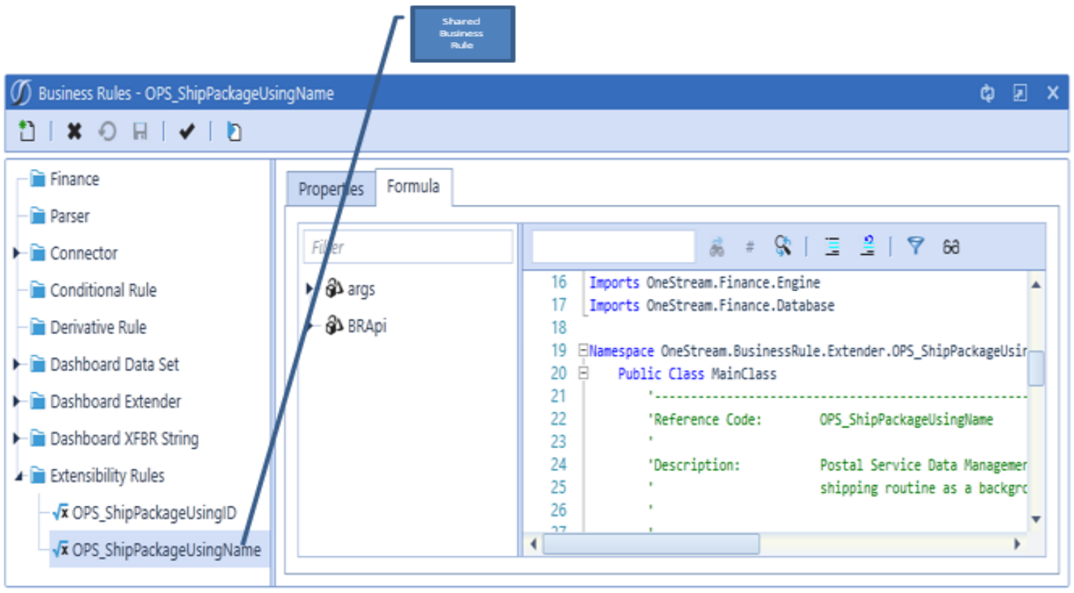
During a consolidation process, a Named Business Rule is associated with the Cube being processed. The Cube contains Member Formulas associated with some of its Dimensions. In this case, the Finance Engine compiles both the Named Business Rule and each individual Member Formula in preparation for the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

Stage Engine Example

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

Shared Business Rules

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



```
Imports OneStream.Finance.Engine
Imports OneStream.Finance.Database

Namespace OneStream.BusinessRule.Extender.OPS_ShipPackageUsir
    Public Class MainClass
        '--------------------------------------------------------------
        'Reference Code:        OPS_ShipPackageUsingName
        '
        'Description:           Postal Service Data Managemer
        '                       shipping routine as a backgrc
        '
```

Event Handler Business Rules

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

Predefined Event Handler Business Rules

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

- Data Management Event Handler
- Data Quality Event Handler
- Forms Event Handler
- Journal Event Handler
- Save Data Event Handler

- Transformation Event Handler
- Workflow Event Handler
- Wcf Event Handler

Item Specific Business Rules

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

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

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

Item Specific vs Shared Code Structure

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

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

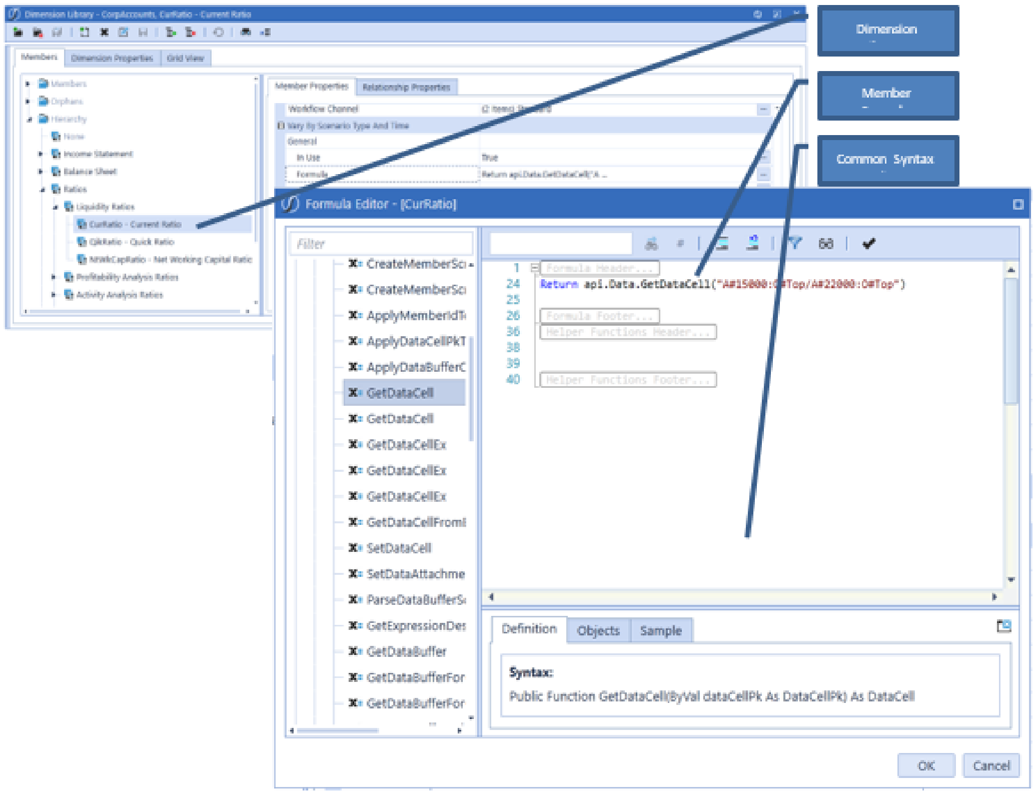
These hidden sections *(i.e. Regions)* keep the formula / expression as readable as possible. In a Shared Business Rule, these sections are visible which make the rule more verbose. The idea behind the Item Specific Business Rule is to create discrete code blocks that are easy to manage and have limited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

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

Member Formulas

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

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



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

Formula Footer...
Helper Functions Header...


Helper Functions Footer...
```

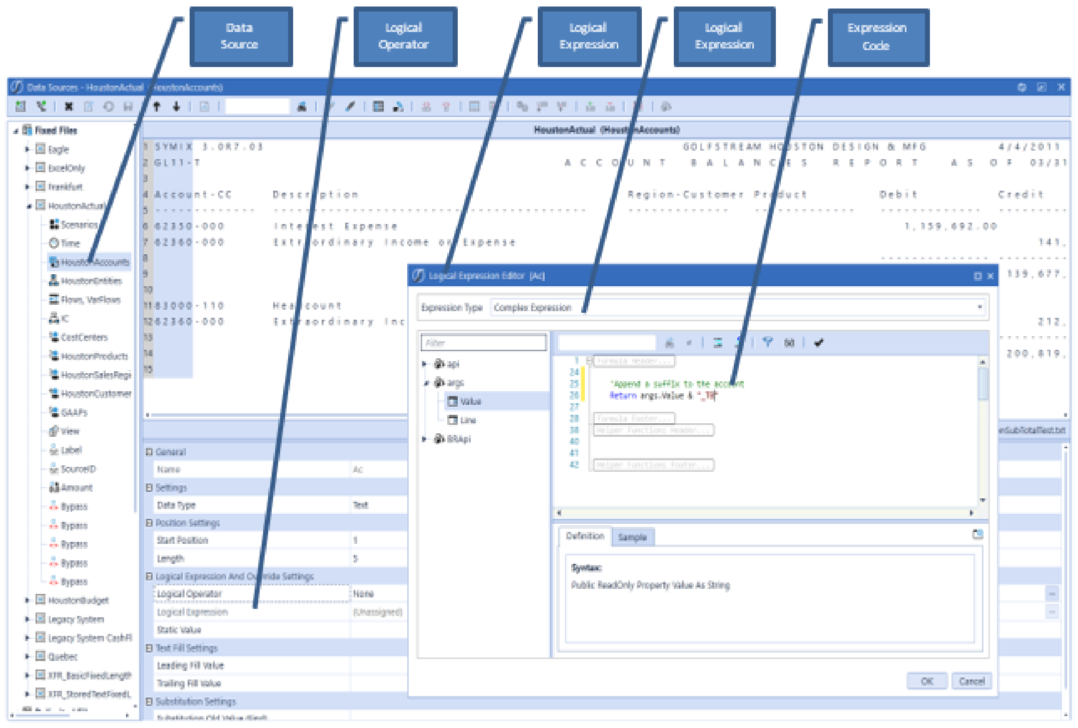
Syntax:
Public Function GetDataCell(ByVal dataCellPk As DataCellPk) As DataCell

Complex Expressions

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

Using Complex Expressions in a Data Source

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



Expression Type: Complex Expression

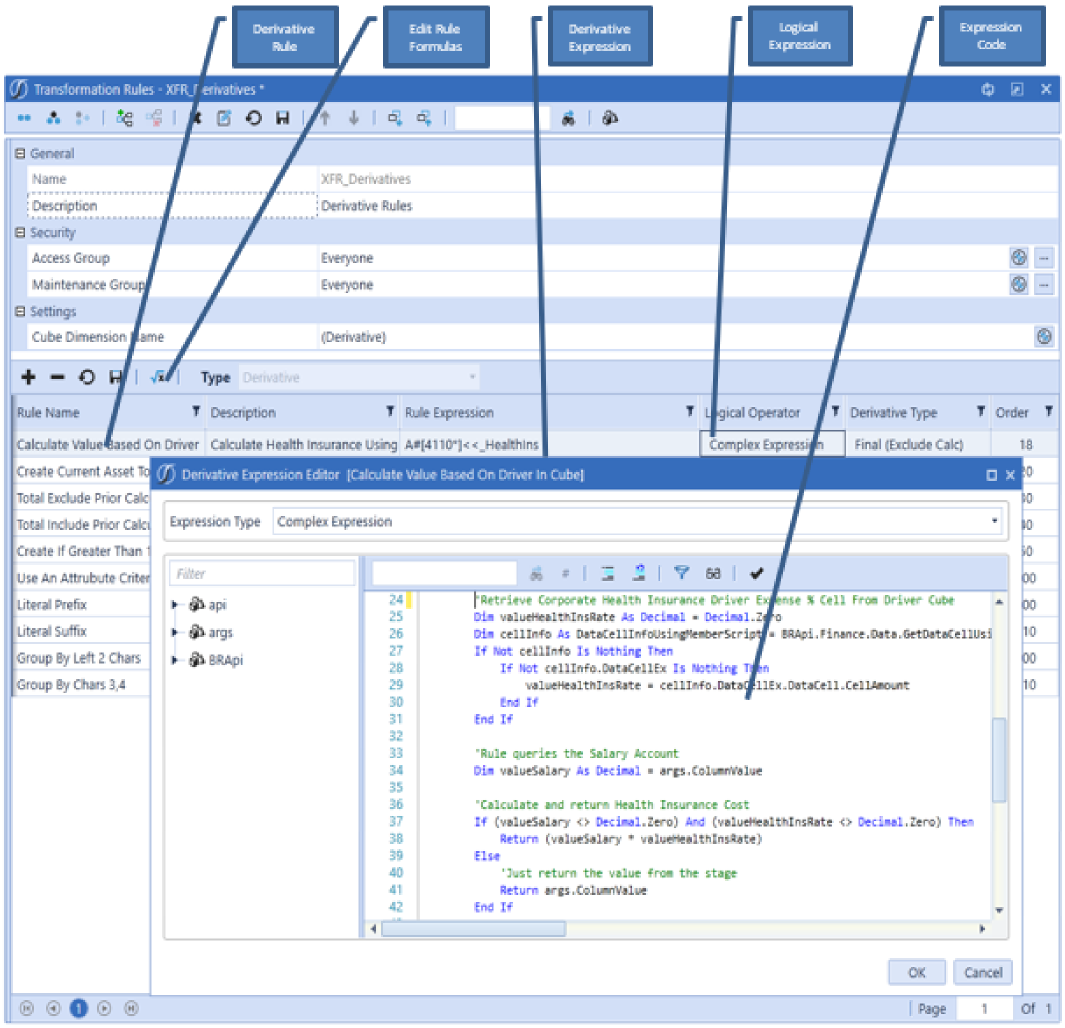
```
'Append a suffix to the account
Return args.Value & "_TR"
```

Syntax:
Public ReadOnly Property Value As String

General	
Name	Ac
Settings	
Data Type	Text
Position Settings	
Start Position	1
Length	5
Logical Expression And Override Settings	
Logical Operator	None
Logical Expression	(Unassigned)
Static Value	
Text Fill Settings	
Leading Fill Value	
Trailing Fill Value	
Substitution Settings	

Using Complex Expressions in a Derivative Rule

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



General

Name	XFR_Derivatives
Description	Derivative Rules

Security

Access Group	Everyone
Maintenance Group	Everyone

Settings

Cube Dimension Name	(Derivative)

Type Derivative

Rule Name	Description	Rule Expression	Logical Operator	Derivative Type	Order
Calculate Value Based On Driver	Calculate Health Insurance Using	A#[4110*]<<_HealthIns	Complex Expression	Final (Exclude Calc)	18
Create Current Asset To					
Total Exclude Prior Calc					
Total Include Prior Calc					
Create If Greater Than 1					
Use An Attrubute Criter					
Literal Prefix					
Literal Suffix					
Group By Left 2 Chars					
Group By Chars 3,4					

Derivative Expression Editor [Calculate Value Based On Driver In Cube]

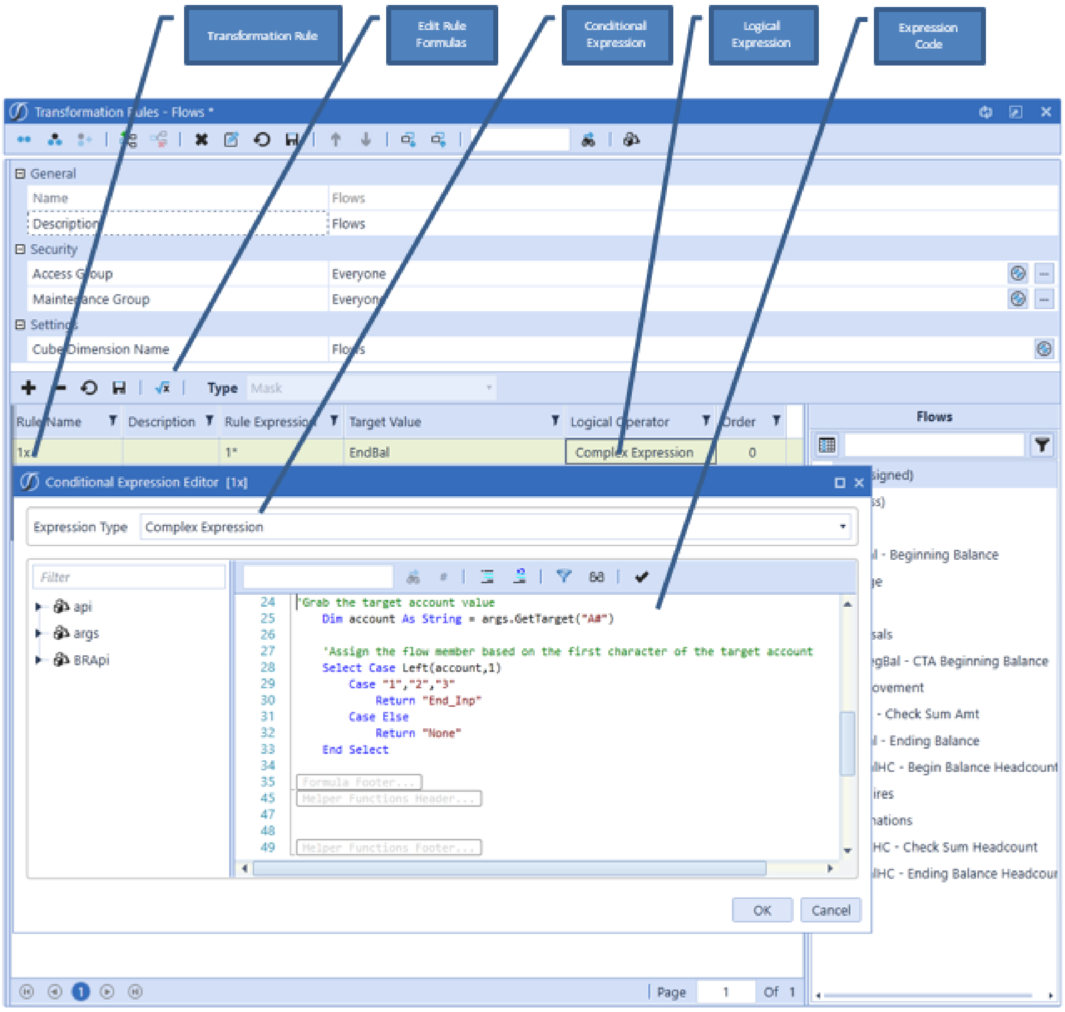
Expression Type: Complex Expression

```
'Retrieve Corporate Health Insurance Driver Expense % Cell From Driver Cube
Dim valueHealthInsRate As Decimal = Decimal.Zero
Dim cellInfo As DataCellInfoUsingMemberScript = BRApi.Finance.Data.GetDataCellUsi
If Not cellInfo Is Nothing Then
    If Not cellInfo.DataCellEx Is Nothing Then
        valueHealthInsRate = cellInfo.DataCellEx.DataCell.CellAmount
    End If
End If

'Rule queries the Salary Account
Dim valueSalary As Decimal = args.ColumnValue

'Calculate and return Health Insurance Cost
If (valueSalary <> Decimal.Zero) And (valueHealthInsRate <> Decimal.Zero) Then
    Return (valueSalary * valueHealthInsRate)
Else
    'Just return the value from the stage
    Return args.ColumnValue
End If
```

Using Complex Expressions in a Conditional Transformation Rule

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

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



General
Name: Flows
Description: Flows
Security
Access Group: Everyone
Maintenance Group: Everyone
Settings
Cube Dimension Name: Flows

Type: Mask

Rule Name	Description	Rule Expression	Target Value	Logical Operator	Order
1x		1*	EndBal	Complex Expression	0

Expression Type: Complex Expression

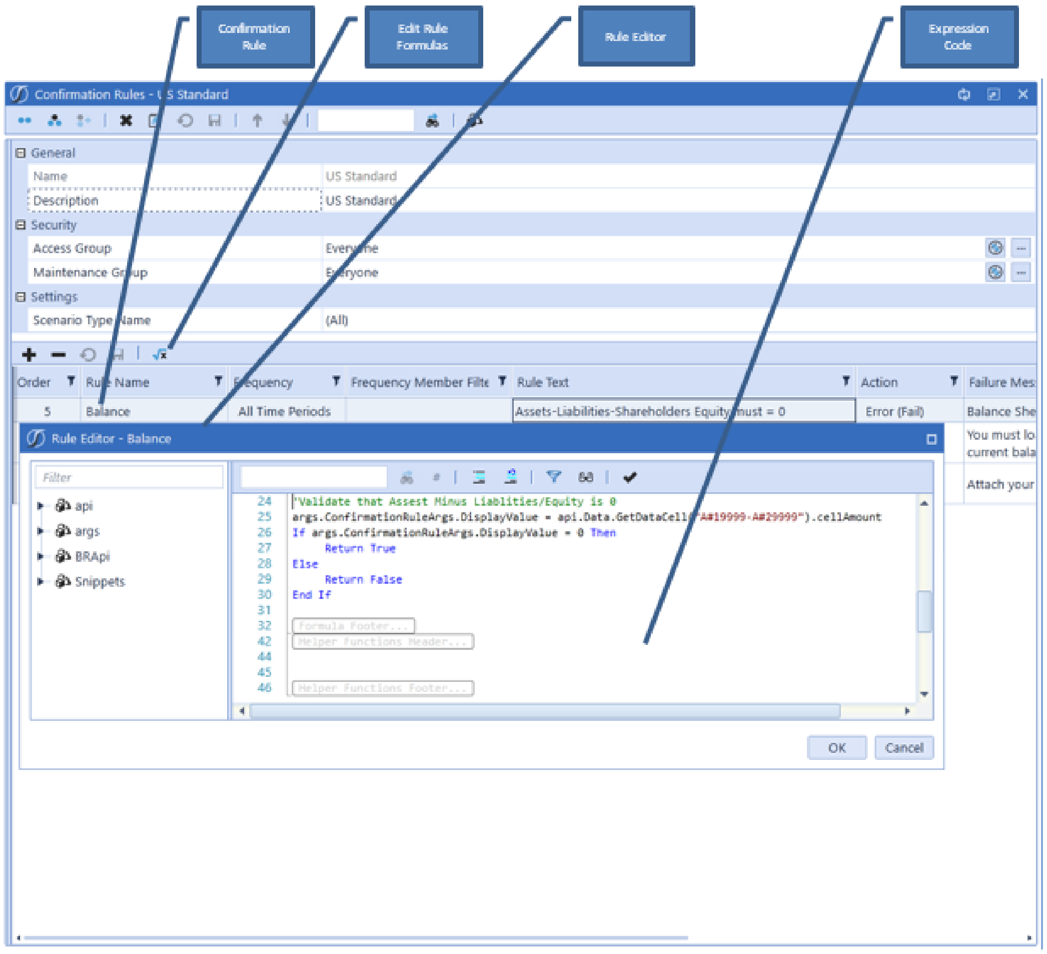
```
'Grab the target account value
    Dim account As String = args.GetTarget("A#")

    'Assign the flow member based on the first character of the target account
    Select Case Left(account,1)
        Case "1","2","3"
            Return "End_Inp"
        Case Else
            Return "None"
    End Select

Formula Footer...
Helper Functions Header...


Helper Functions Footer...
```

Confirmation Rules

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

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

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



General

Name	US Standard
Description	US Standard

Security

Access Group	Everyone
Maintenance Group	Everyone

Settings

Scenario Type Name	(All)

Order	Rule Name	Frequency	Frequency Member Filte	Rule Text	Action	Failure Mes
5	Balance	All Time Periods		Assets-Liabilities-Shareholders Equity must = 0	Error (Fail)	Balance She

```
'Validate that Assest Minus Liablities/Equity is 0
args.ConfirmationRuleArgs.DisplayValue = api.Data.GetDataCell("A#19999-A#29999").cellAmount
If args.ConfirmationRuleArgs.DisplayValue = 0 Then
     Return True
Else
     Return False
End If

Formula Footer...
Helper Functions Header...


Helper Functions Footer...
```

Business Rule Types

Finance

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

Invoking Engine
Finance

API Object Type
FinanceAPI

Args Object Type
FinanceRulesApi

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

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

Common Usage

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

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

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

Parser

Parser Business Rules are used to evaluate and/or modify field values being processed by the Stage Parser Engine as it reads source data. These Business Rules are written as Shared Business Rules or Logical Expressions and applied to a Data Source Dimension.

Invoking Engine
Stage

API Object Type
ParserDimension

Args Object Type
ParserArgs

Common Usage

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

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

Connector

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

Invoking Engine
Stage

API Object Type
Transformer

Args Object Type
ConnectorArgs

Common Usage

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

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

Conditional Rule

Conditional Rules *(mapping)* are used to conditionally evaluate mapping criteria during the data transformation process. These Business Rules are written as Shared Business Rules or Logical Expressions and applied to a Transformation Rule definition.

Invoking Engine
Stage

API Object Type
Transformer

Args Object Type
ConditionalRuleArgs

Common Usage

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

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

DerivativeRule

Derivative Rules *(derive data prior to mapping)* are used to evaluate and/or calculate values during the data derivation process. These Business Rules are written as Shared Business Rules or Logical Expressions and applied to a Derivative Rule definition.

Invoking Engine
Stage

API Object Type
Transformer

Args Object Type
DerivativeRuleArgs

Common Usage

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

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

Cube View Extender

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

Invoking Engine
Presentation

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

Args Object Type

CubeView

CubeViewExtenderFunctionType

CubeViewExtenderReport

CustomSubVars

FunctionType

Common Usage

- Display different logos on select reports based on conditional logic or security and manage their placement and size
- Customize the page number in the header or footer
  Page numbers can be on the top or bottom row of a report and the horizontal position can be specified for rows. This only applies to the top or bottom rows.
- Format individual header and footer fields
- Customize the Cube View Header
  - Control the Left, Right, Center Subtitle widths
  - Control the font size of Title and Subtitles
- Customize the date display
- Customize bottom text alignment
- Apply Conditional Formatting
  Format cells based on their contents. Change the text color of a value in order to effectively hide the result.
- Customized Report row and column formatting such as borders, background and text colors and alignment

DashboardDataSet

DashboardDataSet Rules are used to create programmatic query results. This rule type combines multiple types of data into a single result set using the full syntax capability of VB.Net. These Business Rules are written as Shared Business Rules and applied to Dashboard Data Adapters or Dashboard Parameters.

Invoking Engine
Presentation

API Object Type

No specific API (used General BRApi)

Args Object Type
DashboardDataSetArgs

Common Usage

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

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

DashboardExtender

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

Invoking Engine
Presentation

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

Args Object Type
DashboardExtenderArgs

Common Usage

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

- Execute a Task when the User Clicks a Button
- Perform a Task and Show a Message to the User
- Perform a Custom Calculation
- Upload a File from the End User’s Machine
- Automate a Workflow
- Build a Custom Workflow
- Create Custom Data Tables
- These rules are basically limited to the imagination of the developer

DashboardStringFunction

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

Invoking Engine
Presentation

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

Args Object Type DashboardStringFunctionArgs

Common Usage

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

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

Extender

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

Invoking Engine Business Rule, Data Management, Finance

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

Args Object Type

ExtenderArgs

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

- ExtenderArgs.DataMgmtArgs
- ExtenderArgs.ExternalDimSourceArgs

Common Usage

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

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

Business Rule Organization and Referencing

The Business Rule framework provided by OneStream organizes the Business Rules so that reuse can be maximized. There are many situations where a Business Rule writer may create a standard function that is reused in many other Business Rules. The platform provides a way for Business Rules to be linked and called from other Business Rules. In addition, the platform provides a way for external DLL's to be linked and called from a Business Rule.

This section describes how to reference a Shared Business Rule from a within a Business Rule and how to reference an external DLL from within a Business Rule.

Defining a Reference to a Shared Business Rule

When a Shared Business Rule is created, its public members can be referenced and executed by other Shared and Item Specific Business Rules.

Common reasons to create a shared or referenced Business Rule:

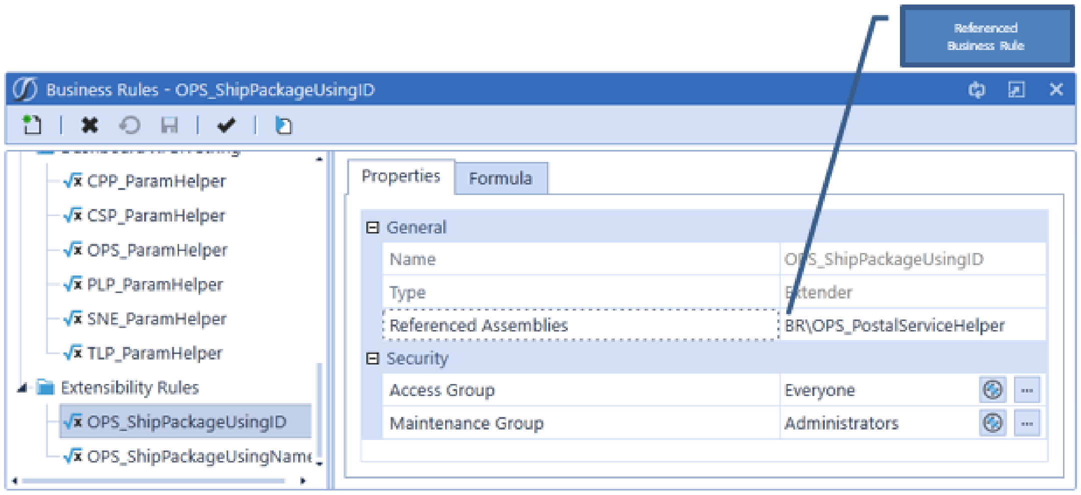
- Create a list of shared constant values
- Create a set of standard helper functions
- Centralize maintenance of shared logic

Reference Specification

This section defines the syntax required to reference a Shared Business Rule from another Shared or Item Specific Business Rule.

Shared Business Rules Referencing Other Shared Business Rules
In order to create a reference from one Shared Business Rule to another, navigate to the rule calling a Public Method of another Shared Business Rule and make a declaration in the *Referenced Assemblies* property. The syntax used to create a reference to another Shared Business Rule requires a *BR* prefix and the Business Rule name to reference.

Note: Reference more than one Business Rule by creating a comma-separated list of reference statements.



General	
Name	OPS_ShipPackageUsingID
Type	Extender
Referenced Assemblies	BR\OPS_PostalServiceHelper
Security	
Access Group	Everyone
Maintenance Group	Administrators

Syntax

BR\<Business Rule Name to Reference>

Example (Single Reference)

BR\OPS_PostalServiceHelper

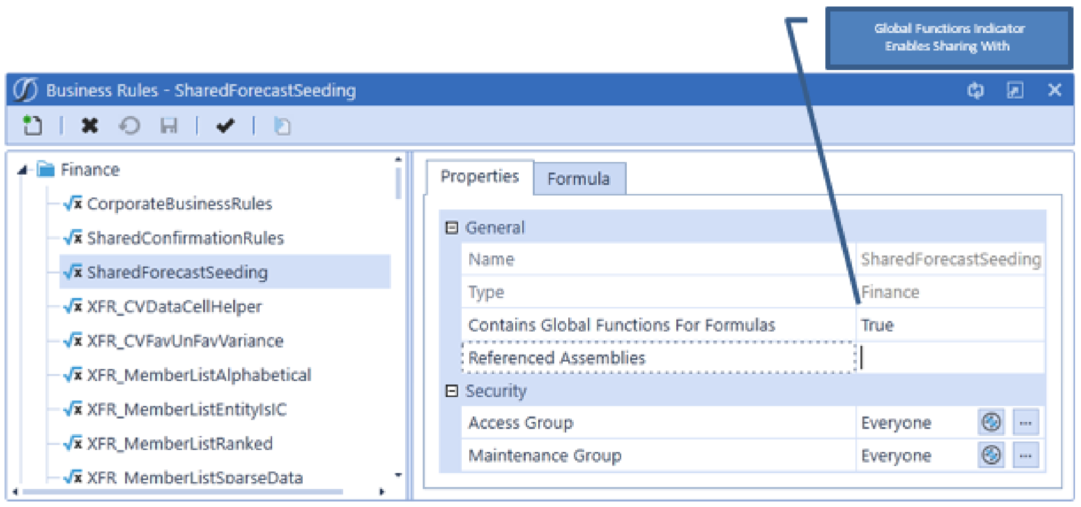
Example (Multiple References)

BR\OPS_PostalServiceHelper; BR\CPP_SolutionHelper

Item Specific Rules Referencing Shared Business Rules

Finance, Parser, ConditionalRule and DeriviativeRule Shared Business Rules all have equivalent Item Specific Business Rules. When creating a Shared Business Rule, set the *Contains Global Functions For Formulas* to *True* in order to make the Shared Business Rule available to Item Specific Business Rules. Setting this property makes the Shared Business Rule available without having to place a reference on the item using the rule. Item Specific Business Rules do not have a *Referenced Assemblies* property and therefore can only reference Shared Rules of the same engine type with the *Contains Global Functions For Formulas* property set to *True.*

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


Global Functions Indicator Enables Sharing With
Business Rules - SharedForecastSeeding
Properties
Formula

Finance
- CorporateBusinessRules
- SharedConfirmationRules
- SharedForecastSeeding
- XFR_CVDataCellHelper
- XFR_CVFavUnFavVariance
- XFR_MemberListAlphabetical
- XFR_MemberListEntityIsIC
- XFR_MemberListRanked
- XFR_MemberListSparseData

General	
Name	SharedForecastSeeding
Type	Finance
Contains Global Functions For Formulas	True
Referenced Assemblies	
Security	
Access Group	Everyone
Maintenance Group	Everyone

Note: When a Finance Business Rule has its *Contains Global Functions For Formulas* set to *True*, any changes made to the Business Rule causes a metadata status impact and changes the Calculation Status to *OK, MC*. This dependency must occur because a global rule can be used by a Member Formula calculation, and therefore can impact the status of the Finance Engine’s data *(analytic / Cube data).*

Code Declaration

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

Example Declaration

‘Declaring an object variable to reference a Shared Business Rule

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

Example Usage

‘Executing a Function on the Reference Business Rule Object Variable

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

Defining a Reference to an External .Net DLL

OneStream allows developers to build and reference their own custom Microsoft .Net DLLs from Shared Business Rules. These are written in either VB.Net or C#. Custom business logic can be encapsulated and protected within an external DLL written in Microsoft Visual Studio.

Common reasons to create a custom DLL referenced by a Business Rule:

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

DLL Installation and Configuration

This section defines the configuration steps that must be completed before an external DLL can be referenced within a Shared Business Rule. This is a three step process.

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

   This setup process is a best practice, but is not required. As an alternative, reference the external DLL from a folder located on each application server and any time the DLL is updated, it needs to be copied to a standard folder on each application server.

2. Identify or create the external DLL to be called and copy it to the BusinessRuleAssemblyFolder. When a Business Rule is executed and an external DLL reference containing the XF\ prefix is found in the Referenced Assemblies property of the rule, the application server will look in the BusinessRuleAssemblyFolder defined in the application server configuration file in order to find the DLL to be referenced.

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

Reference Specification

This section defines the syntax required to reference an external DLL by setting the Shared Business Rule's *Referenced Assemblies* property. There are three methods available for referencing an external DLL.

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

Syntax
XF\<External DLL Name to Reference>

Example (Single Reference)
XF\ExternalCode.DLL

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

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

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

Using a file system path reference is a valid method when referencing an external DLL that already exists on an application server. The DLL exists on the application server as part of the operating system or another installed software component.

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

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

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

Code Declaration

Once a reference is made to an External DLL from a Shared Business Rule, the Public Methods *(Functions / Subs)* of that External DLL can be called. In order to access the Shared Business Rule's Public Methods, declare an Import to the Namespaces defined by the DLL, and then create an instance of the desired class to utilize in the code.

Example Import

Imports YourNamespace.SubNamespace

Example Declaration

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

Dim extHelper As New YourClass

Example Usage

'Executing a Function on the external DLL

Dim desc As String extHelper.YourFunciton("SomeParameter")

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

Syntax
%System%\DLLName.DLL

Example
%userprofile%\documents\WindowsBase.DLL

API Structure and Organization

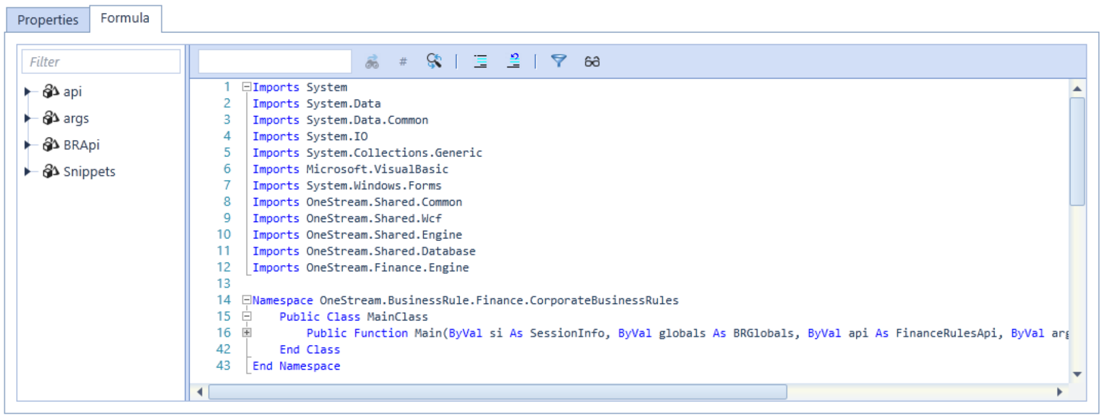
Namespaces

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

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

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

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



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

Namespace OneStream.BusinessRule.Finance.CorporateBusinessRules
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As FinanceRulesApi, ByVal arg
    End Class
End Namespace
```

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’)
```

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. OneStream provides a client tier API called from a custom developed client application. The client API is regularly used within the PowerShell script to perform automation tasks.

Client API

The OneStream Client API is intended to provide a set of methods that connect to the OneStream environment, request data via a Cube View, and execute a Data Management Sequence. At first glance, these three capabilities may seem limiting, but it is important to realize that a Data Management Sequence can contain any combination of Data Management Steps and these steps can consist of custom Business Rules. Client side developers can create Data Management Sequences that execute Business Rules to accomplish server side tasks. Developers can create sophisticated solutions that combine in-solution Business Rule logic with client side custom solution logic.

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.

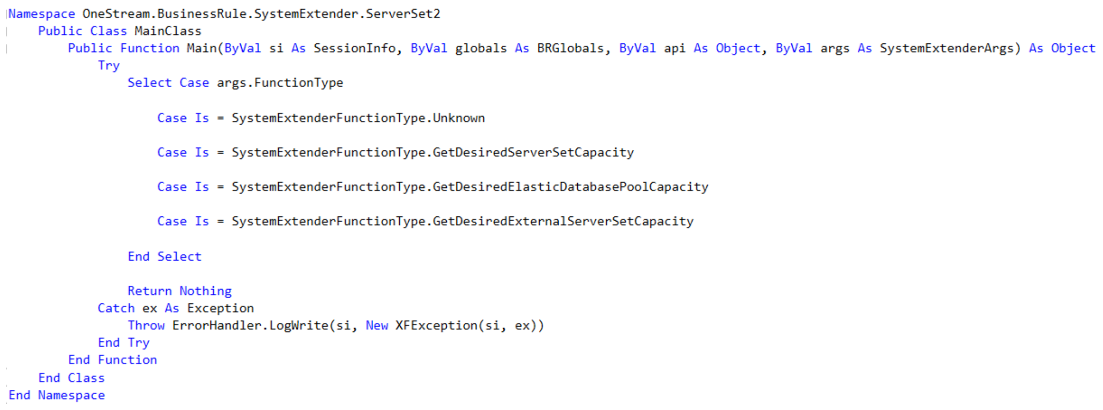
System Tools

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.

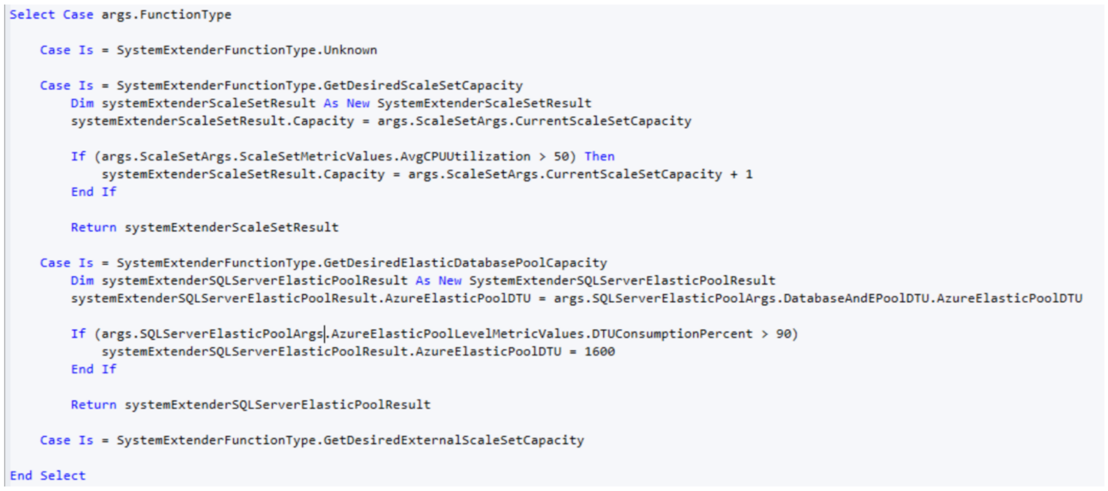
Refer to the *Installation and Configuration Guide* under *Azure Database Connection Settings* and *Server Sets* for where to refer to these Business Rules. Example starting point of empty System Extender Business Rule upon creation:



```
Namespace OneStream.BusinessRule.SystemExtender.ServerSet2
    Public Class MainClass
        Public Function Main(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As Object, ByVal args As SystemExtenderArgs) As Object
            Try
                Select Case args.FunctionType

                    Case Is = SystemExtenderFunctionType.Unknown

                    Case Is = SystemExtenderFunctionType.GetDesiredServerSetCapacity

                    Case Is = SystemExtenderFunctionType.GetDesiredElasticDatabasePoolCapacity

                    Case Is = SystemExtenderFunctionType.GetDesiredExternalServerSetCapacity

                End Select

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

Sample System Business Rule

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



```
Select Case args.FunctionType

    Case Is = SystemExtenderFunctionType.Unknown

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

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

        Return systemExtenderScaleSetResult

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

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

        Return systemExtenderSQLServerElasticPoolResult

    Case Is = SystemExtenderFunctionType.GetDesiredExternalScaleSetCapacity

End Select
```

Database

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

Tables

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

Tools

Database Tools allow System Administrators to manage the database.

Data Records

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

Client API Listing

This API provides a simple set of functions that have the ability to connect to OneStream's server, authenticate, execute OneStream Data Management Sequences, and perform basic data retrieval.

Client API Object Hierarchy

- OneStreamClientAPI
  - LogonInfo
    - Type: LogonInfo
  - **SI**
    - Type: SessionInfo
  - Authentication
    - Logon
      - Parameters:
        - string webServerUrl
        - string userName
        - string password
        - XFClientAuthenticationType clientAuthenticationType
      - Return Value:

        - LogonInfo

- Logoff
    - Parameters:
        - None
    - Return Value:
        - None

- OpenApplication
    - Parameters:
        - string application
    - Return Value:
        - LogonInfo

- LogonAndOpenApplication
    - Parameters:
        - string webServerUrl
        - string username
        - string password
        - string application
        - XFClientAuthenticationType clientAuthenticationType

            - Return Value:
                - LogonInfo

        - EncryptPassword
            - Parameters:
                - string clearTextPassword
                - XFClientAuthenticationType clientAuthenticationType
            - Return Value:
                - string

- DataManagement
    - ExecuteSequence
        - Parameters:
            - string sequenceName
            - string customSubstVarsAsCommaSeparatedPairs
        - Return Value:
            - DataMgmtResult

    - ExecuteStep
        - Parameters:
            - string dataMgmtGroupName
            - string stepName

        - string customSubstVarsAsCommaSeparatedPairs
    - Return Value:
        - DataMgmtResult

- DataProvider
    - GetAdoDataSetForCubeViewCommand
        - Parameters:
            - string cubeViewName
            - bool dataTablePerCubeViewRow
            - CubeViewDataTableOptions dataTableOptions
            - string resultDataTableName
            - Dictionary<string, string> customSubstVars
            - bool throwExceptionOnError
        - Return Value:
            - DataSet
    - GetAdoDataSetForSqlCommand
        - Parameters:
            - DbLocation dbLocation
            - string xfExternalDBConnectionName
            - string sqlQuery
            - string resultDataTableName

        - Dictionary<string, string> customSubstVars
        - bool throwExceptionOnError
    - Return Value:
        - DataSet

- GetAdoDataSetForMethodCommand
    - Parameters:
        - XFCommandMethodTypeId xfCommandMethodType
        - string methodQuery
        - string resultDataTableName
        - Dictionary<string, string> customSubstVars
        - bool throwExceptionOnError
    - Return Value:
        - DataSet

PowerShell

PowerShell is an object-oriented programming language and interactive command line shell for Microsoft Windows. It was designed to automate system tasks, such as batch processing, and create systems management tools for commonly implemented processes. PowerShell includes more than 130 standard command line tools for functions that formerly required users to create scripts in VB, VBScript or C#.

PowerShell offers a variety of ways to automate tasks which include:

Cmdlets
Very small .NET classes that appear as system commands

Scripts
Combinations of cmdlets and associated logic

Executables
Standalone tools

Instantiation of standard .NET classes

PowerShell integrates with the .NET environment and can also be embedded in other applications. Over one hundred cmdlets are included and can be used separately or combined with others to automate more complex tasks. Users can also create and share cmdlets.

PowerShell is built into Windows Operating Systems, where it is included as an optionally installed feature. In addition, the Windows Task Scheduler can be used to automate PowerShell script execution.

Using PowerShell Script Editor

To run PowerShell on Windows, Click left lower corner Windows icon start typing *PowerShell* and open to begin.

There are two programs used to interact with PowerShell.

Windows PowerShell ISE
This is the integrated scripting environment or Script Editor. The editor allows users to type PowerShell commands as well as edit and run PowerShell script files which are text files with a *ps1* extension.

Windows PowerShell
This program is a command line execution tool that looks like a DOS prompt. It allows a user to run a command or a script file, but it does not perform editing/creating scripts as well.

Configuring PowerShell to use the OneStream Client API

Before PowerShell can be used to interact with the OneStream client API, three configuration steps must be completed on each machine used for PowerShell script execution. First, execute a PowerShell command enabling the execution of unsigned scripts. Second, create or alter the PowerShell execution and IDE configuration files, so the script engine understands how to use the *.Net Framework v4.0*. Finally, OneStream Studio must be installed on each machine executing PowerShell scripts.

Allowing Execution of Unsigned Scripts
The first time this runs, the following line needs to run in a PowerShell command prompt. This will allow PowerShell to run unsigned scripts created on the local computer.

```
set-executionpolicy remotesigned
```

Configuration for .Net Framework v.4.0
In order to use the *OneStreamClientApi* with PowerShell, PowerShell needs to be configured to use the *.NET Framework v4.0*. In order to do this, modify or create two configuration files if they do not already exist.

Configuration File Folder
C:\Windows\System32\WindowsPowerShell\v1.0

File 1 (Config for Execution)
powershell.exe.config

File 2 (Config for IDE)
powershell_ise.exe.config

Required File Contents (Must be added to each configuration file)

```
<?xml version="1.0"?>
<configuration>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0.30319"/>
    <supportedRuntime version="v2.0.50727"/>
  </startup>
</configuration>
```

Refer to the following web resources for more information on this process.

http://stackoverflow.com/questions/2094694/how-can-i-run-powershell-with-the-net-4-runtime
http://tfl09.blogspot.com/2010/08/using-newer-versions-of-net-with.html.

Install OneStream Studio
The OneStream Studio product installation includes the Client API installation used by PowerShell scripts to interact with the OneStream server.

Learning PowerShell

Microsoft provides extensive resources to help IT professionals get the most out of PowerShell.

Refer to the following web resource in order to learn more about scripting with PowerShell.
http://technet.microsoft.com/en-us/scriptcenter/powershell.aspx

Using OneStream's Client API in a PowerShell Script

OneStream provides a client API (OneStreamClientApi) specifically designed to enable PowerShell scripts to call a OneStream function. This API exposes functions for authentication and Data Management. Over time, OneStream expanded the number of functions exposed to this API. The client API component is installed as part of the OneStream Studio installation.

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

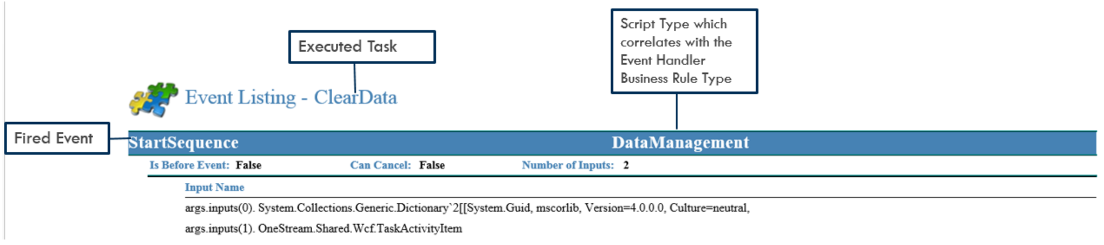
EndSequence

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.


Executed Task
Script Type which correlates with the Event Handler Business Rule Type
Fired Event

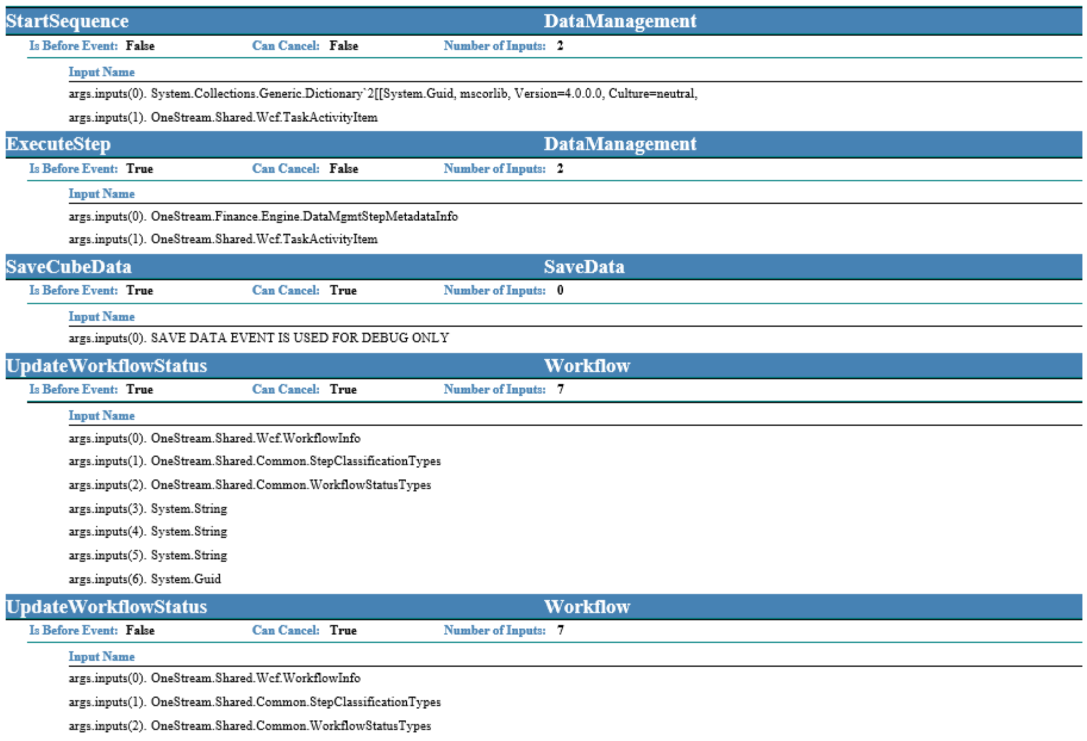
Event Listing - ClearData

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

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**

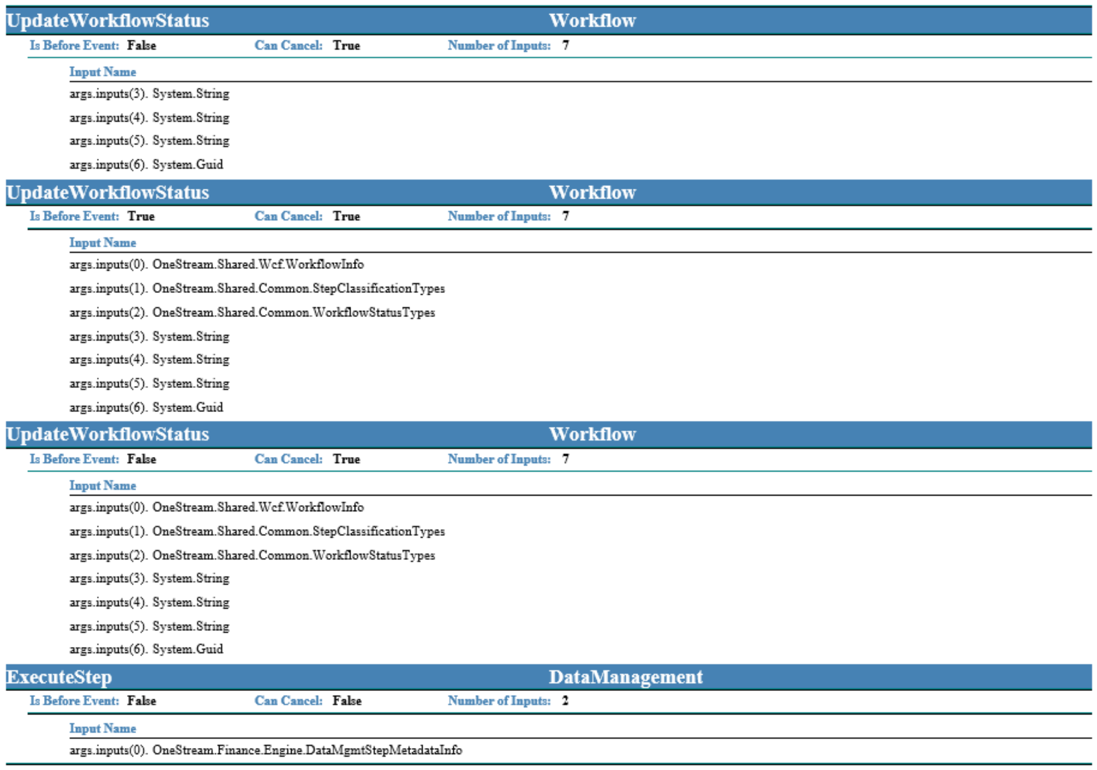
Is Before Event: False Can Cancel: True Number of Inputs: 7

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

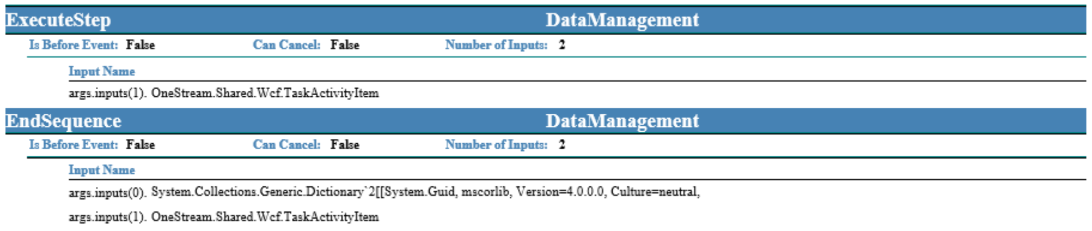
Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowInfo
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes
args.inputs(3). System.String
args.inputs(4). System.String
args.inputs(5). System.String
args.inputs(6). System.Guid

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



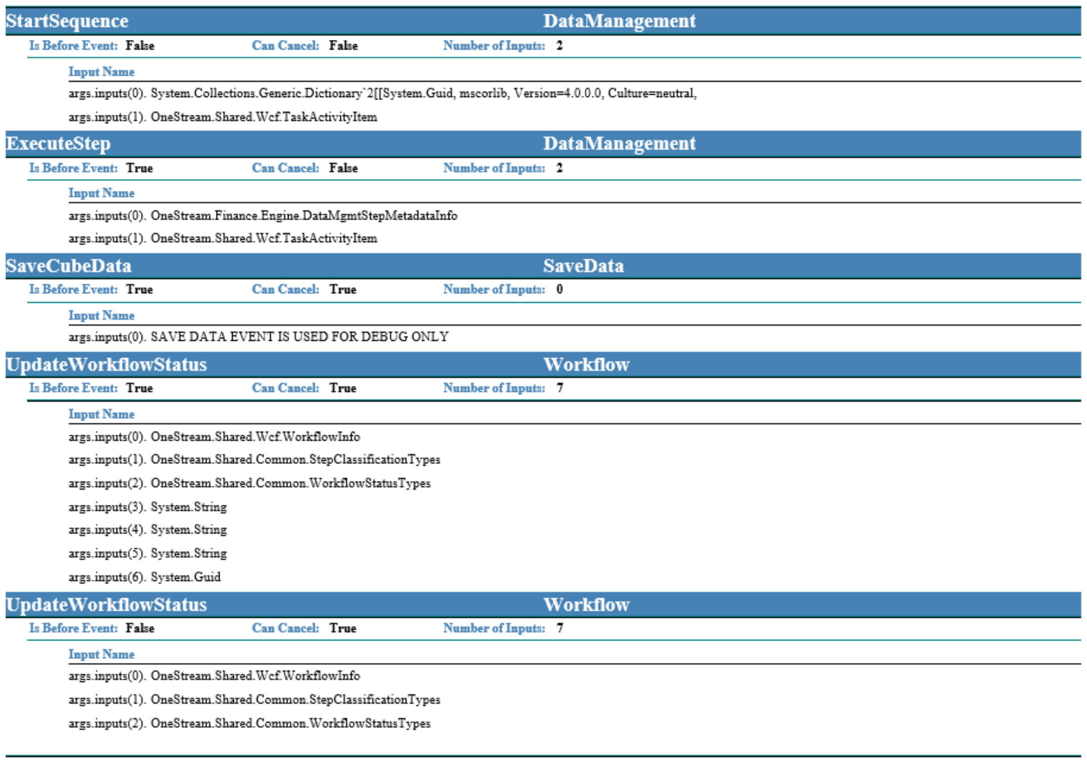
ExecuteStep	DataManagement	
Is Before Event: False	Can Cancel: False	Number of Inputs: 2

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**

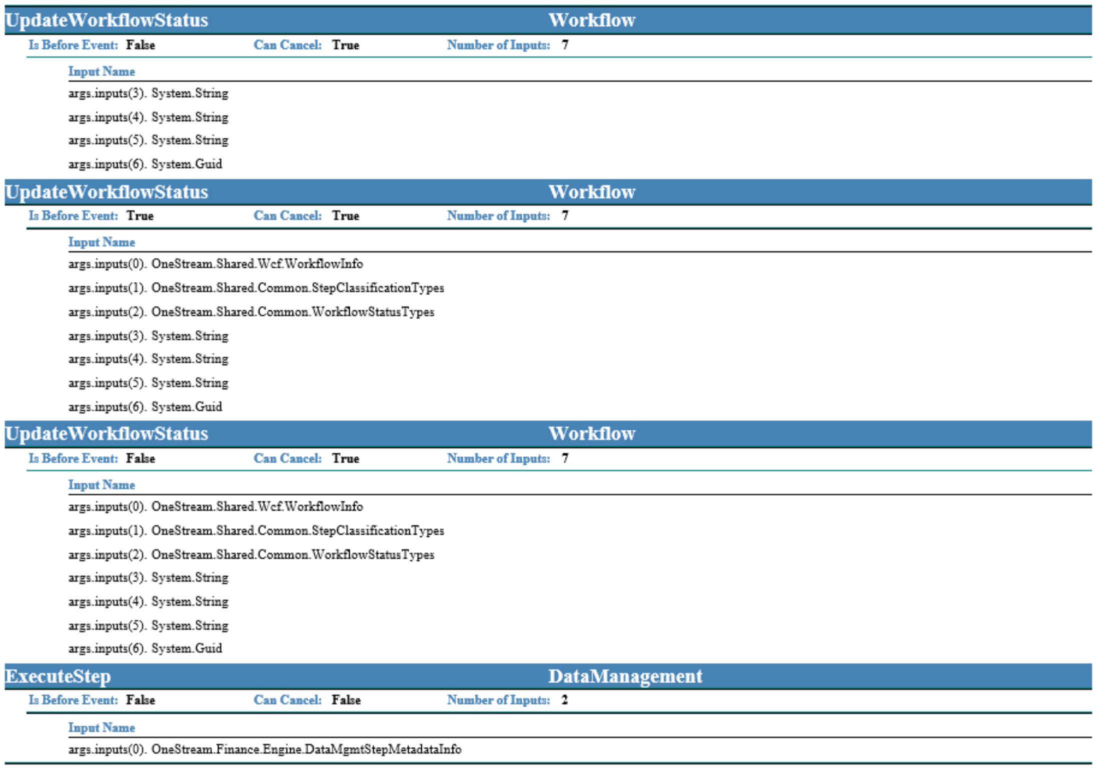
Is Before Event: True Can Cancel: True Number of Inputs: 7

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**

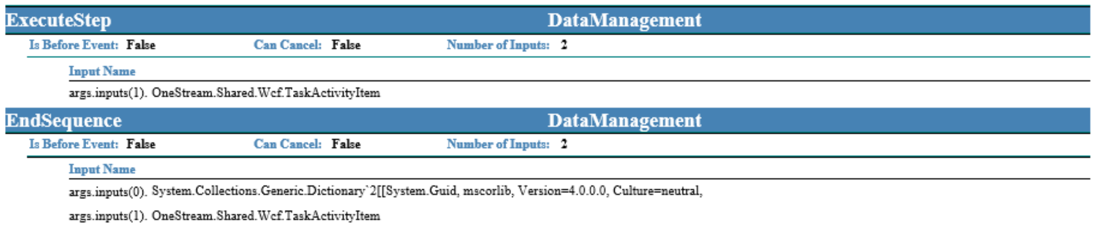
Is Before Event:	False	Can Cancel:	True	Number of Inputs:	7

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

args.inputs(3). System.Guid

ProcessTransformRules — **Transformation**

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

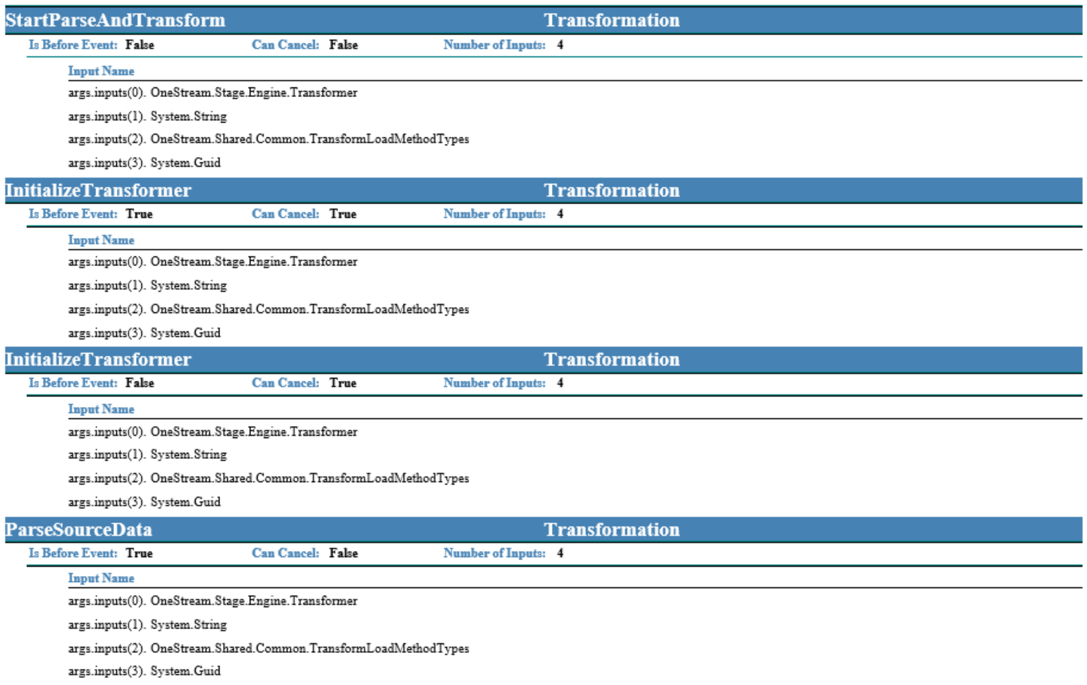
UpdateWorkflowStatus	Workflow	
Is Before Event: False	Can Cancel: True	Number of Inputs: 7

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

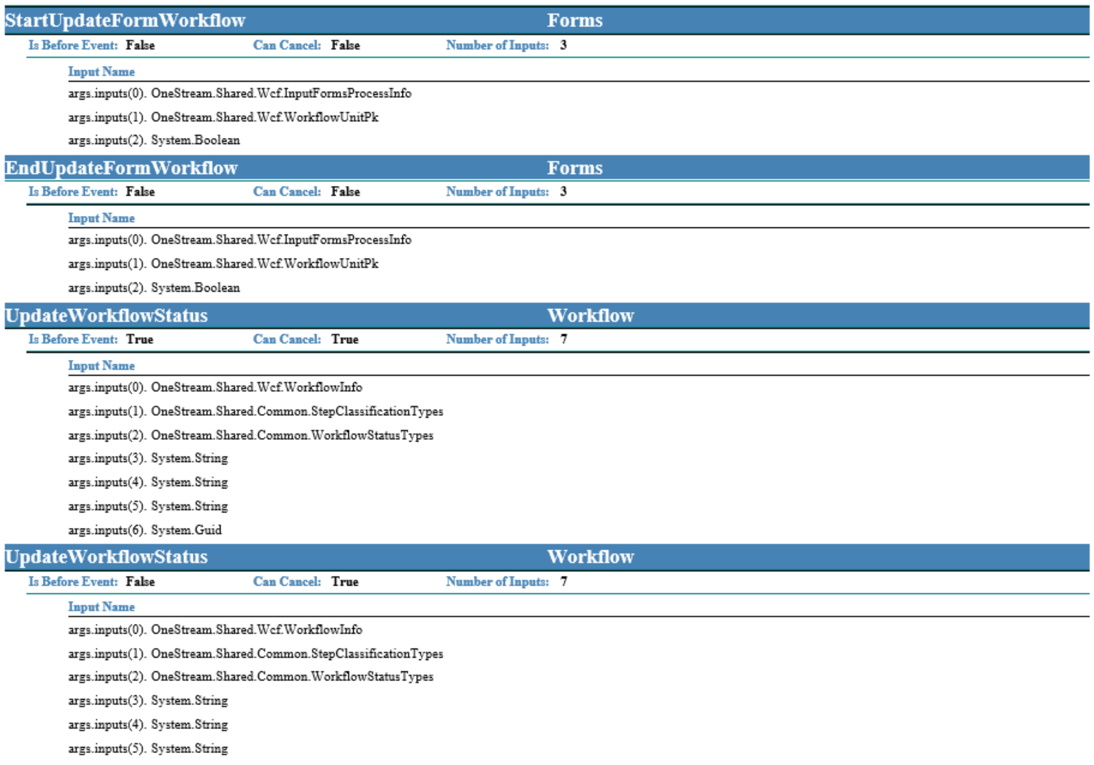
Input Name
args.inputs(0). OneStream.Shared.Wcf.XFFormEx
args.inputs(1). System.Boolean
args.inputs(2). System.Boolean
args.inputs(3). OneStream.Shared.Common.WorkflowStatusTypes

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

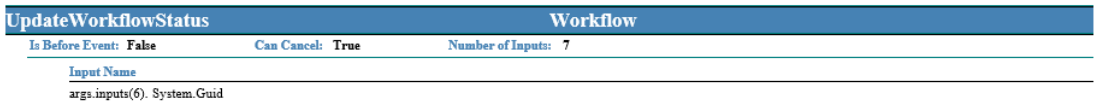
args.inputs(1). OneStream.Shared.Common.StepClassificationTypes

args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes

args.inputs(3). System.String

args.inputs(4). System.String

args.inputs(5). System.String

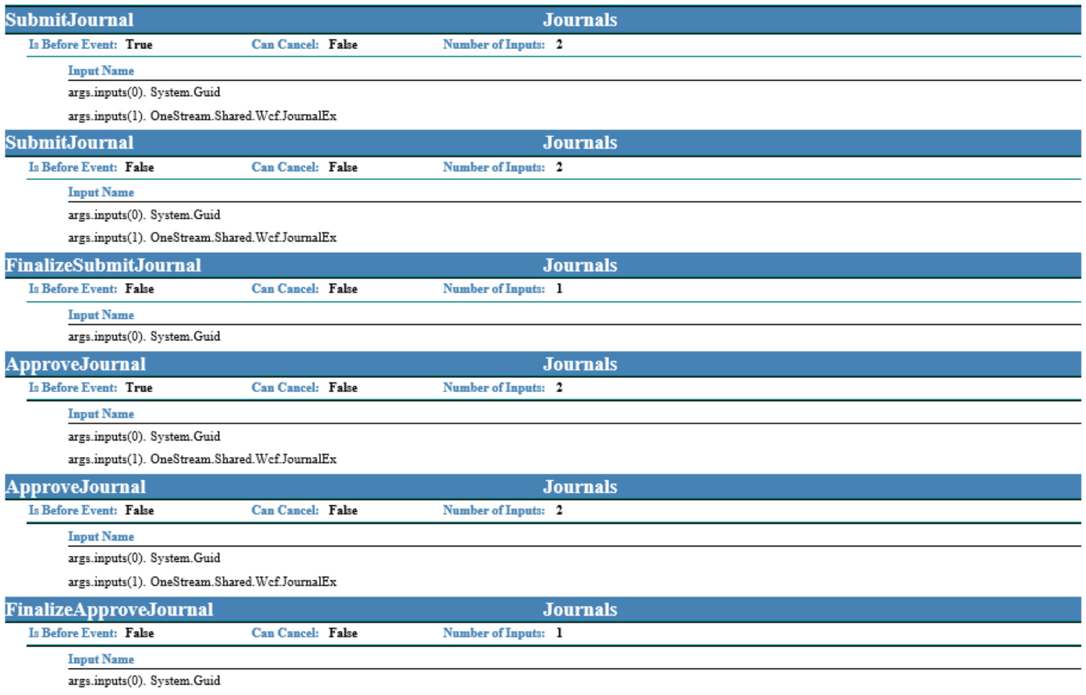


UpdateWorkflowStatus		Workflow	
Is Before Event: False	Can Cancel: True	Number of Inputs: 7	

Input Name

args.inputs(6). System.Guid

Process Journal



SubmitJournal — **Journals**

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

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

SubmitJournal — **Journals**

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

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

FinalizeSubmitJournal — **Journals**

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

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

ApproveJournal — **Journals**

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

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

ApproveJournal — **Journals**

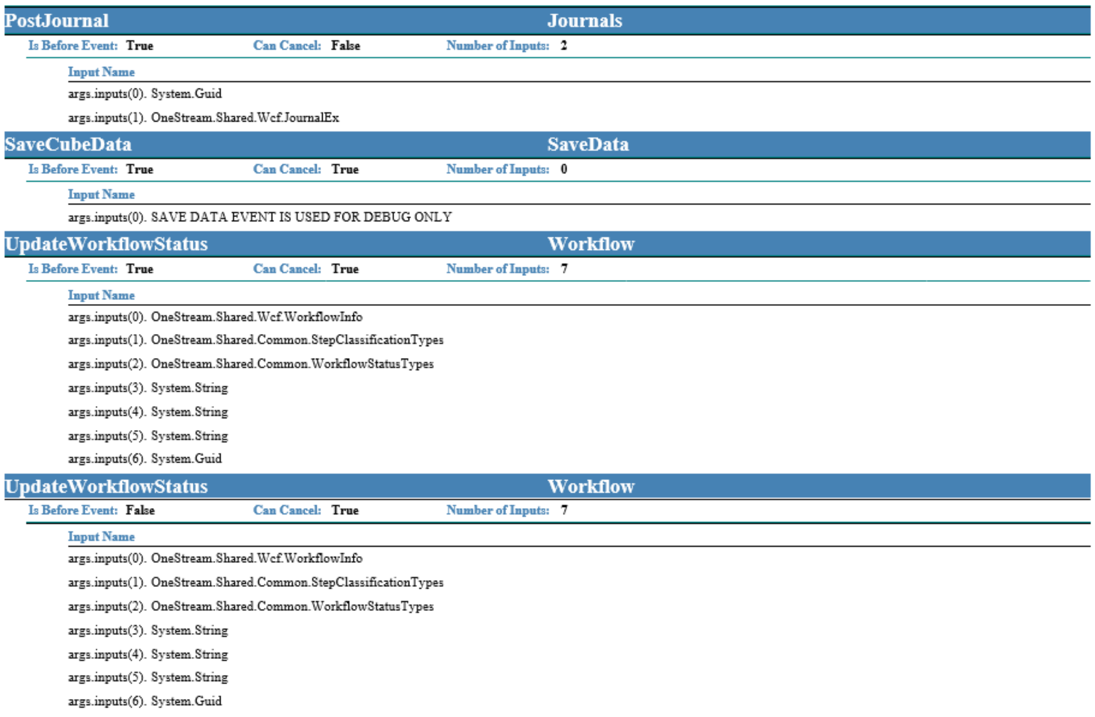
Is Before Event:	Can Cancel:	Number of Inputs:
False	False	2

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

FinalizeApproveJournal — **Journals**

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

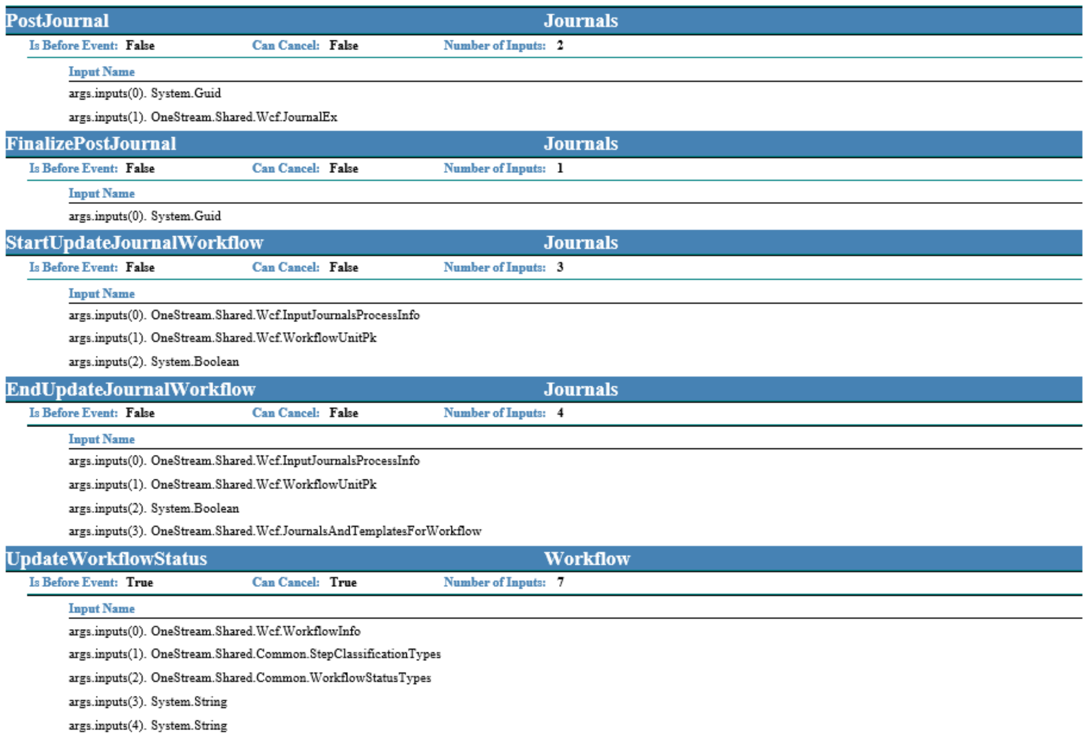
Is Before Event: **True** Can Cancel: **True** Number of Inputs: **7**

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

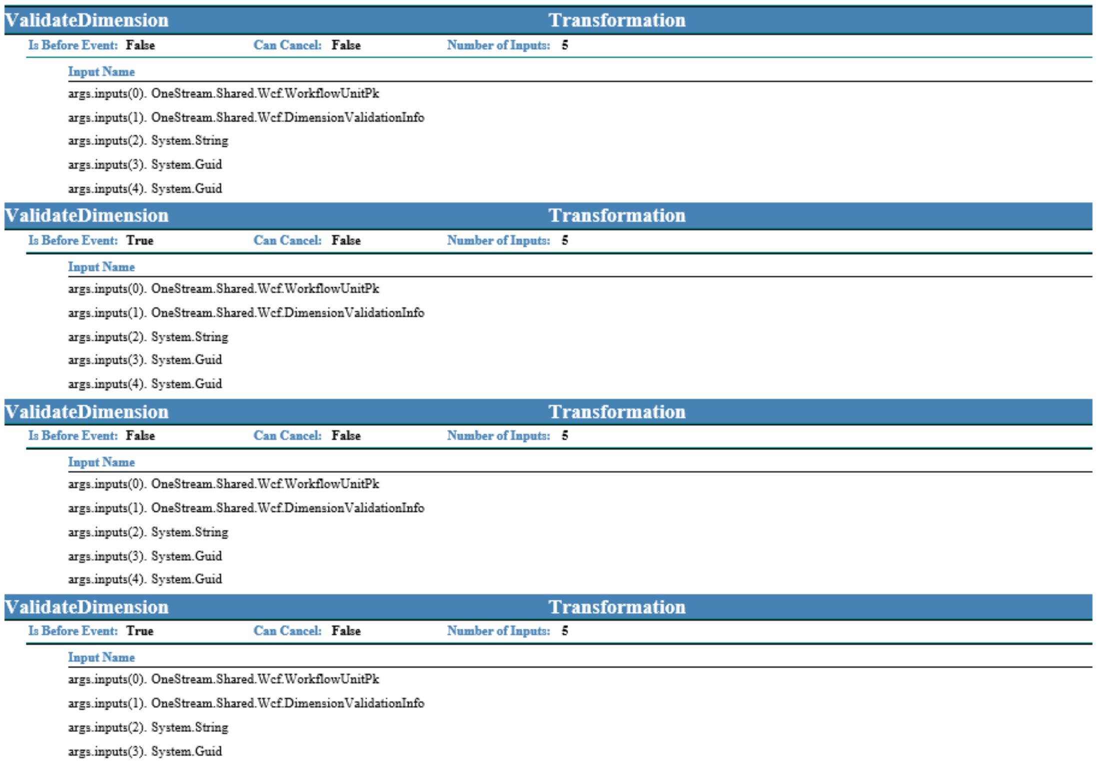
Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

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**

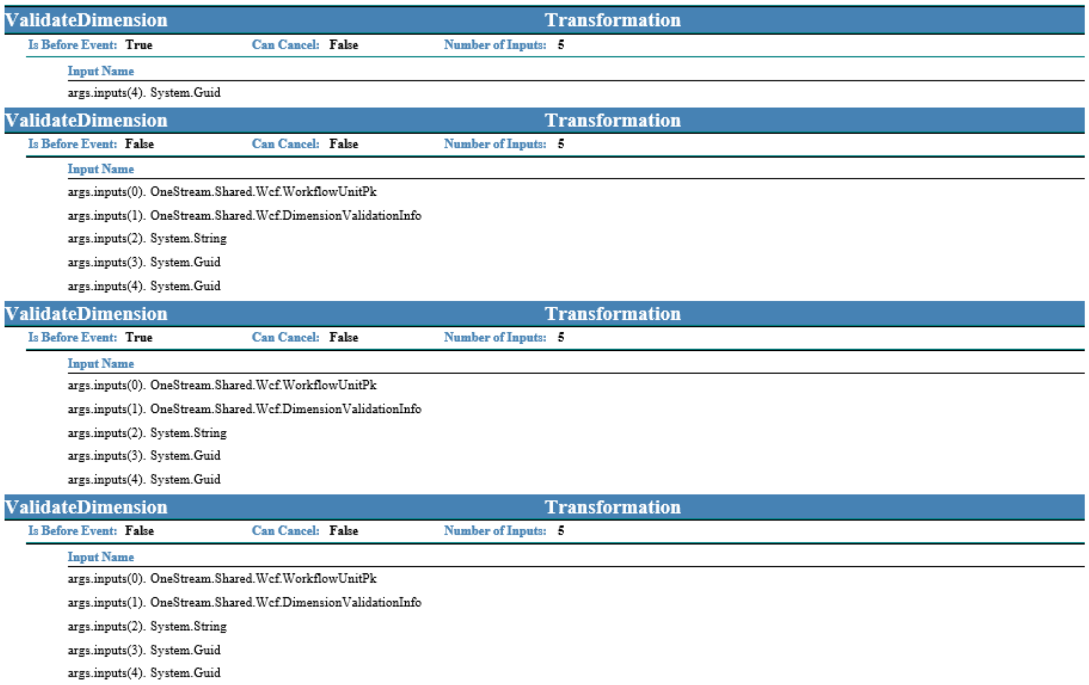
Is Before Event: False Can Cancel: False Number of Inputs: 5

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**

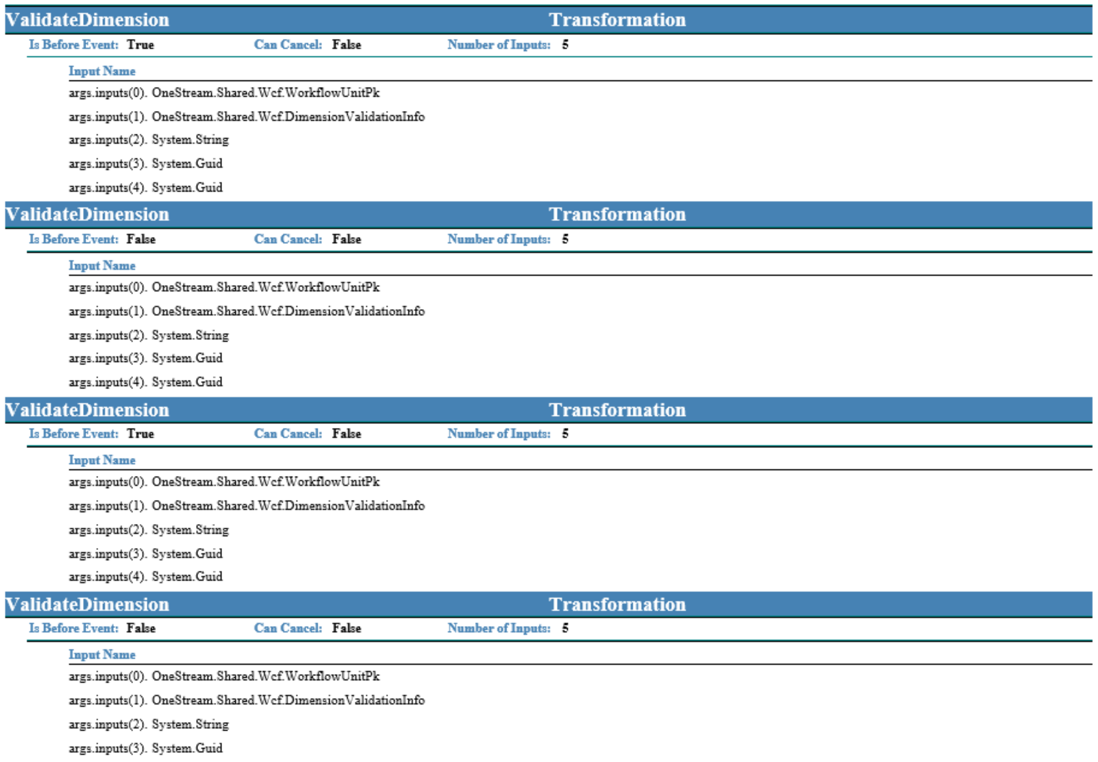
Is Before Event: True Can Cancel: False Number of Inputs: 5

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

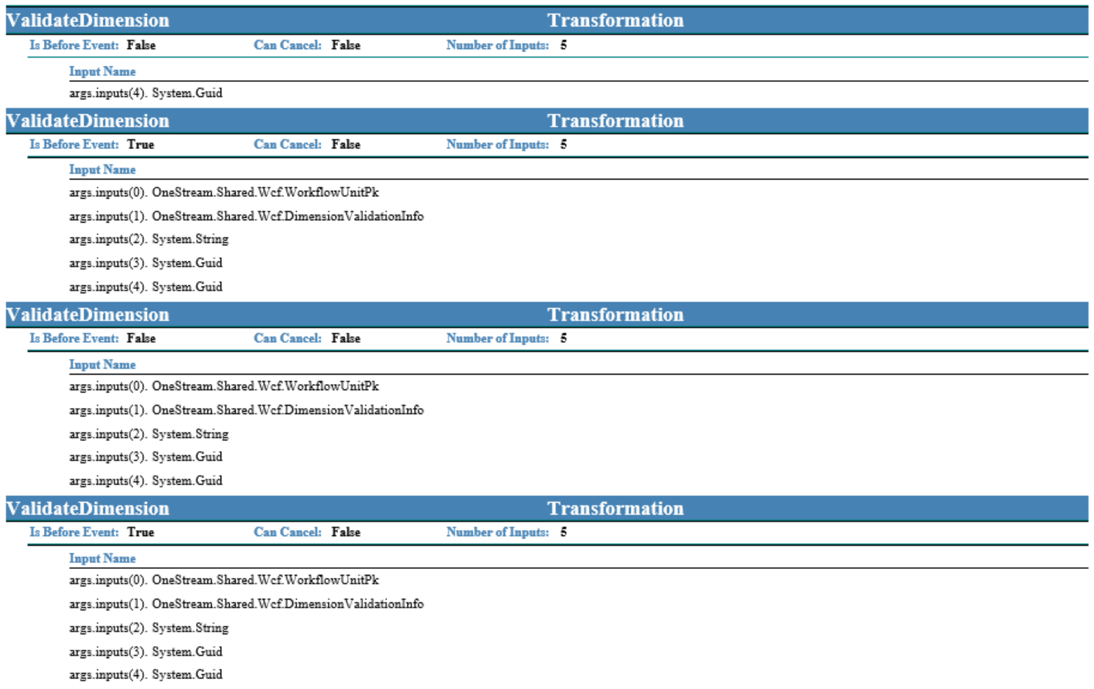
Input Name
args.inputs(0). OneStream.Shared.Wcf.WorkflowUnitPk
args.inputs(1). OneStream.Shared.Wcf.DimensionValidationInfo
args.inputs(2). System.String
args.inputs(3). System.Guid
args.inputs(4). System.Guid

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

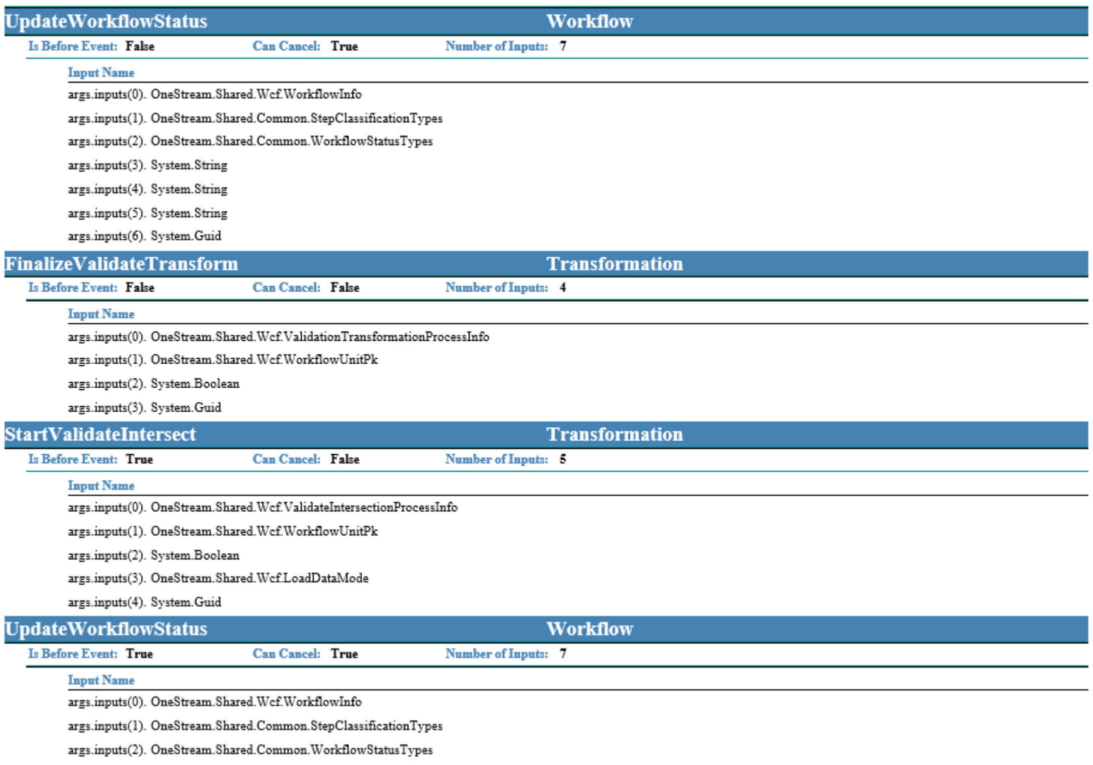
args.inputs(2). OneStream.Shared.Common.WorkflowStatusTypes

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

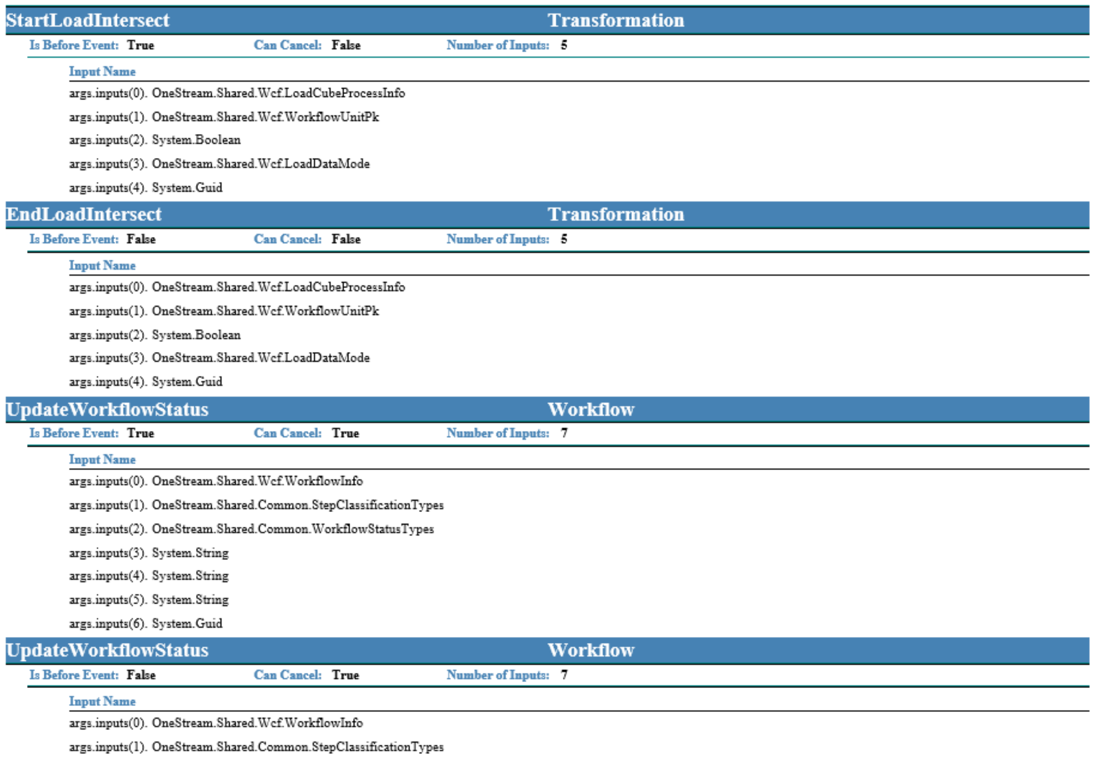
UpdateWorkflowStatus		Workflow	
Is Before Event: False	Can Cancel: True	Number of Inputs: 7	

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

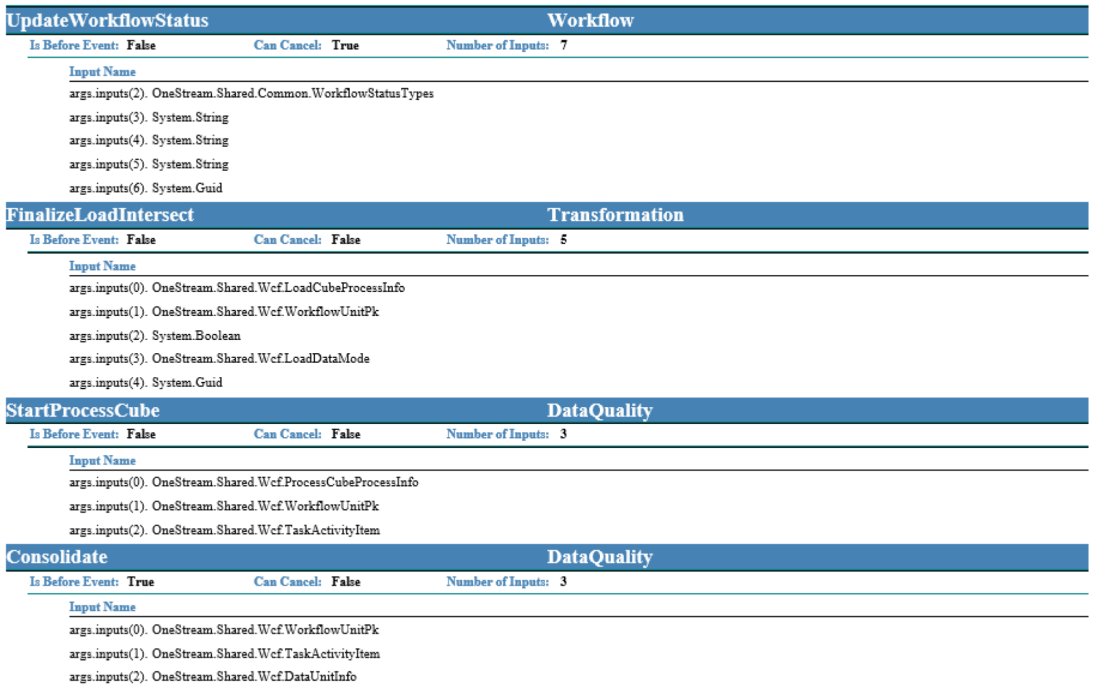
Input Name

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