

Data Collection Guide

9.4.1 Release

Copyright © 2026 OneStream Software LLC. All rights reserved.

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

Table of Contents

Data Collection Overview 1
Key Tenets 1
Data Sources and Integration 1
Mapping and Transformation 2
Workflow Processing 2
Data Quality and Control 2
User Input Channels 3
Extensibility and Automation 3
Data Collection Based on Role 3
Administrators 4
Implementers 4
End Users 5
Data Collection 6
Data Sources 6

Data Source Properties 7
Fixed Files 8
Delimited Files 9
Data Sources 10
Connectors 10
Connector Data Source 11
Data Management Export Sequences Data Source 38
Source Dimensions 51
Source Dimension Properties 51
Matrix Settings 56
Numeric Settings 56
Using Excel as a Data Source 57
Long Running Requests 57
Batch File Loading 57
Setting Up and Using Batch File Loading 58
Batch File Name Format Specification 59

Workflow Stage Import Methods 61
Standard Import Methods 62
Direct Import Methods 62
Blend Import Methods 62
Stored versus In-Memory Workflow Imports 63
Guidelines on Volumes and Limits 63
Standard Workflow Record Analysis 64
Direct Workflow Record Analysis 65
Blend Record Analysis 65
In-Memory Workflow Imports 65
Blend Workflow Import 71
Transformation Rules 73
Transformation Rules Toolbar 73
General Settings 74
General 74
Security 75

Settings 75
Mapping Types 75
One-to-One 76
Composite 76
Range 78
List 79
Mask 79
Derivative Rules 81
Create Derivative Rules 83
Derivative Rule Properties 83
Sequencing 85
Derivative Rule Expression 87
Logical Operators 95
Lookup 99
Transformation Rule Profiles 99

Forms 100
Forms Channel Workflow 100
Form Allocations 101
Advanced Distribution 101
Form Templates 107
Applying Literal Value Parameters to Form Templates 107
Form Template Profiles 109
Security 110
Form Template Properties 110
Form Template Group Properties 116
Managing Form Templates 116
Journals Overview 118
Create a New Journal 119
Create Journal 119
Create Journal Using Selected Template 121
Create Journal Using File 122

Open a Journal 123

Bulk Selection of Time Periods for Copying 125

Rename a Journal 126

Delete a Journal 127

Loading Data into Journals 128

Loading Journal Data 129

Filter Journals 138

Process a Journal 139

Validate a Journal 140

Reject a Journal 140

Post a Journal 141

Unpost a Journal 142

Central Workflow Processing 142

Recurring Journals 145

Create a Standard Recurring Journal 146

Final Recurring Period 147

Journal Templates 147
Journal Template Properties 148
Journal Template Group Properties 151
Security 151
Loading Data With Excel Templates or CSV Files 153
Loading Stage Data 153
Import Excel Template 153
Header Abbreviations 156
Import Data Extracted via Data Management 156
Loading Form Data 157
Form Excel Template 157
Loading Form Details via Workflow 163
Loading Form Details via Business Rules 164
Loading Journal Data 164
Journal Excel Template 164
Header Abbreviations 167

Journal CSV Template 168

Loading Journal Details via Workflow 170

Extracting/Loading Journal Details via BRApi 171

Loading Cell Detail 173

Cell Detail Excel Template 173

Cell Detail CSV Template 176

Loading Cell Detail via Workflow 177

Extracting/Loading Cell Detail via Business Rules 178

Exporting Data Attachments 179

Exporting Data Attachments via Business Rule 179

Exporting Data Attachment Text 179

Loading Excel Templates to Custom Tables 180

Load Method Definitions 180

Importing Text-based Data 184

Data Collection Overview

Data collection in OneStream is the framework used to bring data into the platform, structure it correctly, and make it ready for reporting, analysis, and financial processes. It includes everything required to obtain data from external systems, transform it into the OneStream model, validate it through the workflow, and adjust data where needed.

At a high level, data collection answers four key questions:

- Where does data come from? (files, systems, or user input)
- How is data interpreted? (data sources and parsing rules)
- How is data aligned to the model? (mapping and transformation)
- How is data processed and governed? (workflow, validation, and security)

Data collection represents a coordinated set of capabilities that work together to ensure data is accurate, controlled, and usable across the platform.

Key Tenets

Data collection provides capabilities in the areas outlined below.

Data Sources and Integration

Data collection begins with defining data sources, which act as blueprints for imports. These define the structure of incoming data and decide how it should be parsed. Data can be loaded from:

- Fixed or delimited files
- Excel and CSV templates
- Direct system integrations (connectors)
- Data management processes

Mapping and Transformation

After data is acquired, it must be aligned to the application model. Transformation rules map source values to target dimension members, while derivative rules apply logic to enrich or generate additional records.

Workflow Processing

Data is processed through workflow using different methods depending on your business needs. The three methods are:

- Standard (auditable and stored)
- Direct (high-performance, in-memory)
- Blend (analytic and relational reporting)

Workflow ensures that data is validated, traceable, and controlled throughout the process.

Data Quality and Control

Built-in controls ensure data consistency and reliability, including:

- Numeric formatting and standardization
- Row-level filtering and bypass logic
- Validation and error handling
- Security and access control

User Input Channels

Data collection includes manual data entry through:

- Forms for structured operational input
- Journals for financial adjustments and transactions

These inputs follow the same workflow and validation processes as system-loaded data.

Extensibility and Automation

Data collection supports automation and customization through:

- Excel and CSV templates
- Business Rules (BRApi)
- Batch file loading
- Custom table integration

This flexibility enables organizations to scale from simple imports to fully automated enterprise integrations.

Data Collection Based on Role

Data collection can change based on your role within your organization.

Administrators

Administrators are responsible for governing and securing data collection across the application. Responsibilities include:

- Configuring and maintaining data sources
- Managing security for data sources, rules, forms, and journals
- Monitoring workflow processing and performance
- Enforcing data governance and auditing standards
- Optimizing data volume, partitioning, and batch processing

Administrators focus on control, security, performance, and reliability of the data collection framework.

Implementers

Implementers design and build the data integration and transformation logic that power data collection. Responsibilities include:

- Defining data sources and structure (fixed, delimited, connectors)
- Configuring source dimensions and parsing logic
- Building transformation and derivative rules
- Designing workflows (standard, direct, blend)
- Developing connectors, business rules, and automation processes

Implementers focus on data integration design, mapping accuracy, and scalable architecture.

End Users

End users interact with data collection primarily through workflow, forms, and journals. Responsibilities include:

- Loading or submitting data using provided templates or workflows
- Entering data through forms
- Creating and managing journals for adjustments
- Validating, reviewing, and correcting data where needed

End users focus on data entry, validation, and completing assigned workflow tasks.

Data Collection

Data sources are built to act as a blueprint for the types of imports that need to be done and to define how the data should be parsed and imported. These include fixed files and delimited files, which use connectors to pull the data directly from the source. Data sources also have dimensions associated with them along with specific properties. In this section you will learn about these data sources and how to leverage them to build your data collection.

> **TIP:** OneStream recommends that any credentialed integration with the OneStream platform should change passwords regularly.

Data Sources

Data sources are blueprints of the types of imports required and define how to parse and import data. Data sources can be a fixed file, delimited file, connectors file, or a Data Management Export sequence that pull data from a source system. Data sources also have dimensions associated with them along with specific properties.

Once built, you can assign a data source to one workflow profile or to many workflow profiles sharing a common file format.

Associate a source file with a data source by selecting the  icon in the upper toolbar. The file opens in the top area of the screen and you can select fields and functions to build Data Source dimension definitions.

Data Source Properties

General properties are standard across all data types.

Name
The name of the data source.

Description
The field for a detailed description of the data source.

Security

The security properties are standard across all data types.

Access Group
Members of the assigned group have the authority to access the data source.

Maintenance Group
Members of the assigned group have the authority to maintain the data source.

> **NOTE:** Select the ellipses  icon and begin typing the name of the Security group in the blank field. As the first few letters are typed, the groups are filtered making it easier to find and select the desired group. Once the group is selected, click CTRL double-click. This will enter the correct name into the appropriate field.

Settings

Settings are standard across all data types.

Cube Name
The cube associated with this data source which will dictate the available dimensions that can be used.

Scenario Type
This allows the profile to be assigned to a specific scenario type or all scenario types. If the data source is assigned to a specific Scenario Type, it will only be available when assigned to the Workflow Profile.

Data Structure Settings

Data structure settings are standard across all data types.

Type
This defines the source file structure. The Type can be Fixed, Delimited, Connector, or Data Management Export Sequence. Details on these types can be found below.

Data Structure Type

Tabular
This will have a line or lines specific to a single intersection with one amount.

Matrix
This will have multiple amounts on a given line using rows and columns to determine the intersection that corresponds to each amount.

Allow Dynamic Excel Loads
If set to True, users will be able to load Excel templates as well as the data file for which the data source has been built. See the section on Using Excel as a data source for information on the proper formatting of these files.

Fixed Files

Fixed files are in a column format with data in predefined columns.

Connector Settings

Connector Name
A drop-down menu of available Connector Type business rules. The rules will be built containing the code to connect to and pull data from a given source system or database.

Delimited Files

Delimited files have fields that are separated by a common character.

Delimited File Settings

Delimiter
The character being used to separate the distinct fields in the source file is defined here.

Quote Character
In delimited files, the fields are often put in quotes in case the delimiter is also a valid character in one of the field members. This option specifies what quote character is being used in the source file.

Connector Settings

Connector Name
A drop-down menuof available Connector Type business rules. The rules will be built containing the code to connect to and pull data from a given source system or database.

Data Sources

Delimited Data Source

A Delimited Data Source with a separate column for debit and credit for the Amount Dimension is accomplished by using a Parser Complex Expression, or a business rule. First, assign the Debit column as the Amount. Next, create a Parser Complex Expression if this is a one-time occurrence, or create a business rule if it can be applied to numerous Data Sources, and assign it to the Amount Column. In the Complex Expression or business rule, check the value returned for the debit. If it is empty, or 0.00, refer to the credit value with the method outlined below.

Assessing the credit value is possible in OneStream because the transformation engine provides an array list of all fields in the line. In the example below, debit is column six, and credit is column 7.

Dim creditValue as String = api.Parser.DelimitedParsedValues(6)
This is a zero based list, so index six is equal to column seven in the Data Source. Using this expression would only return the credit value, so apply the statement above to a broader Complex Expression.

Connectors

Connectors pull data directly from a source system or database. See for more information.

Connector Settings

Connector Name
A drop-down menu of available Connector Type business rules. The rules will be built containing the code to connect to and pull data from a given source system or database.

Connector Uses Files
Set to True to process files that cannot be parsed using a data source. For example, if an Excel or text file has complex formatting, this setting allows you to code the way the file should be imported rather than defining it with a data source. See "Business Rules" in Application Tools.

Connector Data Source

OneStream can connect and import data from any external system using direct database connection to the external system. This means data can be imported and processed all the way through the Workflow certification process without ever having to use a source system extract file.

Connector Settings

Connector Name
A drop-down menu of available Connector Type business rules. The rules will be built containing the code to connect to and pull data from a given source system or database.

Connector Uses Files
Set to True to process files that cannot be parsed using a data source. For example, if an Excel or text file has complex formatting, this setting allows you to code the way the file should be imported rather than defining it with a data source.

Connector Business Rule

A Connector Business Rule defines the connection, data result sets, and drill-back option capabilities of an external data connection. A Connector functions as a Business Rule called by a Data Source and reveals what information is required from an external system. See Business Rules for an example of this rule.

Connector Data Source

Fields from the external data query results are mapped to Dimensions creating a processing behavior similar to the behavior of a Delimited File. Using this mapping process enables a Connector Data Source to use all the same built processing capabilities available with file-based Data Sources. This capability enables the design of an external data Connector to be entirely focused on connecting to and reading data from an external source instead of focusing on integrating complex business logic. The specific business logic can be added to the Data Source Dimensions in the form of a Complex Expression or Business Rule. This design methodology will help with writing the Connector Business Rule in a way that requires very little maintenance by business users.

Connector Information Request Types

GetFieldList

This is called by the Data Source designer screen when the user selects a Connector Data Source or one of its defined Dimensions. A list of available fields in the external Data Source will be visible as a list of Vb.Net Strings [List(Of String)] is requested.

GetData

This is called by the Import Workflow task when the Load and Transform button is clicked. The execution of a data query(s) that retrieves the row values for the chosen Workflow Unit is requested.

Fields

The field names returned by this query must match the field names returned by the GetFieldList request.

Where Clause

Typically the active Workflow Unit Time and Scenario values are converted to equivalent criteria values for the Time or Scenario of the external system.

Data Volume
Consider loading summarized data rather than full transaction system data replication because drill back is provided for more detailed values.

GetDrillBackTypes
Drill Back types can deliver results based on the different visualization types. This is called when a user double-clicks or right-clicks and selects Drill Back from a row in the source data load or drill down screens. A set of supported drill-back options to present to the end user as a list of DrillBackTypeInfo objects [List(Of DrillBackTypeInfo)] is requested. Drill Back types provide the Connector designer with the power to provide the end user with a menu list of drill back options.

DataGrid
This presents a grid of data rows to the end user.

TextMessage
This presents a text message to the end user.

WebUrl
This presents a website or custom HTML web content to the end user.

WebUrlPopOutDefaultBrowser
Opens a website or custom HTML web content in an external browser. From the Stage Import data grid, right-click on a data record, and select Drill Back. A dialog presents a menu of pre-configured Drill-back options. When you choose WebUrlPopOutDefaultBrowser, a standard browser session is launched, and you go to a web page based on variables.

FileViewer
This presents file contents to the end user from one of three locations.

FileShareFile
A file located in a folder in the OneStream File Share.

AppDBFile
A file stored in an application database.

SysDBFile
A file stored in a framework (System) database.

GetDrillBack
This is called when a user selects a specific Drill Back type presented by the GetDrillBackTypes request. When this action is executed, the Business Rule arguments will contain a reference to the DrillBackTypeInfo object the user selected which allows the Connector designer to determine how to get proper information to display for the DrillBackTypeInfo.

Connector Integration Prerequisites

The following items provide an overview of the major technology components involved in integrating external systems with deployment.

Determine Source System Inventory
The first step in integrating various source systems is to determine all the ones needed. This includes:

Source System Location & Identification

Database Type and Source System
Oracle, SQL, DB2, Syteline, Newstar, Lawson, PeopleSoft, Access, MAS500, etc.

> **NOTE:** The requirement for Oracle Database integrations is that all Oracle Source System TNS Profile details need to be in place on each of the OneStream application servers.

Data Query Method
Detailed Data Query, Data View, Stored Procedure, etc.

Source System Drill Back Criteria (if required)
Detailed Data Query, Data View, Stored Procedure, etc.

Source System Direct Access Credentials

A read-only type of access needs to be granted for the user account because the data from these external systems will be read. The read-only access should be granted against the productions instance of the data source as the data queries will be used to tie out data and do not present any risk to the source system themselves.

Source System 64-bit Client Data Provider

OneStream is a Microsoft .NET application with a 64-bit architecture. To communicate with any source system, a 64-bit source system client data provider needs to be available and installed on each OneStream application server. The source system's client data provider is what gives the ability to make an OLEDB or ODBC connection to the system.

Determine Connection String

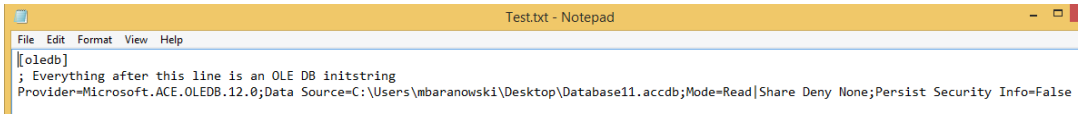
A connection string specifies information about a data source and the means of connecting to it. It is passed in code to an underlying driver or provider to initiate the connection. Whilst commonly used for a database connection, the data source could also be a spreadsheet or text file. The connection string may include attributes such as the name of the driver, server and database, as well as security information such as user name and password.

Create a Connection String from the OneStream Application Server

1. Right-click the Desktop icon of the OneStream application server and select **New** > **Document**.
2. Name the document and change the file extension from txt to udl.

   This creates a Data Link File to assist in the formation of the source system connection string.
3. Determine the DB Provider that the GL Source System is using. For example, SQLor Oracle.

4. Determine the server name where the data resides for the GL Source System.
5. Determine the user name and password used to connect to the server for the GL Source System.
6. Determine the database name on the server where the GL Source System data resides.
7. Save the completed UDL file and then rename the extension back to txt from udl.
8. Open the text file to see the connection string provided.



Example Connection Strings

SQL Server

Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=DBName;Data Source=SQLSERVERNAME

ORACLE (11i or R12)

Provider=OraOLEDB.Oracle.1;Password=<xxxxx>;Persist Security Info=True;User ID=<username>;Data Source=frepro.world

DB2

Provider=IBMDA400.DataSource.1;Password=<xxxxx>;Persist Security Info=True;User ID=OSuser;Data Source=HUTCH400;Use SQL Packages=True

MS Access

Provider=Microsoft.ACE.OLEDB.12.0;Data Source=\\UNCFileShare\DB1.accdb;Mode=Read|Share Deny None;Persist Security Info=False

Determine the Data Query Method

To extract data from any source system, the data query method and facility need to be determined. Data can be queried through a SQL Query, a SQL View, or Stored Procedure. OneStream executes this request against the source system using the defined source system connection string and processes the returned results within OneStream.

For example, if directly pulling in Trial Balance Data is required, then the detailed query that currently makes up the existing Trial Balance Report would be necessary for OneStream to pull the same data.

SQL Query
A SQL Query can be broken down into numerous elements, each beginning with a keyword. Although it is not necessary, a common convention is to write these keywords in all capital letters. The standard sections of a SQL Query are made up of the following four elements:

- SELECT
- FROM
- WHERE
- ORDER BY

The example below is a SQL Query used to pull Trial Balance Data from several different tables in an Oracle Database:

```
SELECT
GL_SETS_OF_BOOKS.NAME
,GL_BALANCES.ACTUAL_FLAG
,GL_BALANCES.PERIOD_NAME
,GL_BALANCES.PERIOD_NUM
,GL_BALANCES.PERIOD_YEAR
,GL_CODE_COMBINATIONS.CODE_COMBINATION_ID
,GL_CODE_COMBINATIONS.SEGMENT1
,GL_CODE_COMBINATIONS.SEGMENT2
,GL_CODE_COMBINATIONS.SEGMENT3
```

```
,GL_CODE_COMBINATIONS.SEGMENT4
,GL_CODE_COMBINATIONS.SEGMENT5
,GL_CODE_COMBINATIONS.SEGMENT6
,GL_CODE_COMBINATIONS.SEGMENT7
,GL_CODE_COMBINATIONS.SEGMENT8
,GL_CODE_COMBINATIONS.SEGMENT9
,GL_CODE_COMBINATIONS.SEGMENT10
,SUM( NVL(GL_BALANCES.BEGIN_BALANCE_DR,0) - NVL(GL_BALANCES.BEGIN_BALANCE_CR,0))"OPEN BAL"
,NVL(GL_BALANCES.PERIOD_NET_DR,0) "DEBIT"
,NVL(GL_BALANCES.PERIOD_NET_CR,0) "CREDIT"
,SUM( NVL(GL_BALANCES.PERIOD_NET_DR,0) - NVL(GL_BALANCES.PERIOD_NET_CR,0))"NET MOVEMENT"
,SUM(( NVL(GL_BALANCES.PERIOD_NET_DR,0) + NVL(GL_BALANCES.BEGIN_BALANCE_DR,0))) - SUM
       (NVL(GL_BALANCES.PERIOD_NET_CR,0)+NVL(GL_BALANCES.BEGIN_BALANCE_CR,0))"CLOSE BAL"
,GL_BALANCES.CURRENCY_CODE
,GL_BALANCES.TRANSLATED_FLAG
,GL_BALANCES.TEMPLATE_ID
,FND_FLEX_VALUES_VL.FLEX_VALUE
,FND_FLEX_VALUES_VL.DESCRIPTION
,FND_FLEX_VALUES_VL.FLEX_VALUE_SET_ID

FROM
GL_BALANCES,
GL_CODE_COMBINATIONS,
GL_SETS_OF_BOOKS,
FND_FLEX_VALUES_VL

WHERE GL_CODE_COMBINATIONS.CODE_COMBINATION_ID = GL_BALANCES.CODE_COMBINATION_ID
AND GL_BALANCES.ACTUAL_FLAG = 'A'
AND GL_BALANCES.CURRENCY_CODE = GL_SETS_OF_BOOKS.CURRENCY_CODE
AND GL_BALANCES.LEDGER_ID = GL_SETS_OF_BOOKS.SET_OF_BOOKS_ID
AND GL_BALANCES.TEMPLATE_ID IS NULL
AND GL_BALANCES.PERIOD_NAME = 'Jul-14'
AND FND_FLEX_VALUES_VL.FLEX_VALUE = GL_CODE_COMBINATIONS.SEGMENT4
AND FND_FLEX_VALUES_VL.FLEX_VALUE_SET_ID = '101432874'
AND GL_CODE_COMBINATIONS.SEGMENT2 IN (2050, 2100, 2200, 2300, 2400, 2500)

GROUP BY  GL_SETS_OF_BOOKS.NAME
,GL_BALANCES.ACTUAL_FLAG
```

```
,GL_BALANCES.PERIOD_NAME
,GL_BALANCES.PERIOD_NUM
,GL_BALANCES.PERIOD_YEAR
,GL_CODE_COMBINATIONS.CODE_COMBINATION_ID
,GL_CODE_COMBINATIONS.SEGMENT1
,GL_CODE_COMBINATIONS.SEGMENT2
,GL_CODE_COMBINATIONS.SEGMENT3
,GL_CODE_COMBINATIONS.SEGMENT4
,GL_CODE_COMBINATIONS.SEGMENT5
,GL_CODE_COMBINATIONS.SEGMENT6
,GL_CODE_COMBINATIONS.SEGMENT7
,GL_CODE_COMBINATIONS.SEGMENT8
,GL_CODE_COMBINATIONS.SEGMENT9
,GL_CODE_COMBINATIONS.SEGMENT10
,NVL(GL_BALANCES.PERIOD_NET_DR,0)
,NVL(GL_BALANCES.PERIOD_NET_CR,0)
,GL_BALANCES.CURRENCY_CODE
,GL_BALANCES.TRANSLATED_FLAG
,GL_BALANCES.TEMPLATE_ID
,FND_FLEX_VALUES_VL.FLEX_VALUE
,FND_FLEX_VALUES_VL.DESCRIPTION
,FND_FLEX_VALUES_VL.FLEX_VALUE_SET_ID
HAVING SUM(( NVL(GL_BALANCES.PERIOD_NET_DR,0) + NVL(GL_BALANCES.BEGIN_BALANCE_DR,0))) -
       SUM(NVL(GL_BALANCES.PERIOD_NET_CR,0)+NVL(GL_BALANCES.BEGIN_BALANCE_CR,0)) <> 0
```

SQL View

In many cases, creating a SQL View of data to provide information to OneStream is a more preferred option and typically simplifies the complexity of the query.

In the example below, the customer can combine several data tables required in the source system, and present the data in one View for OneStream to query:

```
SELECT
SEGMENT1 As Entity
SEGMENT2 As Establishment
SEGMENT3 As France_Account
SEGMENT4 As US_Account
SEGMENT5 As Cost_Center
```

```
SEGMENT6 As Family
SEGMENT7 As Product_Line
SEGMENT8 As Interco
SEGMENT9 As Future
PERIOD_YEAR As Year
PERIOD_MONTH As Month
CURRENCY_CODE As Currency_Code
CLOSE_NET_BALANCE As Net_Balance
SET_OF_BOOKS_ID As Set_Of_Books_ID
FROM APPS.XXSWM_ONESTREAM_GL_BALANCES
```

Stored Procedure

The example below is a SQL Stored Procedure used to pull Trial Balance Data from several different tables in a SQL Database.

In this example, the Entity, Year, and Period are passed to the Stored Procedure:

```
spGLCalcTrialBalance 'ASCC', '2013', 6
```

Apply Connection String to XFAppServerConfig.xml File

When the connection string is created, then the database connections can be centralized in the Server Configuration under the App Server Configuration File. Under Databases, click on (Collection) for Database Server Connections and the Database Server Connections will appear. The string will then be placed in the Connection String under Connection String Settings. The name of the connection string will be used as part of the source connector.

Defining External Data Connections

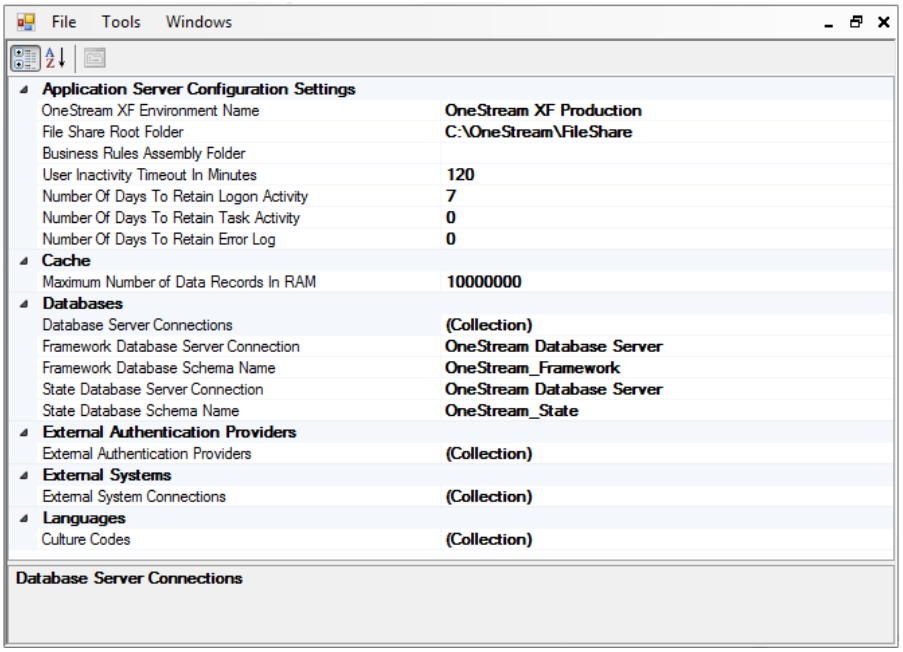
Application Server Configuration File
Creating Named External ODBC / OLEDB Connection

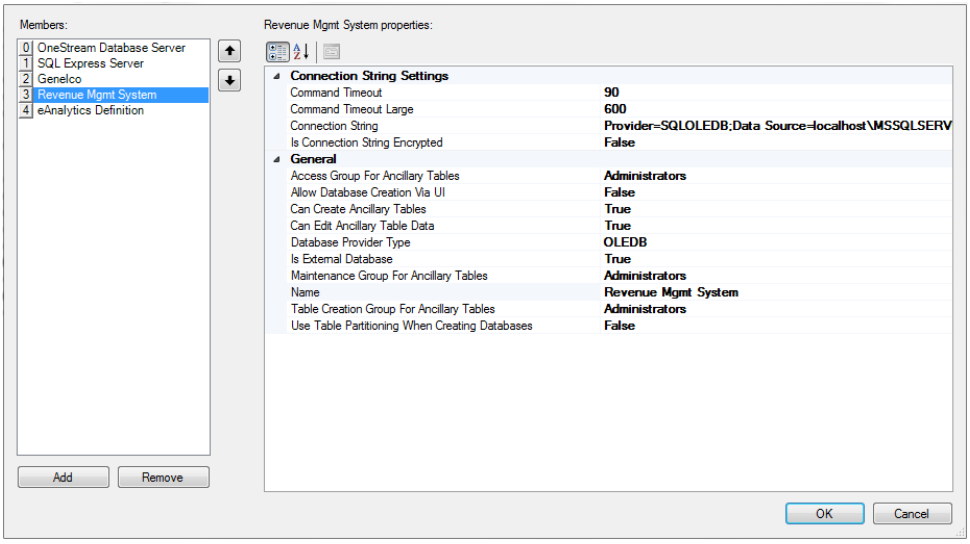
Step 1: Required ODBC/OLEDB Connection Software
Any client ODBC/OLEDB drivers must be installed on each application server for the OneStream application to make a connection to the external database. This way the administrator knows what type of database engine contains which Data Source.

Step 2: Creating the Connection String

The application server configuration file must be modified to add a named external database connection that can be used by the Connector Business Rule and custom reports.

Example of the Server Configuration Utility:





Step 3: Creating an External Database Test Query

The best way to prototype the queries needed to create a Connector Business Rule is to create a set of Dashboard Data Adapters to be used as a test bed.

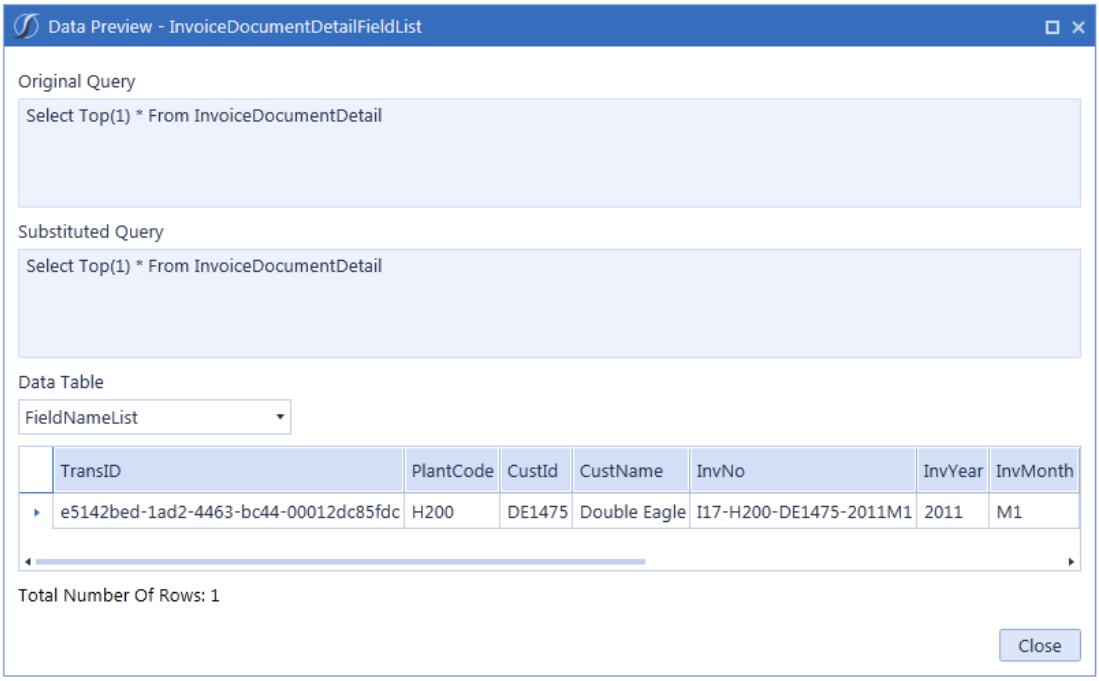
As a best practice, create a new Dashboard Maintenance Unit named EXS The Connector Name. The prefix EXS stands for External System and will provide administrators with an immediate understanding of the Maintenance Unit's contents. The three steps below explain how to create this.

1. Create a new Data Adapter for each type of query needed to proto type (GetFieldList, SelectData, Drill Back, etc.)

   Example of a Data Adapter being used to get all fields in the source table of the external database connection:

General (Data Adapter)	
Name	InvoiceDocumentDetailFieldList
Description	List Of Fields in Source Table
Maintenance Unit	External System (Revenue Mgmt Houston)
Data Source	
Command Type	SQL
Database Location	External
External Database Connection	Revenue Mgmt System
SQL Query	Select Top(1) * From InvoiceDocumentDetail
Results Table Name	FieldNameList

2. Select the  icon in the Dashboard administration toolbar to test the query.
3. Evaluate the results of the query. The Data Adapter test only returns a small subset of rows from the query, but it specifies the actual number of rows that will return during an actual query execution.



Building Data Connectors

ODBC / OLEDB Connectors

GetFieldList

Select Query against the external database. There will be a manual list of strings returned for each field.

GetData

The selected statement should match GetFieldList. Add criteria for Scenario and Time and map the OneStream Workflow Unit Scenario and Time values to corresponding values in the source system as a Where Clause criteria value.

GetDrillBackTypes

This shows the set of drill back options provided to the user.

GetDrillBack

This executes the selected drill back type for the current source data row.

Custom API Connectors

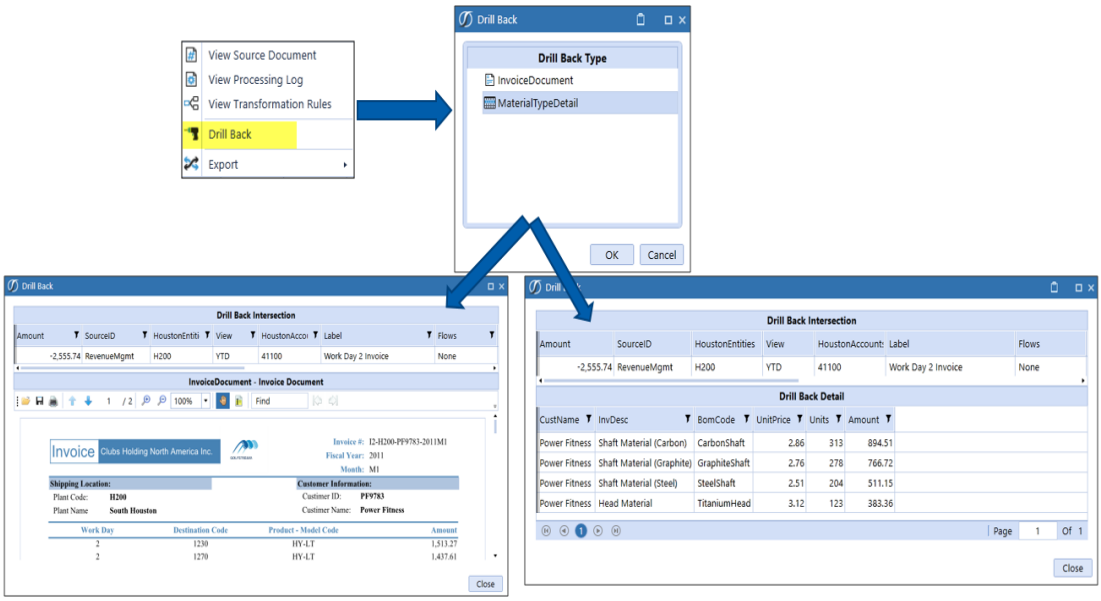
Uses OneStream's External Server Technology. IIS needs to be recycled on all application servers and followed by all web servers after adding the external named connection.

Drill Back

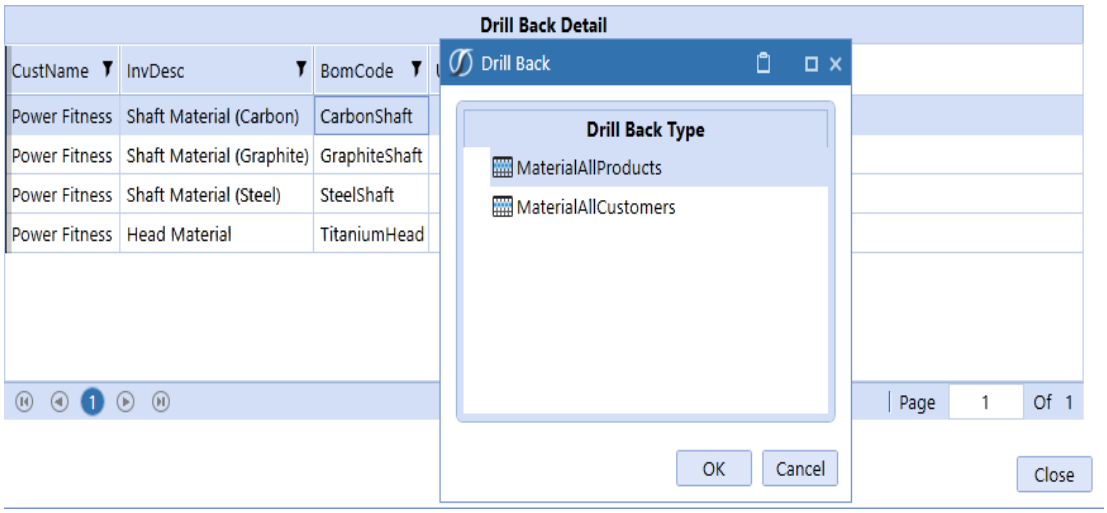
Using a SQL connector allows a user to drill back to a source system and show detailed records from a document, PDF, website. The Connector Data Source, configured by the author, provides a menu of data viewing options such as Year to Date, Month to Date, Invoice Documents or Material Type Detail. Utilizing this feature can reduce the amount of data imported into the Financial Model by allowing analysis to occur at the source system.

Viewing Data

After data is loaded into the Stage, a user can right-click on a data row and select Drill Back. This will bring up the pre-configured options from which the user can choose.



If more detail is needed, another level of Drill Back can be performed. This is configured in the Connector Business Rule and can drill back and around source systems. These nested drill paths can provide as much detail as an application requires.



Key API, Args, or BRAPI Examples

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

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

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

                    Case is = ConnectorActionTypes.GetDrillBackTypes
                        'Return the list of Drill Types (Options) to present to the end user
                        Return  Me.GetDrillBackTypeList(si, globals, api, args)

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

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

```
    I
'Create a Connection string to the External Database
Private Function GetConnectionString(ByVal si As Sessioninfo, ByVal globais As
BRGlobals, ByVal api As Transformer) As String Try
'Connection String Method
Dim connection As New Text.StringBuilder'
connection.Append("Provider=SQLOLEDB.1;")
connection.Append("Data Source=LocalHost\MSSQLSERVER2008;")
connection.Append("Initial Catalog=SampleData;")
connection.Append("Integrated Security=SSPI")
Return connection.ToString
'Named External Connection
Return "Revenue Mgmt System"
Catch ex As Exception
Throw ErrorHandler.LogWrite(si, New XFException(si, ex)) End Try End Function
```

```
'Create the field list SQL Statement
Private Function GetFieldListSQL(ByVal si As Sessioninfo, ByVal globais As BRGlobals, ByVal
api As Transformer) As String Try
```

```
'Create the SQL Statement
Dim sql As Mew Text.StringBuilder
sql.Append("SELECT Top(l)")
sql.Append("TransID, PlantCode, Custld, CustName, InvNo, InvYear, InvMonth, InvDesc,
GLAccount, WorkDay, ProdModel, BomCode, UnitPrice, Units, Amount, DestinationCode sql.Append
("FROM InvoiceDocumentDetail ")
Return sql.ToString
Catch ex As Exception
Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
End Try End Function
```

```
92      'Create the data load SQL Statement
93      Private Function GetSourceDataSQL(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As Transformer) As String
94          Try
95              'Create the SQL Statement
96              Dim statement As New Text.StringBuilder
97              Dim selectClause As New Text.StringBuilder
98              Dim fromClause as New Text.StringBuilder
99              Dim whereClause as New Text.StringBuilder
100             Dim orderByClause as New Text.StringBuilder
101
102             selectClause.Append("SELECT ")
103             selectClause.Append("TransID, PlantCode, CustId, CustName, InvNo, InvYear, InvMonth, InvDesc, GLAccount, WorkDay, ProdModel, BomCode, UnitPrice, Units, Amount, Destina
104
105             fromClause.Append("FROM InvoiceDocumentDetail ")
106
107             whereClause.Append("WHERE ")
108             'Get the YEAR from the current XF workflow unit TimeKey
109             whereClause.Append("(")
110             whereClause.Append("InvYear = " & TimeDimHelper.GetYearFromId(api.WorkflowUnitPk.TimeKey).ToString)
111             whereClause.Append(")")
112
113             'Get the MONTH from the current XF workflow unit TimeKey
114             whereClause.Append(" And ")
115             whereClause.Append("(")
116             whereClause.Append("InvMonth = 'M" & TimeDimHelper.GetSubComponentsFromId(api.WorkflowUnitPk.TimeKey).Month.ToString & "'")
117             whereClause.Append(") ")
118
119             'Select Houston Plant Codes
120             whereClause.Append(" And ")
121             whereClause.Append("(")
122             whereClause.Append("PlantCode IN('H200','H210')")
123             whereClause.Append(") ")
124
125             orderByClause.Append("ORDER BY ")
126             orderByClause.Append("PlantCode, CustId, WorkDay, ProdModel, DestinationCode")
127
128             'Create the full SQL Statement
129             statement.Append(selectClause.ToString)
130             statement.Append(fromClause.ToString)
131             statement.Append(whereClause.ToString)
132             statement.Append(orderByClause.ToString)
```

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

        drillTypes.Add(New DrillBackTypeInfo(ConnectorDrillBackDisplayTypes.FileShareFile, New NameAndDesc("Invoice Document","Invoice Document")))
        drillTypes.Add(New DrillBackTypeInfo(ConnectorDrillBackDisplayTypes.DataGrid, New NameAndDesc("Material Type Detail","Material Type Detail")))

        Return drillTypes

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

'Execute specific drill back type
private Function GetDrillBack(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As Transformer, ByVal args As ConnectorArgs, ByVal drillBackType As ConnectorDrillBackDisplayTypes, ByVal connectionStr
    Try
        Select case drillBackType
            case is = ConnectorDrillBackDisplayTypes.FileShareFile
                'Show FileShare File
                Dim drillBackInfo as new DrillBackResultInfo
                drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.FileShareFile
                drillBackInfo.DocumentPath = Me.GetDrillBackDocPath(si, globals, api, args)
                return drillBackInfo

            case is = ConnectorDrillBackDisplayTypes.DataGrid
                'Return Drill Back Detail
                Dim drillBackSQL As String = GetDrillBackSQL(si, globals, api, args)
                Dim drillBackInfo as new DrillBackResultInfo
                drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid
                drillBackInfo.DataTable = api.Parser.GetXFDataTableForSQLQuery(si, DbProviderType.OLEDB, connectionString, true, drillBackSQL, false, args.PageSize, args.PageNumber)
                return drillBackInfo

            case else
                return Nothing
        End Select

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

```
'Create the drill back Document Path
Private Function GetDrillBackDccPath(Byval si As Sessioninfo, ByVal globais As BRGlobals,
ByVal api As Transformer, Byval args As ConnectorArgs) As String Try
'Get the values for the source row that we are drilling back to
Dim sourcevalues as Dictionary(of string, object) =
api.Parser.GetFieldValuesForsourceDataRow(si, args.RcwlD)
If (Mot sourcevalues is Nothing) And (sourcevalues.Count > 0) then
Return "Applications/GolfStream_v24/DataManagement/RevenueMgmtlnvoices/" & sourcevalues.ltem
(StageCcnstants.MasterDiraensionNames.Attributel).Tostring & ".pdf"
Else
Return String.Empty End If
Catch ex As Exception
Throw ErrorHandler.Logwrite(si, New XFException(si, ex))
End Try
End Function
```

```
    'Create the drill back SQL Statement
    Private Function GetDrillBackSQL(ByVal si As SessionInfo, ByVal globals As BRGlobals, ByVal api As Transformer, ByVal args As ConnectorArgs) As String
        Try
            'Get the values for the source row that we are drilling back to
            Dim sourceValues as Dictionary(Of string, Object) = api.Parser.GetFieldValuesForSourceDataRow(si, args.RowID)
            If (Not sourceValues Is Nothing) And (sourceValues.Count > 0) then

                Dim statement As New Text.StringBuilder
                Dim selectClause As New Text.StringBuilder
                Dim fromClause as New Text.StringBuilder
                Dim whereClause as New Text.StringBuilder
                Dim orderByClause as New Text.StringBuilder

                'Create the SQL Statement
                selectClause.Append("SELECT ")
                selectClause.Append("CustName, InvDesc, BomCode, UnitPrice, Units, Amount ")

                fromClause.Append("FROM InvoiceMaterialDetail ")

                whereClause.Append("WHERE ")
                'Get the YEAR from the source recrod
                whereClause.Append("(")
                whereClause.Append("InvYear = " & TimeDimHelper.GetYearFromId(sourceValues.Item(StageTableFields.StageSourceData.DimWorkflowTimeKey).ToString))
                whereClause.Append(")")

                'Get the MONTH from the source record
                whereClause.Append(" And ")
                whereClause.Append("(")
                whereClause.Append("InvMonth = 'M" & TimeDimHelper.GetSubComponentsFromId(sourceValues.Item(StageTableFields.StageSourceData.DimWorkflowTimeKey)).Month.ToString & "'")
                whereClause.Append(")")

                whereClause.Append(" And ")
                whereClause.Append("(")
                whereClause.Append("PlantCode = '" & sourceValues.Item(StageConstants.MasterDimensionNames.Entity).ToString & "'")
                whereClause.Append(")")

                whereClause.Append(" And ")
                whereClause.Append("(")
                whereClause.Append("InvNo = '" & sourceValues.Item(StageConstants.MasterDimensionNames.Attribute1).ToString & "'")
                whereClause.Append(")")

                whereClause.Append(" And ")
                whereClause.Append("(")
                whereClause.Append("ProdModel = '" & sourceValues.Item(StageConstants.MasterDimensionNames.UD2).ToString & "'")
                whereClause.Append(")")
```

```
                    whereClause.Append(" And ")
                    whereClause.Append("(")
                    whereClause.Append("DestinationCode = '" & sourceValues.Item(StageConstants.MasterDimensionNames.UD3).ToString & "'")
                    whereClause.Append(")")

                    whereClause.Append(" And ")
                    whereClause.Append("(")
                    whereClause.Append("CustID = '" & sourceValues.Item(StageConstants.MasterDimensionNames.UD4).ToString & "'")
                    whereClause.Append(")")

                    orderByClause.Append("ORDER BY ")
                    orderByClause.Append("BomCode")

                    'Create the full SQL Statement
                    statement.Append(selectClause.ToString)
                    statement.Append(fromClause.ToString)
                    If args.ClientFilterRequest.length > 0 then
                        statement.Append(whereClause.ToString)
                        statement.Append(" And ")
                        statement.Append(args.ClientFilterRequest)
                    Else
                        statement.Append(whereClause.ToString)
                    End If
                    If args.ClientSortRequest.Length > 0 then
                        statement.Append(args.ClientSortRequest)
                    Else
                        statement.Append(orderByClause.ToString)
                    End if
                    'ErrorHandler.LogMessage(si, statement.ToString)
                    Return statement.ToString
                Else
                    Return String.Empty
                End If
            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function
    End Class
End Namespace
```

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

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

                     Case Is = ConnectorActionTypes.GetData
                         'Process Data
                         Dim sourceDataSQL As String = GetSourceDataSQL(si, globals, api)
```

```
                        api.Parser.ProcessSQLQuery(si, DbProviderType.OLEDB,
connectionString, true,
                sourceDataSQL, false, api.ProcessInfo)
                        Return Nothing

                    Case is = ConnectorActionTypes.GetDrillBackTypes
                        'Return the list of Drill Types (Options) to present to the end user
                        Return  Me.GetDrillBackTypeList(si, globals, api, args)

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

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

        'Create a Connection string to the External Database
        Private Function GetConnectionString(ByVal si As SessionInfo, ByVal globals As
BRGlobals,
        ByVal api As Transformer) As String
            Try

                'Connection String Method
                '------------------------------------------------------------
'                Dim connection As New Text.StringBuilder'
'                connection.Append("Provider=SQLOLEDB.1;")
'                connection.Append("Data Source=LocalHost\MSSQLSERVER2008;")
'                connection.Append("Initial Catalog=SampleData;")
'                connection.Append("Integrated Security=SSPI")
'                Return connection.ToString

                'Named External Connection
                '------------------------------------------------------------
                Return "Revenue Mgmt System"

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

```
    'Create the field list SQL Statement
        Private Function GetFieldListSQL(ByVal si As SessionInfo, ByVal globals As
BRGlobals,
        ByVal api As Transformer) As String
            Try
```

```
                'Create the SQL Statement
                Dim sql As New Text.StringBuilder

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

                Return sql.ToString

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

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

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

                fromClause.Append("FROM InvoiceDocumentDetail ")

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

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

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

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

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

                Return statement.ToString

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

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

                drillTypes.Add(New DrillBackTypeInfo
(ConnectorDrillBackDisplayTypes.FileShareFile,
 New NameAndDesc("Invoice Document","Invoice Document")))
                drillTypes.Add(New DrillBackTypeInfo
(ConnectorDrillBackDisplayTypes.DataGrid,
 New  NameAndDesc("Material Type Detail","Material Type Detail")))

                Return drillTypes

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

```
    'Execute specific drill back type
        private Function GetDrillBack(ByVal si As SessionInfo, ByVal globals As BRGlobals,
 ByVal api As Transformer,
 ByVal args As ConnectorArgs, ByVal drillBackType As ConnectorDrillBackDisplayTypes,
ByVal connectionString as String) As DrillBackResultInfo
            Try
                Select case drillBackType
                    case is = ConnectorDrillBackDisplayTypes.FileShareFile
                        'Show FileShare File
                        Dim drillBackInfo as new DrillBackResultInfo
                        drillBackInfo.DisplayType =
ConnectorDrillBackDisplayTypes.FileShareFile
                        drillBackInfo.DocumentPath = Me.GetDrillBackDocPath(si, globals,
api, args)
                        return drillBackInfo

                    case is = ConnectorDrillBackDisplayTypes.DataGrid

                        'Return Drill Back Detail
                        Dim drillBackSQL As String = GetDrillBackSQL(si, globals, api, args)
                        Dim drillBackInfo as new DrillBackResultInfo
                        drillBackInfo.DisplayType = ConnectorDrillBackDisplayTypes.DataGrid

                        drillBackInfo.DataTable = api.Parser.GetXFDataTableForSQLQuery(si,
                        DbProviderType.OLEDB, connectionString, true, drillBackSQL,
 false, args.PageSize, args.PageNumber)
                        return drillBackInfo

                        case else
                        return Nothing
                End Select

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

```
    'Create the drill back Document Path
        Private Function GetDrillBackDocPath(ByVal si As SessionInfo,
ByVal globals As BRGlobals,ByVal api As Transformer, ByVal args As ConnectorArgs) As String
            Try
                'Get the values for the source row that we are drilling back to
                Dim sourceValues as Dictionary(Of string, Object) =
api.Parser.GetFieldValuesForSourceDataRow(si, args.RowID)
                If (Not sourceValues Is Nothing) And (sourceValues.Count > 0) then

                    Return "Applications/GolfStream_v24/DataManagement/RevenueMgmtInvoices/"
 & sourceValues.Item(StageConstants.MasterDimensionNames.Attribute1).ToString & ".pdf"
                Else
                    Return String.Empty
```

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

```
    'Create the drill back SQL Statement
        Private Function GetDrillBackSQL(ByVal si As SessionInfo, ByVal globals As
BRGlobals,
 ByVal api As Transformer, ByVal args As ConnectorArgs) As String
            Try
                'Get the values for the source row that we are drilling back to
                Dim sourceValues as Dictionary(Of string, Object) =
api.Parser.GetFieldValuesForSourceDataRow(si, args.RowID)
                If (Not sourceValues Is Nothing) And (sourceValues.Count > 0) then


                    Dim statement As New Text.StringBuilder
                    Dim selectClause As New Text.StringBuilder
                    Dim fromClause as New Text.StringBuilder
                    Dim whereClause as New Text.StringBuilder
                    Dim orderByClause as New Text.StringBuilder

                    'Create the SQL Statement
                    selectClause.Append("SELECT ")
                    selectClause.Append("CustName, InvDesc, BomCode, UnitPrice, Units,Amount
")

                    fromClause.Append("FROM InvoiceMaterialDetail ")

                    whereClause.Append("WHERE ")
                    'Get the YEAR from the source record
                    whereClause.Append("(")
                    whereClause.Append("InvYear = " &  TimeDimHelper.GetYearFromId
(sourceValues.Item
(StageTableFields.StageSourceData.DimWorkflowTimeKey).ToString))
                    whereClause.Append(")")

                    'Get the MONTH from the source record
                    whereClause.Append(" And ")
                    whereClause.Append("(")
                    whereClause.Append("InvMonth = 'M" &
TimeDimHelper.GetSubComponentsFromId(sourceValues.Item
(StageTableFields.StageSourceData.DimWorkflowTimeKey))                    .Month.ToString &
"'")
                    whereClause.Append(")")

                    whereClause.Append(" And ")
                    whereClause.Append("(")
                    whereClause.Append("PlantCode = '" & sourceValues.Item
            (StageConstants.MasterDimensionNames.Entity)                    .ToString & "'")
```

```
                    whereClause.Append(")")

                    whereClause.Append(" And ")
                    whereClause.Append("(")
                    whereClause.Append("InvNo = '" & sourceValues.Item
            (StageConstants.MasterDimensionNames.Attribute1)                    .ToString &
"'")
                    whereClause.Append(")")

                    whereClause.Append(" And ")
                    whereClause.Append("(")
                    whereClause.Append("ProdModel = '" & sourceValues.Item
            (StageConstants.MasterDimensionNames.UD2).ToString& "'")
                    whereClause.Append(")")

                    whereClause.Append(" And ")
                    whereClause.Append("(")
                    whereClause.Append("DestinationCode = '" & sourceValues.Item
            (StageConstants.MasterDimensionNames.UD3).                    ToString & "'")
                    whereClause.Append(")")

                    whereClause.Append(" And ")
                    whereClause.Append("(")
                    whereClause.Append("CustID = '" &sourceValues.Item
(StageConstants.MasterDimensionNames.UD4)                    .ToString & "'")
                    whereClause.Append(")")

                    orderByClause.Append("ORDER BY ")
                    orderByClause.Append("BomCode")

                    'Create the full SQL Statement
                    statement.Append(selectClause.ToString)
                    statement.Append(fromClause.ToString)
                    If args.ClientFilterRequest.length > 0 then
                        statement.Append(whereClause.ToString)
                        statement.Append(" And ")
                        statement.Append(args.ClientFilterRequest)
                    Else
                        statement.Append(whereClause.ToString)
                    End If
                    If args.ClientSortRequest.Length > 0 then
                        statement.Append(args.ClientSortRequest)
                    Else
                        statement.Append(orderByClause.ToString)
                    End if
                    'ErrorHandler.LogMessage(si, statement.ToString)
                    Return statement.ToString
                Else
                    Return String.Empty
                End If
            Catch ex As Exception
                Throw ErrorHandler.LogWrite(si, New XFException(si, ex))
            End Try
        End Function
    End Class
End Namespace
```

Data Management Export Sequences Data Source

Instead of a file or connector, use a data management export sequence in a workflow import. Use a Sequence data source in an Import workflow to move and export data. For example, these can be used in an Import workflow to:

- Copy data from one OneStream cube or scenario to another cube or scenario, such as when cubes or scenarios contain different dimensionality.
- Export OneStream data to an external source system and apply transformation rules through the workflow.

This can be comprised of multiple data management steps. See "Data Management" in Application Tools Guide for more information.

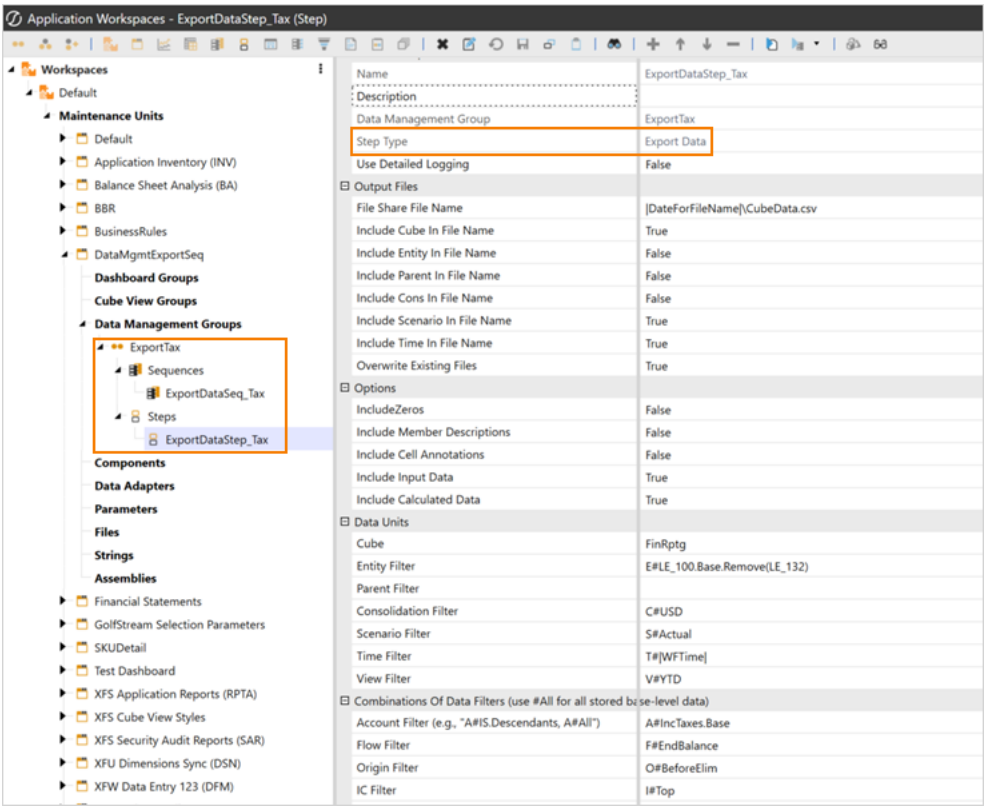
Data Management Settings

Data Export Sequence Name
The name of the data management export sequence being used for this data source.

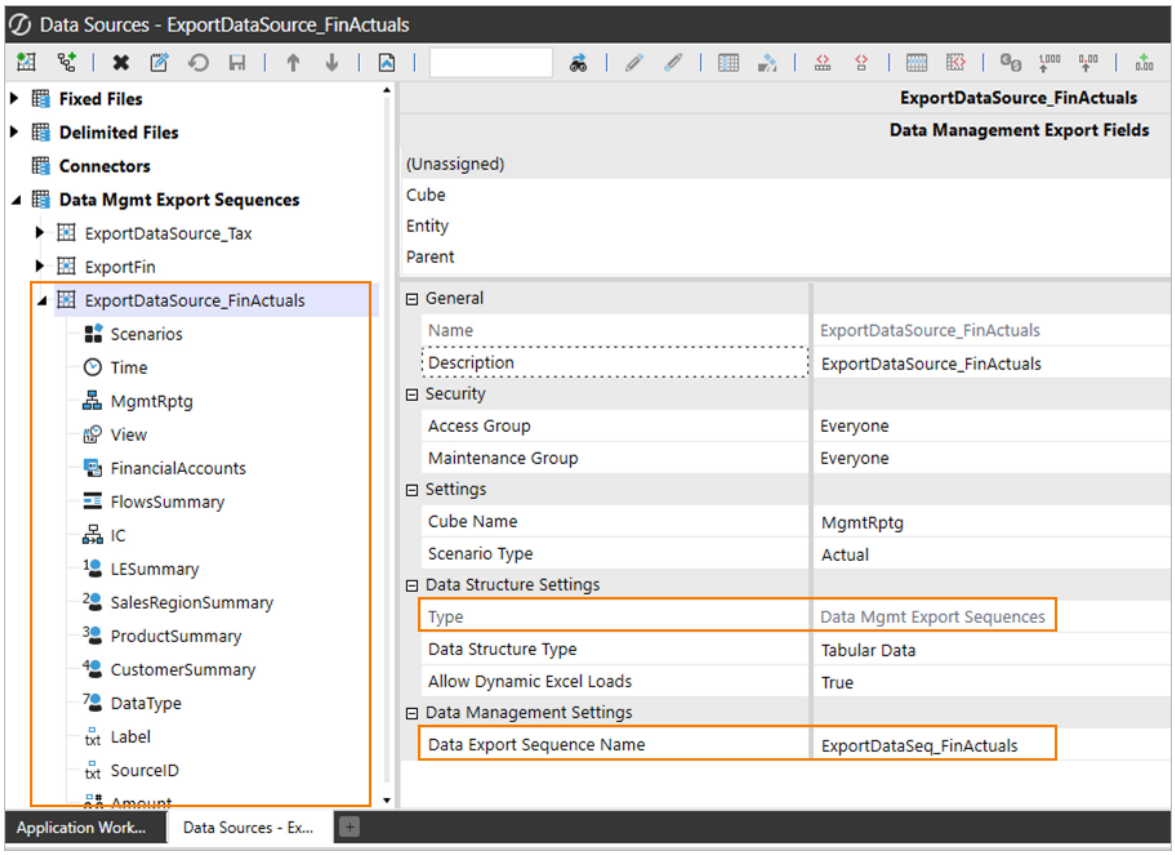
Use Case: Create a Data Management Export Sequence to Copy Data Within OneStream

This use case reviews how to copy OneStream data to another cube or scenario with differing dimensionality through a Data Management Export Sequence data source in an Import workflow. In this example, users will copy actuals data from a financial close and a consolidations cube to a planning cube with differing dimensionality on a monthly basis. Below are the configuration steps and the result.

1. Configure a data management step with the Export Data type to specify the source data to move. Assign the step to a data management sequence. See Data Management in Application Tools for more information.

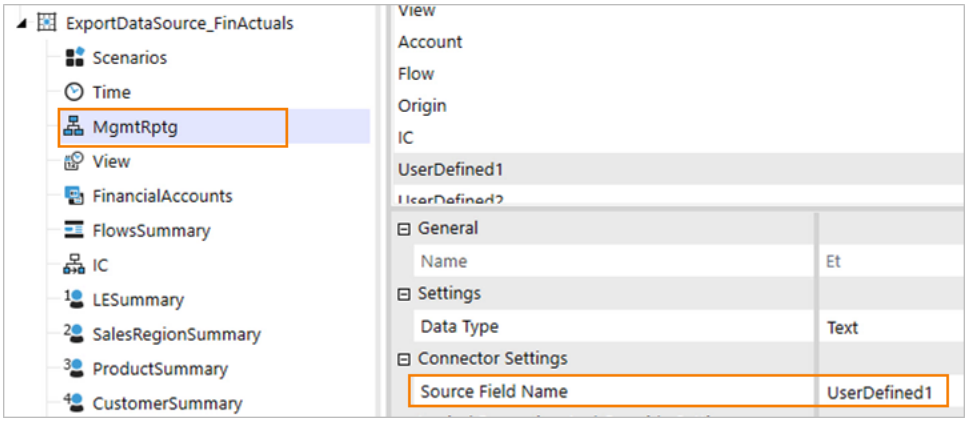


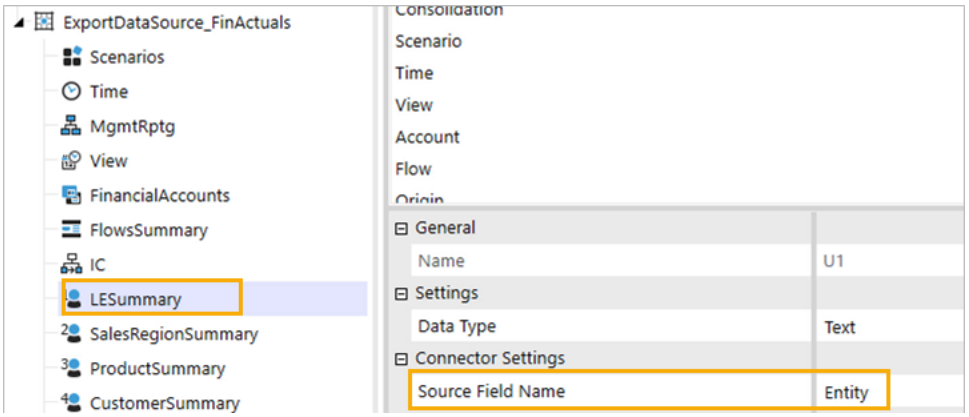
2. Create a data source with the Data Mgmt Export Sequences type. Set the Data Export Sequence Name to the name of the data management sequence.



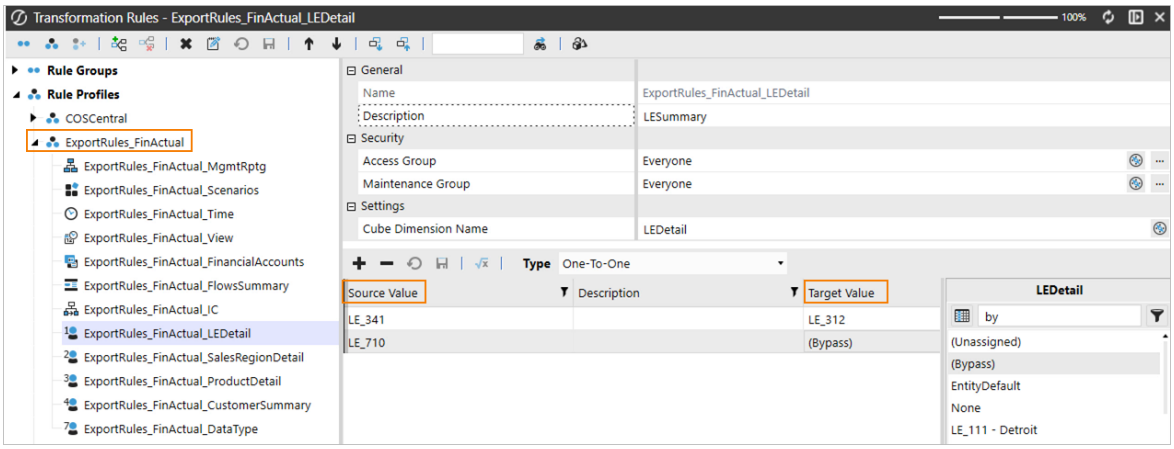
Configure the source dimensions for the data source. See Source Dimension Properties in Data Sources for more information.

In this example, actuals data will be copied from the financial close and consolidation cube (FinRptg) to the planning cube (MgmtRptg), which contains different dimensionality. The FinRptg's legal entity data will be copied to the MgmtRptg planning cube's UD1 dimension, and the FinRptg's UD1 cost center data will be copied to the MgmtRptg planning cube's Entity dimension as defined in the Source Field Name properties.

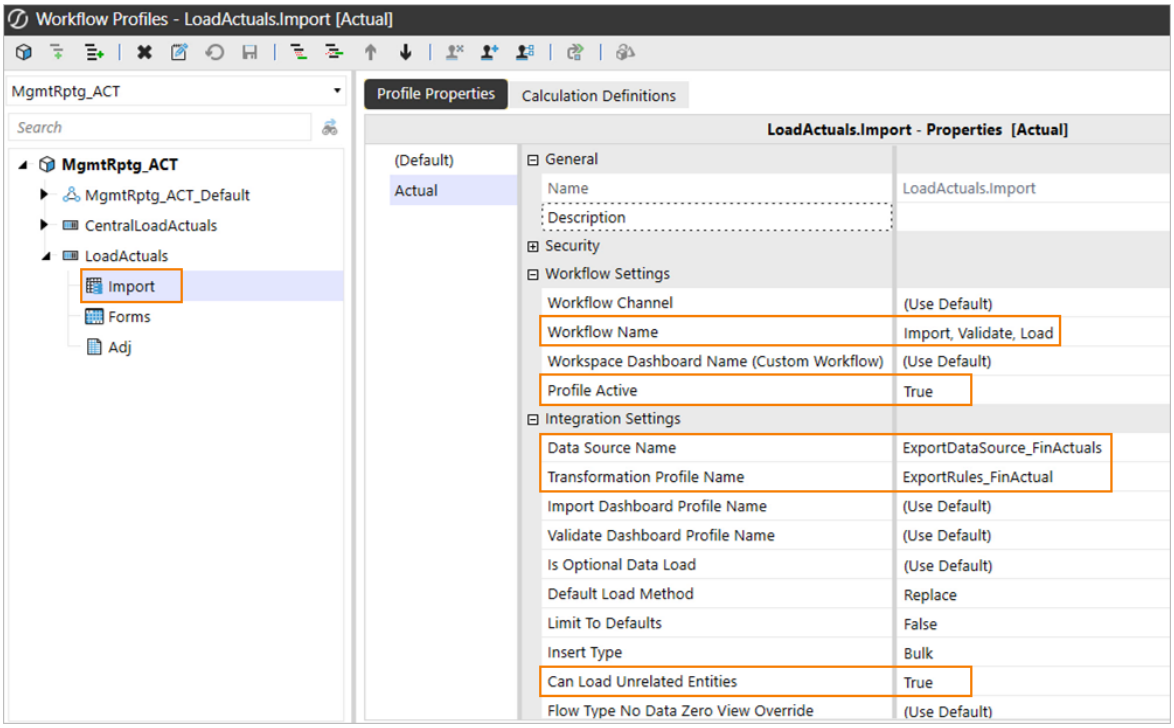




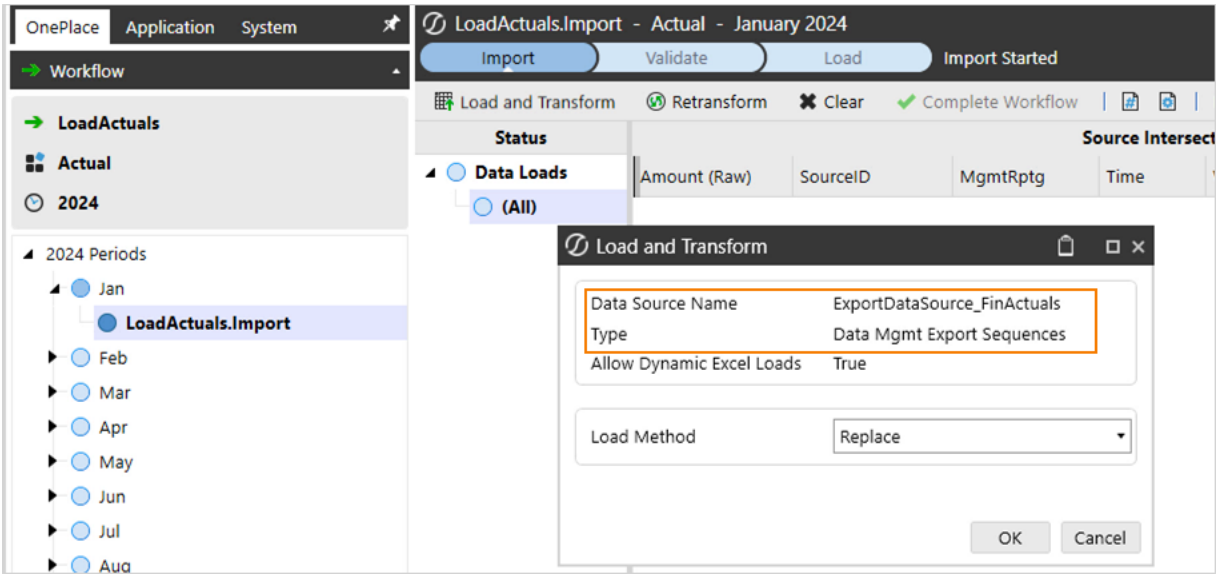
3. Create a transformation profile and groups to map the source data to the target. See “Transformation Rules” for more information.



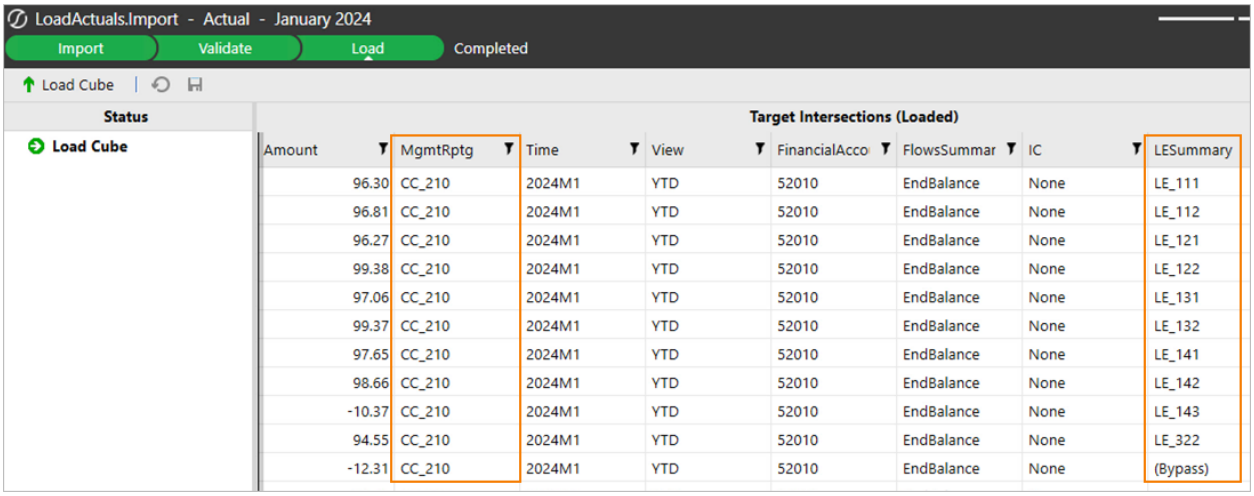
4. Create an Import Workflow Profile and assign the Data Management Export Sequence data source and the Transformation Profile Name. Set the Workflow Name to Import, Validate, and Load. The Import task will copy and transform the source data to the Stage tables. The Validate and Load tasks will map and load the Stage data to the target. Also, ensure that Profile Active is set to True, and Can Load Unrelated Entities is set to True if data is loaded to multiple entities.



In OnePlace, when a user clicks Load and Transform in the Workflow Import task, the Data Source Name and Type show the assigned Data Management Export Sequence data source.



In this example, when a user completes the Import, Validate, and Load tasks in a workflow, the data from FinRptg is transformed and loaded into MgmtRptg cube. For the planning MgmtRptg cube, the FinRptg legal entities were loaded to the UD1 dimension, and the FinRptg UD1 cost centers were loaded to the Entity dimension.



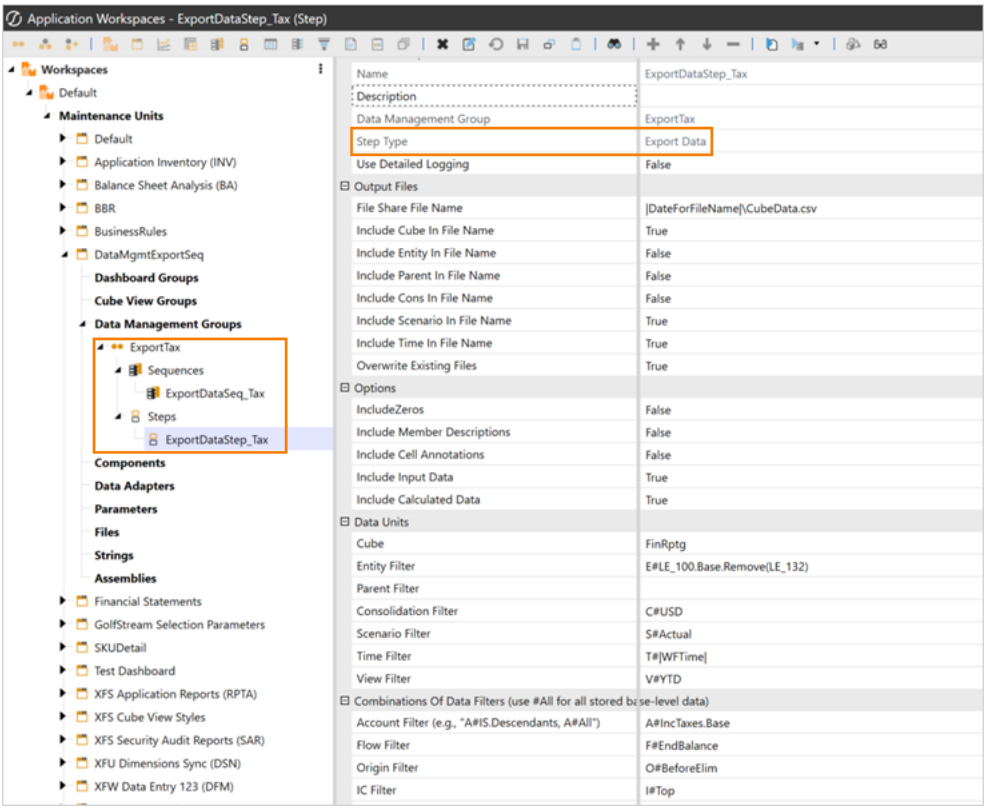
Amount	MgmtRptg	Time	View	FinancialAcco	FlowsSummar	IC	LESummary
96.30	CC_210	2024M1	YTD	52010	EndBalance	None	LE_111
96.81	CC_210	2024M1	YTD	52010	EndBalance	None	LE_112
96.27	CC_210	2024M1	YTD	52010	EndBalance	None	LE_121
99.38	CC_210	2024M1	YTD	52010	EndBalance	None	LE_122
97.06	CC_210	2024M1	YTD	52010	EndBalance	None	LE_131
99.37	CC_210	2024M1	YTD	52010	EndBalance	None	LE_132
97.65	CC_210	2024M1	YTD	52010	EndBalance	None	LE_141
98.66	CC_210	2024M1	YTD	52010	EndBalance	None	LE_142
-10.37	CC_210	2024M1	YTD	52010	EndBalance	None	LE_143
94.55	CC_210	2024M1	YTD	52010	EndBalance	None	LE_322
-12.31	CC_210	2024M1	YTD	52010	EndBalance	None	(Bypass)

Use Case: Create a Data Management Export Sequence to Export Data From OneStream

This use case reviews how to export OneStream data to an external source system with differing data structures through a Data Management Export Sequence data source in an Import Workflow. Below are the configuration steps and the result. In this example, users will export and transform data every month from OneStream to their external tax system, which has its own entity structure and chart of accounts.

1. Configure the Data Management Step and Sequence:

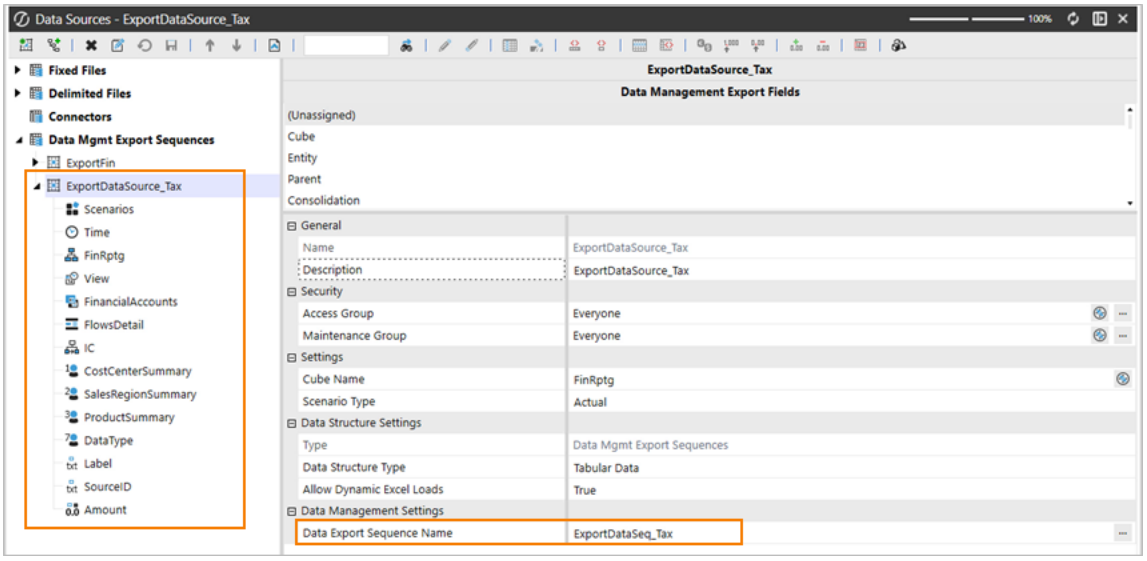
   Configure a data management step with the Export Data type to specify the source data to export. Assign the step to a data management sequence. See Data Management in Application Tools for more information.



2. Configure the Data Source:

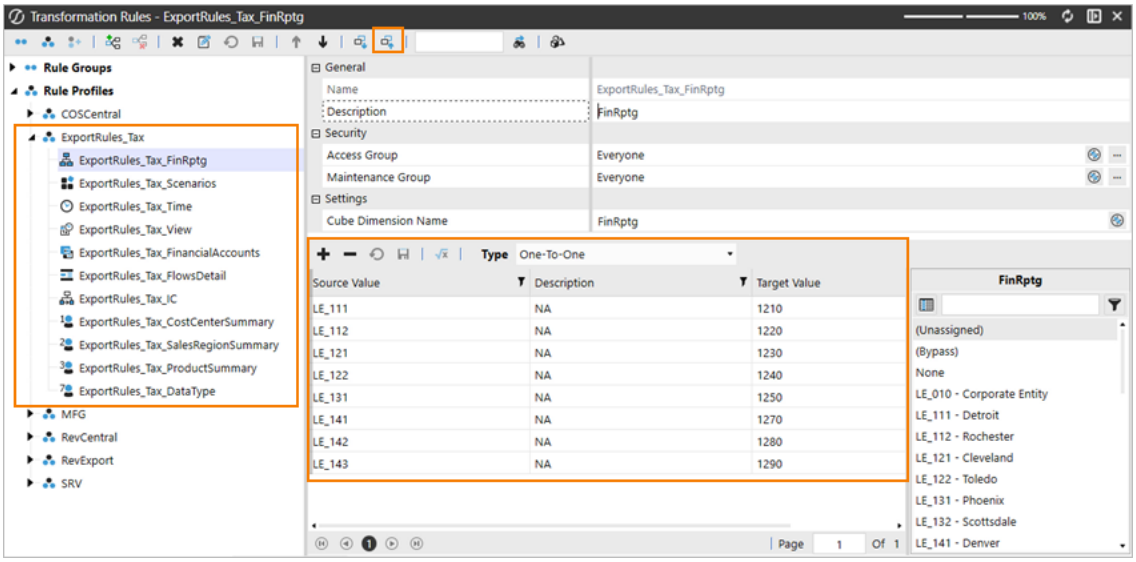
   Create a data source with the Data Mgmt Export Sequences type. Set the Data Export Sequence Name to the name of the data management sequence.

Configure the source dimensions for the data source. See "Source Dimension Properties" in Data Sources for more information.



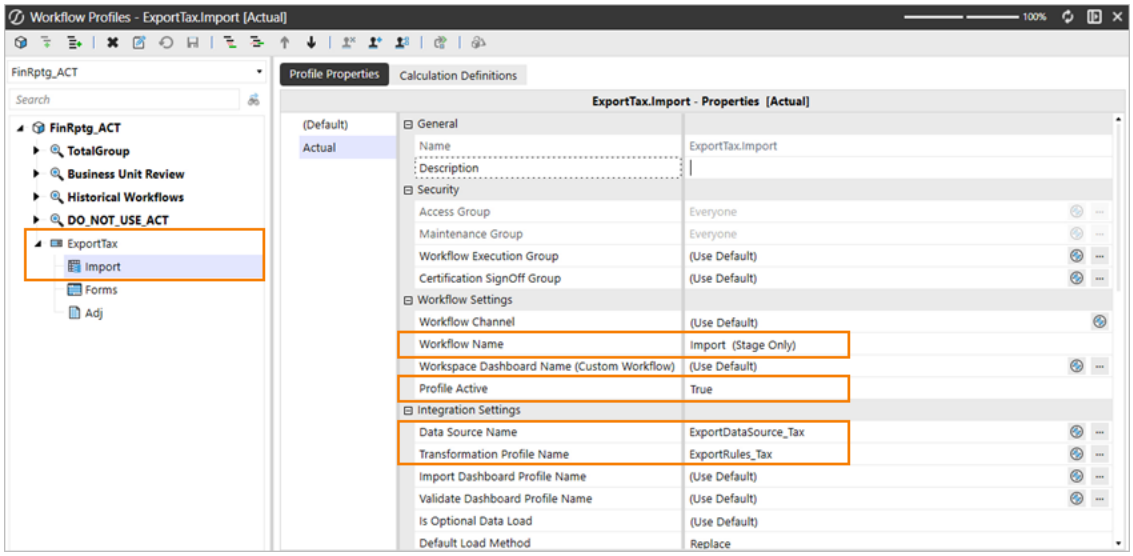
3. Configure the Transformation Rules:

   Create a transformation profile and groups to map the source data to the target. For dimensions that need to be transformed, specify the Source Value and Target Value. Import external target values using a TRX file. See Transformation Rules for more information.



4. Configure an Import Workflow Profile:

   Create an Import Workflow Profile and assign the Data Management Export Sequence data source and Transformation Profile Name. Ensure the Workflow Name contains an Import (Stage Only) task, which will import the transformed data to the Stage tables. Also, ensure that Profile Active is set to True.



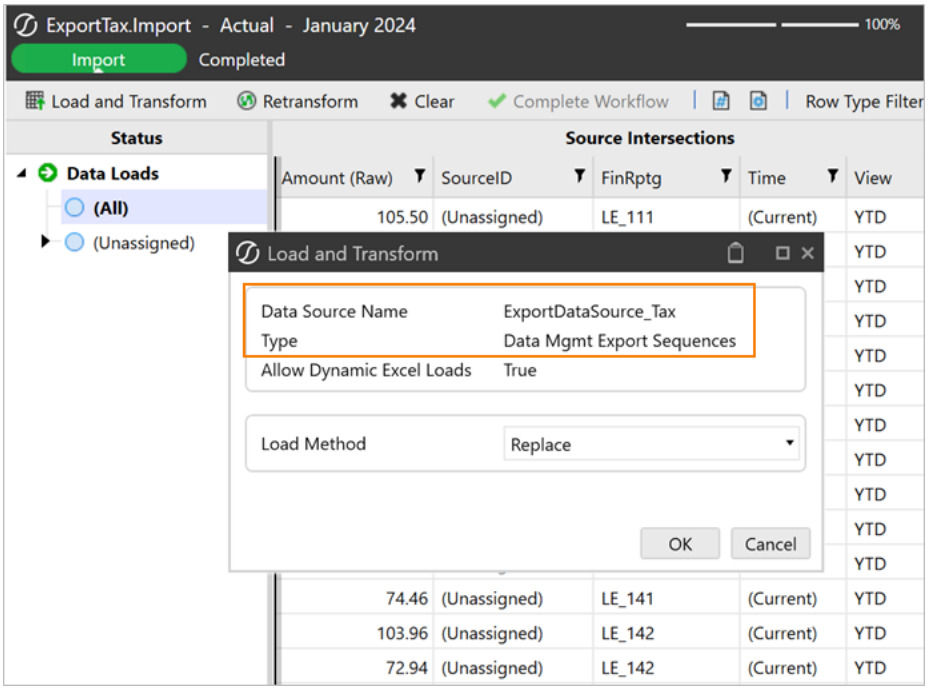
5. Configure the Method to Retrieve Source Data

   When a user completes the Import workflow task, the transformed data will be in the Stage tables. In this example, data is being exported from OneStream to an external source, so the data will need to be retrieved from the Stage tables. There are multiple methods to retrieve the transformed data, such as using a:

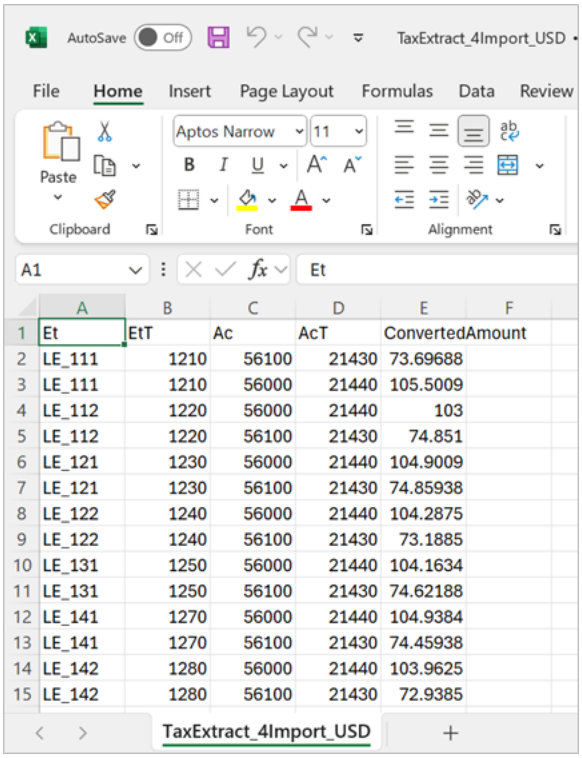
   • Workflow Event Handler: This business rule can run code when the Workflow completes a certain step. For example, the Workflow Event Handler can automatically generate a file with the exported and transformed data when the user completes the Workflow Import task.

- Transformation Event Handler: This business rule can run code at various points of the Workflow from Import through Load. For example, the Transformation Event Handler can automatically generate a file with the exported and transformed data when the user completes the Import task.
- Extender Rule: Data can be retrieved from Stage ad hoc with an Extender business rule. The rule can be run as needed or scheduled to run on a recurring basis to export the data to a table or file.
- Data Adapter with OneStream Rest API: Stage data can be retrieved using an Application Workspace Data Adapter with the Command Type of Method and Method Type of StageSummaryTargetData. A OneStream Rest API can call the data adapter and give the data to the caller in JSON format.

In OnePlace, when a user clicks Load and Transform in the Workflow Import task, the Data Source Name and Type show the assigned Data Management Export Sequence data source. When the Import task is completed, the transformed data is populated in the Stage tables, and it can be retrieved for the external system.



For example, a Workflow Event Handler business rule can automatically retrieve the transformed data from Stage and export it to a file when the Import task is completed. In this example, the exported file contains the OneStram source entity (ET), transformed target entity (EtT), OneStream source account (Ac), transformed target account (AcT), and amount.



	A	B	C	D	E	F
1	Et	EtT	Ac	AcT	ConvertedAmount	
2	LE_111	1210	56100	21430	73.69688	
3	LE_111	1210	56000	21440	105.5009	
4	LE_112	1220	56000	21440	103	
5	LE_112	1220	56100	21430	74.851	
6	LE_121	1230	56000	21440	104.9009	
7	LE_121	1230	56100	21430	74.85938	
8	LE_122	1240	56000	21440	104.2875	
9	LE_122	1240	56100	21430	73.1885	
10	LE_131	1250	56000	21440	104.1634	
11	LE_131	1250	56100	21430	74.62188	
12	LE_141	1270	56000	21440	104.9384	
13	LE_141	1270	56100	21430	74.45938	
14	LE_142	1280	56000	21440	103.9625	
15	LE_142	1280	56100	21430	72.9385	

Source Dimensions

Each data source has assigned Source dimensions. To add Source dimensions to a data source, click **Add** in the toolbar and select a dimension. These dimensions correspond with the dimensions in the cube specified in data source properties.

Source Dimension Properties

You can specify the following properties for a Source dimension.

Settings

- **Data Type**: Not all data types will be available for every Source dimension.
- **DataKey Text**: This will read the value from the file as defined in the position settings.
- **Stored DataKey Text**: This will override the position settings and force the Time value to be a constant value for every line.
- **Global DataKey Time**: This will use the Time value from the current Global POV time being processed.
- **Current DataKey Time**: This will use the Time Value from the current Workflow POV.
- **Current DataKey Scenario**: This will use the Scenario value from the current Workflow POV.
- **Matrix DataKey Text**: This is used as a Data Type when the data source is setup as a Matrix Load with multiple periods and used to identify the defined Start Position and Length of a single period.
- **Text**: This will read the value from the file as defined in the position settings.
- **Stored Text**: This will override the position settings and force the value to be a constant value for every line.
- **Matrix Text**: This will have multiple amounts on a given line using rows and columns. This will determine the intersection that corresponds with each amount when more than one column contains the same dimension.
- **Label**: This will read the value from the file as defined in the position settings.
- **Stored Label**: This will override the position settings and force the value to be a constant for every line.

- **Numeric**: This defines the numeric amount field for the data source. This field will be read and stored as a number, not as text or string.

Position Settings

Position settings are the definition of where the Source dimension will be found in the source file. For both fixed width and delimited files, there are tools in the toolbar and an attached file that assist in populating these values. Highlight the specific area to assign it to a dimension (for a delimited file, it only needs to be a portion of the column). The highlight will appear in red. When the defined area is selected, click  . This will commit the selection to the dimension and the corresponding values will be populated in either the Start Position and Length fields for Fixed Width or the Column Number field for Delimited. To clear this selection without committing it to the dimension, click  .

- **Start Position (Fixed Files Only)**: This is the numerical representation of the starting point for a line item.
- **Length (Fixed Files Only)**: This defines how many characters will be taken from the start position.

  A Fixed data source with a start position of 20 and a length of five will start with the 20th character and include the next five characters.
- **Column (Delimited Files Only)**: A delimited data source will use a column number. If the column number is four, that dimension will be represented by the 4th column in the source file.

Connector Settings

Source field names will be provided by the Connector business rule assigned to the connector data source. These field names will either be explicitly listed out in the business rule, or dynamically returned from a SQL query. Source field names for Data Management Sequences are provided and always contain the same list.

Logical Expression and Override Settings

- **Logical Operator**: Allows the ability to assign a .NET scripting functionality to a dimension of a data source.
- **Complex Expression**: This selection is used when .NET scripting is needed for the dimension, but not needed elsewhere. The script used in the complex expression will only be available within that dimension.
- **Business Rule**: This selection is used when .NET scripting is needed for the dimension and the script is available in the Business Rule Library.
- **Logical Expression**: This is the name of the business rule assigned to the dimension when Business Rule is selected for Logical Operator.
- **Static Value**: This is an override setting which allows a hard-coded value to be assigned to a dimension rather than being read from a file or data source.

Date Fill Settings

- **Date Format**: Users can specify a list of datetime formats to apply, as accepted by DateTime.TryParseExact. See Microsoft DateTime.TryParseExact Method for more information. One of these formats must match exactly, in order for the value to be considered valid. See Microsoft Custom date and time format strings for more information.

If no format is specified, the system falls back to Invariant Culture short date formats. For example, M/d/yyyy, M/d/yy, MM/dd/yy, MM/dd/yyyy, yy/MM/dd, yyyy-MM-dd, or dd-MMM-yy.

Valid values must be between 1 January 1900 and 31 December 2200.

Text Fill Settings

- **Leading Fill Value**: Characters entered in this field will precede whatever value is brought in from the file upon import.

  Lead Fill Mask = xxx, data value = 00, results value = x00

- **Trailing Fill Value**: Characters entered in this field will be placed after any value brought in from the file upon import.

  Trail Fill Mask = xxx, Data Value = 00, Results Value = 00x

Substitution Settings

- **Substitution Old Value (Find)**: If the value entered in this field is encountered in the dimension, it will be replaced with what is entered in the Substitution New Value.

  Single Value = value1

  Multiple Values = value1^value2

- **Substitution New Value (Replace)**: This will replace the value in Substitution Old Value if it occurs in the dimension.

  Single Value = value1

  Multiple Values = value1^value2

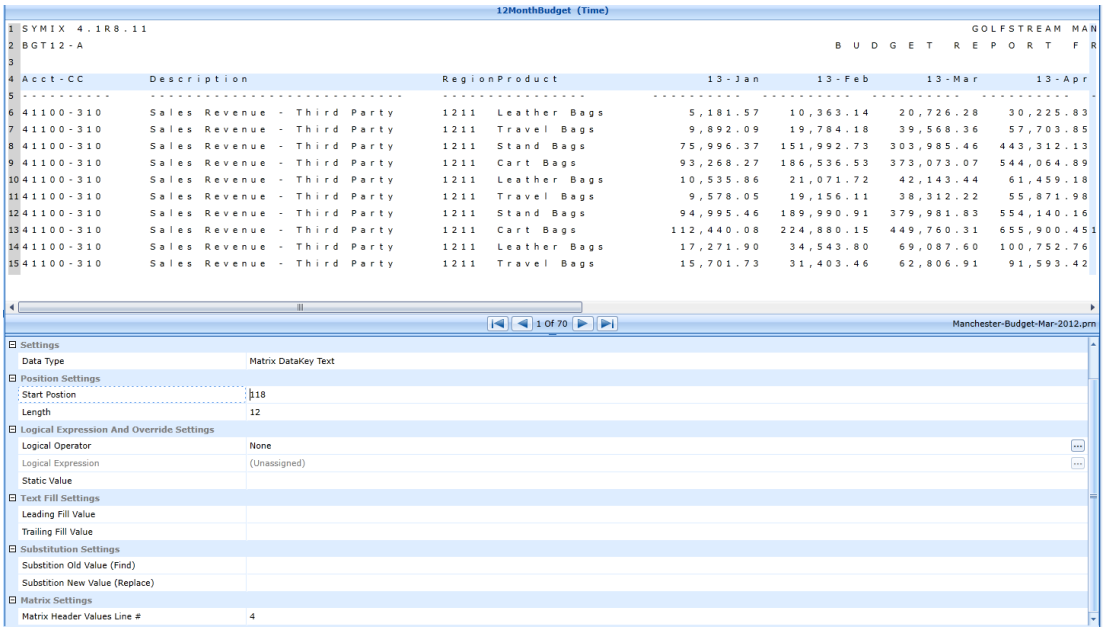
  Empty String | Null|

  Single Space |Space|

Matrix Settings

This setting is only available when Matrix is set as the Data Structure Type.

Matrix Header Values Line #: This setting indicates which row to look to for the dimension being matrixed. For example, if months are listed across columns on line four, and time is the intended Member to be matrixed, a 4 would be entered in this field.



Numeric Settings

These settings are only available in the Amount Source dimension which will help with the formatting and properties of the amount values.

- **Thousand Indicator**: Enter the character used to separate thousands in the value. For example, for the value 1,000 the Thousand Indicator is “,”. This can also be done by highlighting the character in the file and clicking 

.

- **Decimal Indicator**: Enter the character used to separate decimals in the amount value.

  This can also be done by highlighting the character in the file and clicking  .

Using Excel as a Data Source

An Excel file can be imported without having to configure the data source completely by setting Allow Dynamic Excel Loads to True and configuring an Excel template. See Loading Data With Excel Templates or CSV Files for more information.

Long Running Requests

Tasks that timeout can be resolved by OneStream Support. To contact OneStream Support, register at OneStream support.

For additional information, see OneStream Web API Endpoints in the *REST API Guide*.

Batch File Loading

You can import and process files all the way through the Workflow certification process. In addition, as the Workflow batch process is executed, the batch processing engine raises events used to monitor the processing and notify administrators and users of the status of the batch.

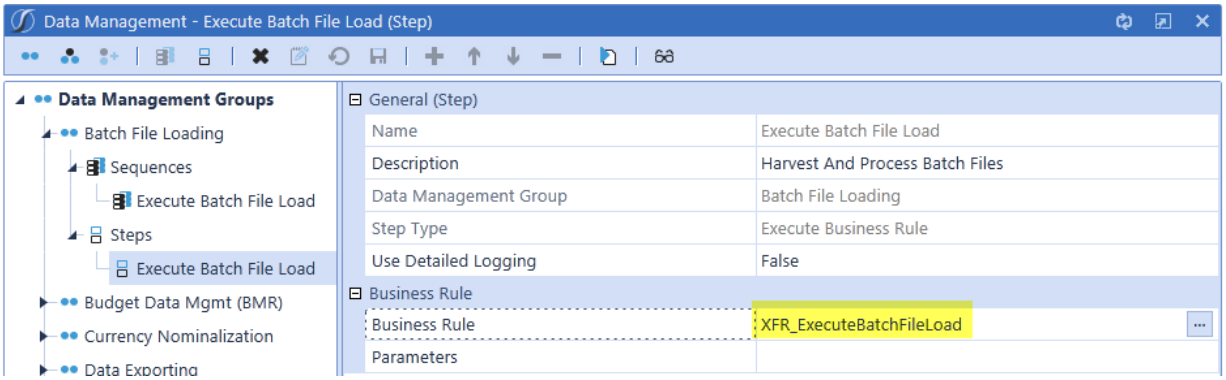
Setting Up and Using Batch File Loading

1. Create Batch Processing Extender Business Rule

   Batch file processing is executed by creating an Extender Business Rule that calls the OneStream API function BRAPi.Utilities.ExecuteFileHarvestBatch. This function also accepts switches that control the level of Workflow processing execution.

2. Create Data Management Sequence

   Batch file processing is executed by creating a Business Rule Data Management Step that calls an Extender Business Rule such as the rule example in step one.



3. Formatting File Names

   Batch processing requires each file being processed to use a specific file name format to tell the batch engine how to process the file (Format: File Id-ProfileName-ScenarioName-TimeName-LoadMethod.txt). See Batch File Name Format Specification later in this section for details on creating file names that comply with OneStream's required format.

4. Copy Files to Batch\Harvest Folder

   OneStream automatically creates a Batch\Harvest folder in each application's file share folder structure. This can be found in:

   System Tab|File Explorer|File Share|Applications|Application Name|Batch|Harvest

5. Execute the Batch

   After the files have been copied to the Harvest folder, execute the Data Management Sequence created in step one and the files will be processed.

   Manual Execution

   Navigate to Application Tab| Data Management, select the Batch Processing Data Management Sequence, and click  to run the sequence.

6. Evaluate Batch Processing Results

   Task Activity Logging

   Each batch process creates a detailed Task Activity entry that provides overall status results for the batch as well as detailed information about each processed file and Workflow step.

   Batch Function Return Value

   In addition to writing information to the Task Activity Log, the batch processing function returns a detailed results object to the Extender Business Rule. This object provides information and can be programmatically evaluated and used to create custom reporting and notification.

7. Scheduling Batch Processing

   Batch file processing can run using the Windows Task Scheduler or any other scheduling tool an organization may use. This is accomplished by creating a PowerShell Script to execute a batch processing Data Management Sequence when called from the specific scheduling tool.

See Data Management Automation through PowerShell in Foundation Guides for more information on executing OneStream Data Management Sequences from PowerShell scripts.

Batch File Name Format Specification

The information below provides a detailed list for each segment of the required batch file format.

Field Layout

File ID-ProfileName-ScenarioName-TimeName-LoadMethod.txt

aTrialBalance-Houston;Import-Actual-2011M1-R.txt

Field Definitions and Values

File ID

Any text value used for file identification and controlling sort order.

Profile Name

A valid Import Child Workflow Profile name. Use a ; to delimit Parent and Child Profile names.

Houston.Import becomes Houston;Import

Scenario Name

This is a valid Scenario name passed to Data Sources using the Dimension data type Current DataKey Scenario. C can be passed as a substitution variable to reference the Scenario name passed in the function call: HarvestAndProcessFiles. G can be passed as a substitution variable to reference the Global Scenario name set for the application.

Using Current Scenario: A-Houston;Import-C-2011M1-R.txt

Using Global Scenario: A-Houston;Import-G-2011M1-R.txt

Time Name

This is a valid Time name passed to Data Sources using the Dimension data type Current DataKey Time. C can be passed as a substitution variable to reference the Time name passed in the function call: HarvestAndProcessFiles. G can be passed as a substitution variable to reference the Global Time name set for the application.

Using Current Time: A-Houston;Import-Actual-C-R.txt

Using Global Time: A-Houston;Import-Actual-G-R.txt

Load Method

This is a value used to control how the file is loaded.

R = Replace, A = Append

Workflow Stage Import Methods

Workflow Stage Import has three primary methods of integrating data from the Stage Engine to meet the reporting requirement for each OneStream application model.

- **Standard**: Highly durable and auditable, stored details that target Finance Engine Cubes.
- **Direct**: In-memory, performance-focused, no storage of record details, that target Finance Engine Cubes.
- **Blend**: In-memory, high-performance import designed to blend the multi-dimensional structure with transactional data. No storage of record details or targeting external relational tables needed.

Standard Import Methods

A Consolidation Model requires that data be durable and auditable, as OneStream is functioning as the “book-of-record” for financial reporting. In this regard, the Stage Engine’s Standard Type Workflow is ideal in that each Import performed has its source and target records stored in Stage tables. This allows every current and historical period loaded to be audited and analyzed at the source and target record level used for loading data to the Finance Engine. The stored tables also allow detailed analysis from the Finance to the Stage using the Drill-Down feature.

Direct Import Methods

Certain Planning, Forecasting or Operational models may not require detailed audit and historical durability of data, with the resulting database overhead, that is required of the Consolidation Model. The Stage Engine’s Direct Load Type Workflow is specifically designed to support the needs of data that is more operational in nature. The Direct Load Type’s in-memory processing, and lack of storing Source and Target record detail, enhances its performance in processing compared to the Standard Type Workflow. The performance benefits of not storing source and target records is also what makes the Direct Load Type inappropriate for Consolidations, where detailed audit, history and drill-back is required.

Blend Import Methods

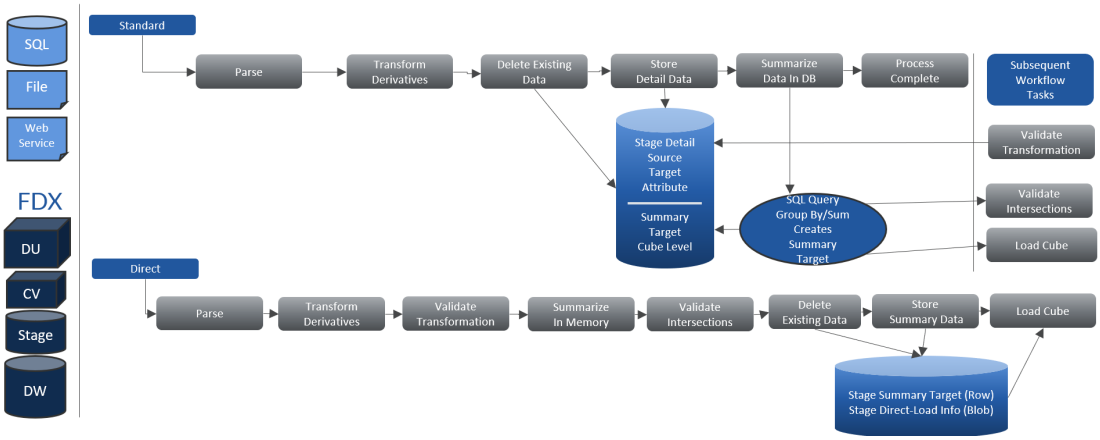
The Blend Type Workflow is an in-memory process and the integration method used by the BI Blend Engine, which writes to external relational database tables and not to a Finance Engine Cube.

The BI Blend Engine is a key element of Analytic Blend models as it is a read-only aggregate storage model. The purpose is to support reporting on large volumes of data that is not appropriate to store in a traditional Cube, such as transaction or operational data.

The Blend Type Import rationalizes the source data into a structure that is uniform and standardized for reporting by leveraging Cube Dimensions, deriving the metadata and aggregation points for the resulting BI Blend relational tables. This enables the transaction content stored in the relational tables to be aligned with the Finance Engine Cube data through common metadata and aggregation points for Analytic Blend Reporting. Refer to the *BI Blend Design and Reference Guide* for more detail.

Stored versus In-Memory Workflow Imports

The primary difference between the Standard and in-memory processing Import methods is how the source and target record details are stored in the Stage tables.



Guidelines on Volumes and Limits

Workflow Import performance is optimized by designing a Workflow structure that considers the data volumes of the total number of source records and the resulting total number of summarized target records. The transformed source records are summarized in the **StageSummaryTargetData** table. The volumes can be managed by structuring the data source loads across Workflows, gaining advantages in parallel processing.

Limit	Description
Row Limit Per Workflow	24 million summarized records
Best-Practice Recommendation	1 million summarized records

Example	Description
Single Workflow Results	24 million summarized records
Best-Practice Solution	Parse the file to be used across 24 partitioned Workflows

The benefits of efficient Workflow structure using partitioning when working with large data sources are:

- Performance gains through parallel processing
- Shorter processing times
- Faster mapping and error correction
- More transparent data validation

Standard Workflow Record Analysis

OneStream Task Activity / Load Cube presents the details of the total number of summary records in the **SelectStageSummaryTargetRows** entry.

Direct Workflow Record Analysis

The Direct Load Status / Execution Status screen displays the Summary Row Count for each load process to display the total number of summary records generated.

Blend Record Analysis

The Blend process differs from other Stage Import processes by generating additional data records, or rows, rather than one-to-one or summarizations. This is because the Analytic Blend designs require the generation of aggregation points, which add to the source rows. Detailed analysis of the BI Blend Processing Logs, Live Row Count statistics, and Task Activity BI Blend Load and Transform help guide the requirements for the application and systems. Refer to the *BI Blend Design and Reference Guide* for more detail.

In-Memory Workflow Imports

Direct Load Workflow Import

Workflow Import Type is optimized for performance by combining the Parse/Transform, Validate Transformations and Intersections, and Load Cube into a single Workflow process. This consolidation of processes, and resulting performance gains, is achieved by functioning in-memory and bypassing the overhead of writing and storing Source and Target record details to Stage database tables. The in-memory processing limits certain functionality and therefore may not be an appropriate solution for all model requirements. All other Workflow functionality remains as part of the Direct Type, such as all Business Rules, Transformation Rules and Derivative Rules.

Direct Load Use Cases

The Direct Load Workflow is designed for data that has a high frequency of change and does not demand durability for audit and history.

Common Uses

- Data Integrations where the OneStream Metadata and Source System are mirrored, allowing “* to * “ Transformation rules to pass-through all records, minimizing the need Drill-Down or Transformation analysis.
- Data that is “disposable” in nature. Typically, this may be data that has a high frequency of changes and may only be valid for a short time. Perhaps only valid for one to seven days.
- High-volume data loads, such as nightly batch loading, where optimal performance is desired. Such data is commonly deleted and reloaded frequently.
- Extended Application data moves, where data from a detailed application feeds a summarized target application.
- Where target data in OneStream is not required to be durable “book-of-record” data.

Important Limitations

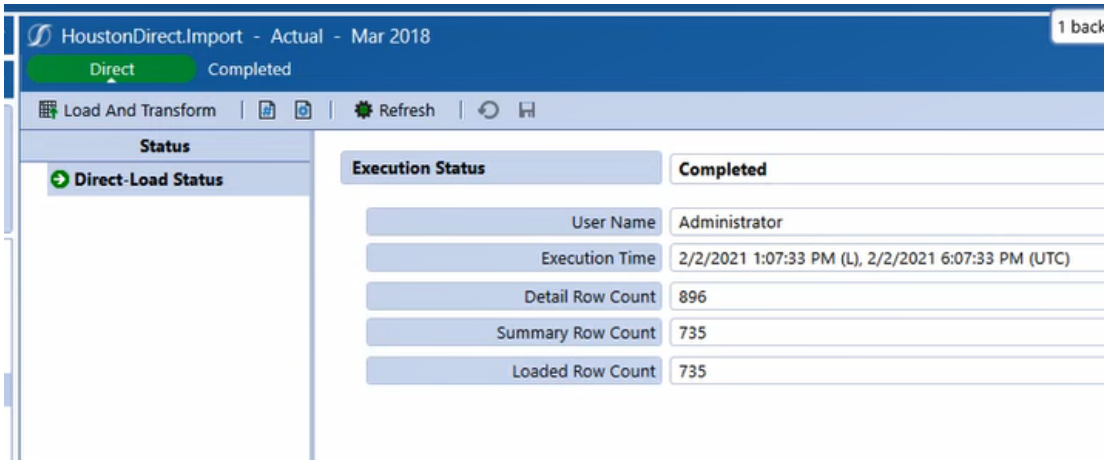
A key differentiator of the Direct Load is that it does not store source and target records in Stage Database tables. This, by design, will eliminate the audit and historical archiving of Workflow Activity. Other limitations as a consequence of the in-memory Workflow are that the Drill-Down feature will not function to support analysis of records between the Finance and Stage tables.

- Direct Load Type is not appropriate where:
    - Source file import history is required for historical reference.
    - Transformation Rule history is required for historical reference.
    - Drill-Down from Finance Engine is required.

- Text based View member values are required to be file based.
- Data loads are required to Append to prior Imports.

- Direct Load Type does not support historical audit of workflow history, such as Import and Transformation Rule history.
- Direct Load Type does not support Re-Transform as Import records are not stored data. Data must be re-loaded.
- Transformation and Validation analysis and map correction is limited to 1000 error records per load.
- Data files cannot load to Time and Scenarios beyond the current Workflow Scenario and Time. The data record's Time and Scenario being loaded must match the Workflow Scenario and Time.

As Direct is an in-memory Workflow with only a single step for the data integration process, Load And Transform. The Direct Load Execution Status screen provides statistics to analyze the Workflow's performance. These key statistics are helpful in determining if the Workflow design is supportive of best-practice designs to optimize application performance.



- **Detailed Row Count**: The total number of Data Source and Derivative Rule records.
- **Summary Row Count**: The total number of records summarized in the Transformation process.
- **Loaded Row Count**: The recorded number of records loaded to the Finance Engine target Cube, which should always equal the Summary Row Count.

Direct Load Transformations and Validations

The Direct Load Workflow's in-memory processing results in Transformation and Validation errors that are not stored being stored in a table. The total number of errors that can be processed and presented in Validation is limited to 1000 records. If the total number of errors exceeds 1000, the data must be re-loaded to re-execute the Transformation and Validation process to generate the next batch of errors, at a maximum count of 1000 records per load.

- Total Direct Load Error Storage Limit = 50,000 records
- Error Presentation Limit = 1000 records presented per Import load

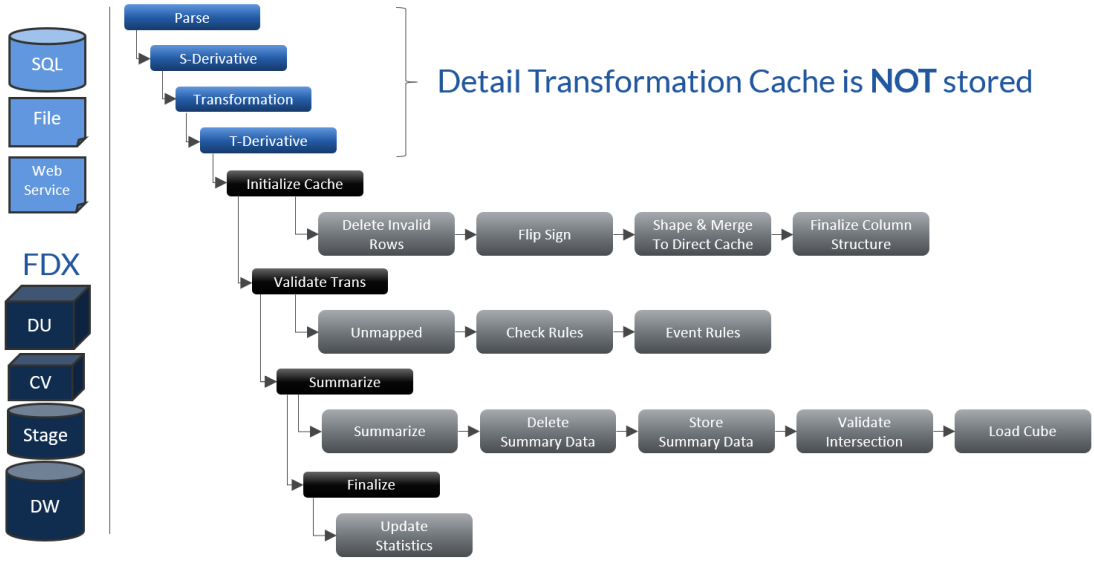
Integrations with high complexity and mapping may benefit by having a "development", Standard Workflow, to finalize the core Transformation Rules. A Standard Workflow supports pageable Validation and Intersection Error analysis, as well as the ability to Retransform source data that the Direct Load does not. The Standard Workflow also provides the Drill-Back from the Finance Engine to Stage that may streamline the data validation process. Once the core Transformation Rules are developed, a "production" Direct Load Workflow can be used, managing only the Validation exceptions.

Direct Load Workflow Implementation

The Direct Load Workflow Design and Requirements should consider strategies for configuration and data validation. Designers should consider the impact of the volume of source records and the complexity of mapping because of the Direct Load Workflow's lack of stored source and target details. Additionally, the Direct Load Transformation and Validation of records is limited to 1000 records per data load. Therefore, multiple imports may be required to resolve all mapping errors.

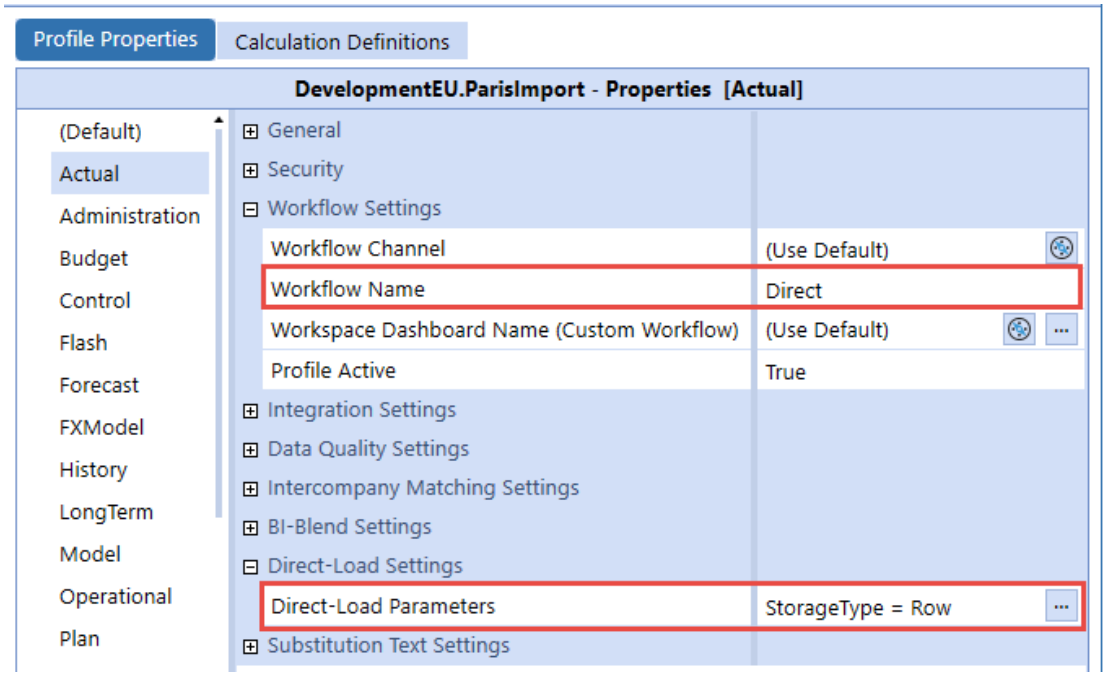
- Tightly coupled source metadata to OneStream metadata allows the use of " * to * " Mask maps to easily associate source data to cube data.
- Integrations with complex mapping may best make use of a Standard Workflow as "development" to provide transparency and as a platform for debugging.

Direct Load Data Flow



Direct Load Workflow Configuration

The Direct Load Type Workflow is an Import channel Workflow configured using various combinations of the “Direct” Workflow Name.



Row or Blob Summarized Data Storage

The Direct Load has two options for managing results of the Transformed records that are ultimately loaded to the Finance Engine, the default being Row.

- **Row**: Transformed summary records are stored in the StageSummaryTargetData table.
- **Blob**: Summarized records are not stored directly in the StageSummaryTargetData, but as a serialized Byte array stored in the StageDirectLoadInformation.

Sample Blob Resource Analysis

Rows	Size	Serialization & Write Time	RAM Utilization
12 Million	300-400 MB	2 Min 45 Sec	6 GB
24 Million	650-800 MB	5 Min	12 GB
36 Million	1 GB+	7 Min	21 GB

Direct Load Error and Troubleshooting

Direct Load Import Source File Data Keys must match the Workflow Data Keys. Unlike the Standard Workflow Import, the Direct Load Import cannot load data beyond the current Workflow. As an example, an Import to M1 cannot contain records for M2 in a Monthly Workflow.

Blend Workflow Import

The Blend Workflow Type utilizes the BI Blend Settings, which vary by Scenario Type. These settings allow the BI Blend Administrator to define, and optimize, the generation BI Blend tables to meet the reporting requirements.

The BI Blend Settings contain core properties used to design and structure the relational tables created by the BI Blend Engine.

BI Blend Settings Property Groups	Description
Data Controls	Defines the core data source and output structure and design of the relational tables.
Aggregation Controls	Settings to leverage Cube Dimension metadata to filter and define the relational tables.
Performance Controls	Server management and optimization settings.

BI Blend Use Cases

BI Blend is intended to provide focused reporting tables that are aggregated and saved as stored parent intersections for fast reporting at a later point in time. BI Blend is not intended to replicate and entire cube, but rather focus on specific reporting use cases that result in many parent intersections that would not perform well under Calc-On-Fly aggregation.

BI Blend also solves for use cases that are not pure analytic reporting problems. By leveraging OneStream hierarchies, along with BI Blend configuration settings, it is possible to aggregate on a few dimensions (Entity or Account as an example) while including transaction information (Invoice number) that is not associated with a cube. The ability to combine the dimensional structure with transaction details allows for selective enrichment of transactional data. Refer to the *BI Blend Design and Reference Guide* for more detail.

Transformation Rules

Transformation rules help map data from source systems to the financial model. The different member scripts and how you can use them with transformation rule types are described below. These are listed in the order in which the rules run during the Validate step in a workflow.

Transformation Rules Toolbar

Create Group

Use this to create a Transformation Rule Group under each dimension.



Create Profile

See Transformation Rule Profiles.



Manage Profile Members

See Transformation Rule Profiles.



Create and Populate Rule Profile

Selecting this will create a Transformation Rule Group and a Transformation Rule Profile with the same name. The Rule Profile will already be populated with each Dimension Rule Group and update as the Group is updated.



Delete Rule Profile and Groups

This can only be done by selecting a specific Transformation Rule Profile. It will delete the Transformation Rule Profile and all non-shared Groups within the Profile.

 **Delete Selected Item**

Use this to delete one item such as one Rule Group or Profile.

 **Rename Selected Item**

This allows a user to rename Transformation Rules, Groups or Profiles.

 **Cancel All Changes Since Last Save**

This will undo any unsaved changes made.

 **Export Selected Group to a TRX File**

Use this to export a Transformation Rule Groups to use in another application or group. This exports a comma delimited file which are valid for only certain Transformation Rule designs. Logical Operators and complex expressions are not supported. Application Tools, Load/Extract of application *.XML files should be used to manage Transformation Rules.

 **Import a TRX file into the Selected Group**

Use this to import Transformation Rule Groups from another application or group. This comma delimited file format may not be valid for all Transformation Rule designs. Properties such as Logical Operators and complex expressions are not supported. The *.TRX import is intended as a utility to import Transformation Rules from compatible legacy systems. Application Tools, Load/Extract of application *.XML files should be used to manage Transformation Rules.

General Settings

General

Name

The name of the Transformation Rule group.

Description
A short description of the rule or where it is used.

Security

Access
Members of this group will have access to the Transformation Rule group.

Maintenance
Members of this group have the authority to maintain the Transformation Rule group as well as load/extract from the Transformation Rules page.

> **NOTE:** Click  and begin typing the name of the Security group in the blank field. As the first few letters are typed, the groups are filtered making it easier to find and select the desired group. After the Group is selected, press CTRL and double-click to add the correct name into the appropriate field.

Settings

Cube Dimension Name
The specific dimension to which the rule group is assigned.

Mapping Types

Mapping Types allow the data to be mapped in different ways with the possibility of using conditional rules, wild cards, ranges and others.

> **IMPORTANT:** The characters **!**, **?**, and **|** are used in operations for Transformation Rules. These characters will cause errors if used with Source Members or Target Members.

One-to-One

One-to-One mapping allows one Source dimension member to be mapped or transformed to one target dimension member explicitly. No member scripts are used.

Source Value
This is the value for the related Cube dimension in a defined data field.

Description (Optional)
This is a description of the mapping rule.

Target Value
The Dimension Library member to which the Source Value is being mapped or transformed.

Order (Optional)
The One-to-One mapping rules will be processed in the order of the (defaulted) alpha numerical sort. The Order field allows a value to be assigned to a record which will allow a custom sort order of the mapping table.

Composite

Composite mapping is a way to do a map conditionally. Dimensional tags may be used to include another dimension in the evaluation.

The * character is used to represent any number of characters. The ? character acts as a placeholder for a single character. In the following example, any similarly formatted account starting with 199 and an entity of Texas would be mapped to a specific target.

> **Example:** A#[199?-???*]:E#[Texas]

In the following example of an entity composite rule using the ? character, records for all entities with accounts that start with H that have 10 or fewer characters would be captured, for example, accounts named Head, Heads, and Headcount.

> **Example:** E#[*]:A#[H??????????]

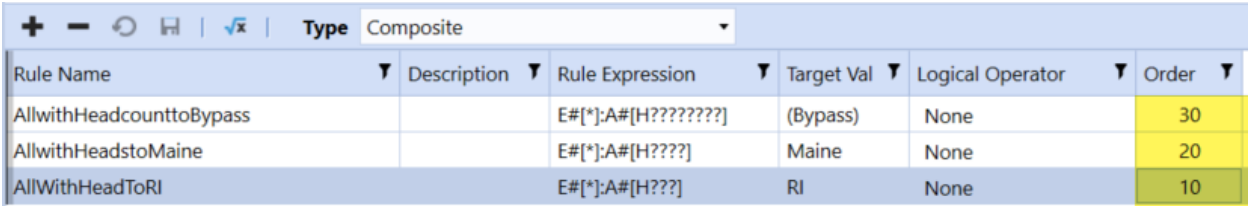
Composite rules stop processing when a record meets the criteria. So, if the record meets the criteria for more than one rule, set the order to ensure they are mapped correctly.

For example, using the following records, we can map those with Head to the entity Rhode Island, Heads to the entity Maine, and Headcount to be bypassed by setting the order from the shortest account name (narrowest search) to the longest account name (broadest search).

Amount (Raw)	Sourc	Eastern	Time	View	SubSidAccour
10.00	Files	Connecticut	(Current)	YTD	Head
10.00	Files	Mass	(Current)	YTD	Head
10.00	Files	VT	(Current)	YTD	Head
1,000.00	Files	Connecticut	(Current)	YTD	Headcount
1,000.00	Files	Mass	(Current)	YTD	Headcount
1,000.00	Files	VT	(Current)	YTD	Headcount
100.00	Files	VT	(Current)	YTD	Heads
100.00	Files	Mass	(Current)	YTD	Heads
100.00	Files	Connecticut	(Current)	YTD	Heads

Set the mapping to run in the following order:

1. E#[*]:A#[H???] to map all accounts that start with H and have 4 or fewer characters
2. E#[*]:A#[H????] to map all accounts that start with H and have 5 or fewer characters
3. E#[*]:A#[H??????????] to map all accounts that start with H and have 10 or fewer characters



Type Composite

Rule Name	Description	Rule Expression	Target Val	Logical Operator	Order
AllwithHeadcounttoBypass		E#[*]:A#[H????????]	(Bypass)	None	30
AllwithHeadstoMaine		E#[*]:A#[H????]	Maine	None	20
AllWithHeadToRI		E#[*]:A#[H???]	RI	None	10

First, the records for the Head account would map to Rhode Island. Then, the records for the Heads account would map to Maine. And last, the records for the Headcount account would be bypassed.

Amount	Eastern	Time	View	SubSidAccour
3,000.00	(Bypass)	2023M12	YTD	Headcount
300.00	Maine	2023M12	YTD	Heads
30.00	RI	2023M12	YTD	Head

Range

A range mapping gives the upper limit and lower limit of a range. Any member that falls within this range will be mapped to the corresponding target member.

Range rules do not stop processing when a record meets the criteria. If the ranges overlap and a member falls within multiple ranges, that member will be mapped only to the final range that is processed. For a member in multiple ranges, set the order so that the last rule run is applied.

If an administrator wants the range of source Accounts from 11202 to 11209 to map to Account 12000, enter 11202~11209 under Rule Expression with ~ as the separator. Range rules use character sets not integers. Use balanced character sets to ensure the full range is captured. For example, a range of 4 to 3000 must be entered as 0004~3000 under Rule Expression.

List

List mapping allows the user to create a delimited list of members that all map to the same target.

If an administrator wants the list of Accounts 41137, 41139 and 41145 to map to Account 61000, enter 41137;41139;41145 under Rule Expression with ; as the separator.

Mask

Wild cards are used in mask mappings. The wild card characters for these mappings are * and ?. The * character is used to represent any number of characters. 27* would capture 270, 2709, or 27XX-009. The ? character acts as a placeholder for a single character. 27?? Would capture 2709, but would not capture 27999 or 2700-101.

Double-sided wild cards can be used as well in Mask transformation rules. For example, *000* would capture any account number with characters before and after the 000 sequence.

The following properties are standard for Composite, Range, List and Mask Mapping Types.

Rule Name
This is a unique name assigned to each mapping rule that will also determine the default sort and processing order.

Description (Optional)
The description of the mapping rule.

Rule Expression
This is where the specific source field processing rule is placed. For example, 27* would capture 270, 2709, or 27XX-009.

Target Value
The Dimension Library member to which the Source Value is being mapped or transformed. The use of wild cards to define Target Values is not recommended. The following exceptions apply to Target Value wild card usage:

- The ? character is not supported.
- The * character is not supported when used as a prefix (left side) to a target value.
- The * character used as a suffix (right side) will yield the Source value.

Logical Operator
This provides the ability to extend a normal mapping rule with VB.NET scripting functionality.

Expression Type:

None (Default)
No script is assigned or employed for this related transformation rule.

Business Rule
This selection is used when .NET scripting is needed for the dimension and the script is available within the Business Rule Library.

Complex Expression
This selection is used when .NET scripting is needed for the dimension but will not need to be used elsewhere. The script used in the complex expression will only be available within that dimension.

Order (Optional)
The Order field allows a value to be assigned to a record which will allow a custom sort order of the mapping table.

Derivative Rules

Derivative rules are used to cleanse and enhance data by filtering the source dataset. They apply logic to Stage data and generate additional rows of data in the Stage environment, then group or aggregate records together into a new single record that stores an aggregated amount. Derivative rules can also perform calculations and store the resulting amounts in the newly derived records.

Derivative rules provide functionality to filter records, perform calculations, and derive new records during data processing, such as:

- Filtering records you want to include in your derivative rule
- Grouping or aggregating values/amounts together and storing the results in new records
- Performing calculations on the amount fields using operators or business rules and storing the results in new records
- Performing calculations based on dates. For example, calculating date differences and storing the derived results in new records. This is only for BI Blend. See "BI Blend Derivative Rules" in the *BI Blend Design and Reference Guide*.

> **NOTE:** Derivative rules can be viewed in Task Activity. If source or target derivative rules are used, Task Activity displays when they are processed and how many rules there are.

Source Derivative Rules

These are created to apply logic defined by the inbound source data in the Stage environment before the data is transformed. The output of this logic is stored in new records. You can use temporary member names for these records and then provide mapping in the transformation rules. The derived rows generated by source derivative rules are added to your source data records. For example, if you are importing data using a standard Import, Validate, Load Workflow Profile, the rows display on the Import page. Source derivative rules are processed before transformation rules.

Target Derivative Rules

These apply logic defined by the post-transformed Stage environment data. The rules are processed after the transformation rules have been applied. The logic outputs are stored in new records. For example, if you are importing using a standard Import, Validate, Load Workflow Profile, the newly-derived rows display on the Validate page. As transformations have already been applied, the members used for these derived records must exist in the cube.

BI Blend Derivative Rules

BI Blend derivative rules are specifically designed for use BI Blend imports. A Blend import stores data in an external Blend database rather than the cube, and the source data set is not retained in Stage. However, you can still apply source and target derivative rules to a Blend import. Blend derivative rules are the last rules to be processed but, because the data is never loaded, the members used for the derived records do not need to exist in the cube.

See "BI Blend Derivative Rules" in the *BI Blend Design and Reference Guide.*

> **NOTE:** Source and target derivative rules can be processed by both the standard Import Workflow Profile import process and the Blend Workflow Profile import. Blend derivative rules can only be processed by a Blend Workflow Profile import.

> **IMPORTANT:** The new records created by derivative rules are stored against members. For source derivative rules, you can use members that don't exist in the cube but you must map these members to cube members in the transformation rules. For target derivative rules you must use members that exist in the cube. For Blend derivative rules you can use members that don't exist in the cube.

Create Derivative Rules

To create a derivative rule, complete the following steps:

1. Go to **Application** > **Data Collection** > **Transformation Rules** > **Rule Groups** > **Derivative**.
2. Select an existing rule group, or select the **Create Group** icon to create a rule group.
3. From the **Type** drop-down menu, select a rule type. The derivative rules displayed reflect the rule type selected.
4. Select the **Insert Row** icon and enter rule information. See Derivative Rule Properties.
5. Select the **Save** icon.

Derivative Rule Properties

Rule Name
The name of the rule. The name must be unique within the rule set.

Description (Optional)
A description of the rule. A description is recommended to detail the derivative rule's function. If you are running a BI Blend import, the description field displays in the Label column when viewing derived records in the BI Blend table.

Rule Expression

A rule expression written in derivative rule syntax. This is where the derivative rule itself is defined and is primarily used to select the records for processing by the rule. If you are running a BI Blend import, the Text Value field in the Blend table displays the rule expression. See Derivative Rule Expression

Logical Operator

The logical operator enables you to specify what calculation should be applied to the amounts in the selected records. These values are placed into new derived records. The original records are not changed. See Logical Operators.

To specify a logical operator, double-click the field to display the Derivative Expression Editor dialog box. The expression types available depend on the derivative rule type selected.



Derivative Type

Select a derivative type. This determines when a rule is run and whether the records are stored. You can choose from the following options:

- **Interim**: This is not stored in the Stage area and cannot be mapped to a target member. It can be used within other subsequently run derivative rules.
- **Final**: This is stored in the Stage area and available to be mapped to a target member.

- **Check Rule**: This is a custom validation rule that uses the same syntax as member filters and can be applied to the source data during the Validation task of the workflow.
- **Interim (Exclude Calc)**: This is similar to Interim but is excluded in other derivative calculations.
- **Final (Exclude Calc)**: This is similar to Final but will be excluded in other derivative calculations.
- **Check Rule (Exclude Calc)**: This is similar to Check Rule but is excluded from other derivative calculations.
- **Inactive**: This deactivates the rule, and it is not processed during import.

Order
This is used to determines in which order rules are run. For example, a rule with an order value of ten will run before a rule with an order value of twenty.

Sequencing

Derivative rules can be sequenced. The results of one rule can be used by another rule by referencing the member name used to store the derived records. The Derivative Type and Order columns are used together to sequence derivative rules. The interim derivative type indicates that this rule is part of a sequence. The final derivative type indicates that the rule is last in the sequence. Rules that run after a final derivative rule type will be part of a different sequence.

Interim and final derivative types produce different results depending on which Workflow Profile name is being used. The following tables display two different Workflow Profile names. They also display the derivative rule type and whether the interim records are written and where they are written.

Interim Derivative Rule

Derivative Rule Type	Interim Records
Workflow Profile name: Import, Validate, Load	
Derivative (Source)	Written to Import only
Derivative (Target)	Written to Import only
Derivative (BlendUnit)	Not written to Import or Stage
Workflow Profile name: Blend	
Derivative (Source)	Not written to BI Blend table
Derivative (Target)	Not written to BI Blend table
Derivative (BlendUnit)	Written to BI Blend table

Final Derivative Rule

Derivative Rule Type	Interim Records
Workflow Profile name: Import, Validate, Load	
Derivative (Source)	Written to Import and Stage
Derivative (Target)	Written to Import and Stage
Workflow Profile name: Blend	

Derivative Rule Type	Interim Records
Derivative (Source)	Written to BI Blend table
Derivative (Target)	Written to BI Blend table
Derivative (BlendUnit)	Written to BI Blend table

> **NOTE:** This is helpful if you are sequencing your derivative rules from source to target to BlendUnit. You need to know if the derived members must be referenced using their original name or a different name due to transformation mapping.

Derivative Rule Expression

Derivative rule expressions determine how derived records are grouped. Groupings occur on an attribute, unless the rule expression is explicitly set to None. The rule expression is also used to determine which member the rows generated by the derivative rule are stored against.

The following dimensions can be defined in the rule expression with the corresponding token:

Dimension	Token
Entity	E#
Consolidation	C#
View	V#
Account	A#

Dimension	Token
Flow	F#
Origin	O#
Intercompany	IC#
UD	UD1#-UD8# or U1#-U8#

The part of the rule that determines which member the derivative record is stored against is defined to the right of the equals sign.

> **Example:** A#[10*]=AllTest. The result of the logical operator on the subset of records is stored against an account member called AllTest.

Derivative Rule Expression Character Options

Derivative rule expression character options can be used to select records from the source data set and dictate the naming convention for the new members used to store derived records. Wild card characters can be used to select multiple member names. For example, an asterisk can be used to replace some or all the characters in a member name.

> **Example:** A#[10*]=AllTest. In this example, the Account dimension is used to select the records from the source data set. A#[10*] selects all members of the Account dimension in the source data set that start with 10, regardless of other characters in that name.

Naming Convention Characters for Derived Records

This describes the characters and naming conventions that can be used to generate new member names for derived records. Suffixes and Prefixes can be added to existing member names and used for the derivative records. Use the following characters and syntax:

- **<<_Suffix**: This can be used to create derived records where the name the records are stored against is the selected member name with the suffix attached.

  > **Example:** A#[*]<<_YTD. This selects all members in the Account dimension and generates additional rows where the account member will have the same name but with a _ YTD suffix.

- **<<Prefix_**: This can be used to create derived records where the names the records are stored against are existing member names with the prefix attached.

  > **Example:** A#[*]<<New_. This selects all members in the Account dimension and generates additional rows where the account member has the same name with the New_ prefix.

Common Records

Rule expression syntax can be used to make records common so that their amounts are summed together for the purpose of a derivative rule calculation. The dimension used to define the members selected in the first part of the rule expression determines which records are processed.

You may have a particular dimension in your source data set with more than one member name that is not being used as the initial defining dimension in your rule expression. However, an aggregated calculation for this dimension may also be needed. To ensure that dimensions in your source data set that have more than one member name are aggregated to a common record, their members need to be set to a common value.

> **Example:** A#[41010]=91010_DistOffset:U2#[*]=None:U3#[*]=None:U4#[*]=None:A1#[*]=All. Dimension members are selected using a wildcard and set to a common value so that all records have the same members. As a result, the records become common and their amounts are summed into a single record before the calculation set in the Logical Operator property is applied. By creating common records for UD2, UD3, UD4, and the Attribute 1 dimension, only a single summary amount is returned and stored against the 91090_DistOffset account.

Derivative Rule Syntax

This section provides examples of the syntax used to write derivative rules. The variation between the use of Source or Target derivative rules is by the definition of the rule. Source derivative rules reference the inbound record members. Target derivative rules reference the post-transformed records. In the rule expression samples, assume these rules will run in the order presented in the example.

Import	
Account	Entity
10200123	TX47
10299999	GB11
14000000	TX47
14300000	GB11

Validate		
Source Derivative	Transformation Rules	Target Derivative
A#[102*]= AdminExp	AdminExp	A#[*Exp]=TotExp
	ItExp	
	GenAdmExp	

Rule Expression	Expression Type	Derivative Type	Notes
A#[11*]=Cash	None	Final	Accounts that start with 11 aggregate to a new account called Cash and stored in Stage.
A#[12*]=AR	None	Interim	Accounts that start with 12 aggregate to a new account called AR, but are not stored.
A#[1300-000;Cash]=CashNoCalc	None	Interim (Exclude Calc)	The derivative account Cash is excluded because the calc is excluded. The CashNoCalc interim account is created as an aggregate ofaAccount 1300-000, but is not stored.

Rule Expression	Expression Type	Derivative Type	Notes
A#[1310-000;Cash]=CashIncludeCalc	None	Interim	The two accounts, 1310-000 and Cash, aggregate to equal the new derivative account called CashIncludeCalc because the calc is included. The semicolon is used as a list separator.

The following rules create additional rows in the Stage area when importing data based on logic.

Rule Expression	Expression Type	Derivative Type	Notes
A#[1000~1999]<<New_:E#[Tex*]=TX	None	Applies to All Types	This creates a new row in Stage for any account that falls between 1000 and 1999 in the source data and adds a suffix to it. Account 1010 creates a new row for Account New_1010. The end of the rule syntax shows that each entity name starting with Tex will be created as the entity called TX in these new Stage rows.

Rule Expression	Expression Type	Derivative Type	Notes
A#[2000~2999]>>_:Liability:U2#[*]= None:U3[*]=None:U4#[*]=None	None	Applies to All Types	This creates a new row in Stage for every account between 2000 and 2999 with a prefix. Account 2300 comes into a new row as 2300_ Liability. The rest of the rule means that all UD2, UD3, and UD4 dimension members will be set as the None member.

Rule Expression	Expression Type	Derivative Type	Notes
A#[3000~3999]@3:E#[Tex*]@1,1	None	Applies to All Types	This takes the first three digits of each account between 3000 and 3999 to create new rows in Stage. Each entity starting with Tex will be shown as T since the @1,1 syntax starts at the first position of the string and looks one character to the right.

Logical Operators

Logical operators are used to perform calculations based on your set of records. They are run when the derivative rule is processed, and the results are stored against the member defined in the rule expression.

> **Example:** A#[41010]=91010_DistOffset:U2#[*]=None:U3#[*]=None:U4#[*]=None:A1#[*]=All. In this example, if a logical operator of multiply is used, the results are stored against the A#91010_DistOffset member.

To specify a logical operator, double-click the field to display the Derivative Expression Editor dialog box. The expression types available depend on the derivative rule type selected.



Expression Types

Expression types are used to perform additional calculations on the transformed member's data. There are built-in mathematical expression types, business rule expression types, and complex expression types. The following expression types are available for source and target derivative rules types. Many of these expression types are also available for BI Blend derivative rule types. For additional operators available for BI Blend derivative rules, see "Logical Operators" in the *BI Blend Design and Reference Guide*.

Operator Suites	Individual Operators	Description
Aggregation	None	When the logical operator is set to None, the amount/values of common records are summed together and returned in a derived record. The values of any unique records are also returned in individual derived records. This is the default and no changes will be made.
Business Logic	Complex Expression Business Rule	The logic to process records can be written in a complex expression or a business rule of the derivative rule type.
Mathematical	Multiply Divide Add Subtract Absolute value Round (Precision) Modulo (Divisor) Power (Exponent) Log (P+1)	Simple mathematical operations can be applied to the amount/values in records and the results returned in derived records. Common records are summed before the mathematical operator is applied.

Operator Suites	Individual Operators	Description
Filtering	Create If (>x) Create If (<x)	A derived record is created from the imported records only if the amount is greater than/less than a set threshold.
Value replacement	Clip (Lower Bound) Clip (Upper Bound)	Values which are higher or lower than the set threshold are replaced with a set value in the derived record.

Business Rule

This expression type enables you to select a derivative business rule. When run, the rule is processed and the result returned and stored against the member defined in the derivative rule expression. This business rule must be a derivative rule type.

Complex Expression

This expression type enables you to write a formula for use with a specific derivative rule. It will run against the resulting derivative member's data.

Multiply

This multiplies the resulting derivative member's value by another specified value.

Divide

This divides the resulting derivative member's value by another specified value.

Add

This adds the resulting derivative member's value by another specified value.

Subtract

This subtracts a specified value from the resulting derivative member's value.

Create If > x
A derivative record is created if the value is greater than a specified threshold.

Create If < x
A derivative record is created if the value is less than a specified threshold.

Lookup

The Lookup transformation rule has a versatile configuration and can be used as a table for formulas, business rules or as a simple look up.

Transformation Rule Profiles

Once transformation rule groups are created, they are organized into transformation rule profiles. The profiles are assigned to workflow profiles.

Click  to create a new profile.

Click  to assign a rule group to a profile so users can select groups for a profile.

Under Rule Profile Settings, choose the cube and scenario type where the profile can be viewed and used when designing a workflow profile. Assign a rule profile to a workflow profile by clicking **Application** > **Workflow Profiles** > **Import** > **Integration Settings**.

Forms

Forms Channel Workflow

To minimize form maintenance, Cube View and Excel XFSetCell updates are not tied to specific forms. Association is at the Input Type level, not the individual form level. The Forms Input Type determines if you can update data from Excel. If the Forms channel is completed, but the process is not certified, you can import data from Excel using XFSetCell or a Cube View. If a cell is updated, the Analytic Engine traces the cell by:

1. Identifying the Workflow that owns the Entity.
2. Checking the Input Child Workflow Profile.

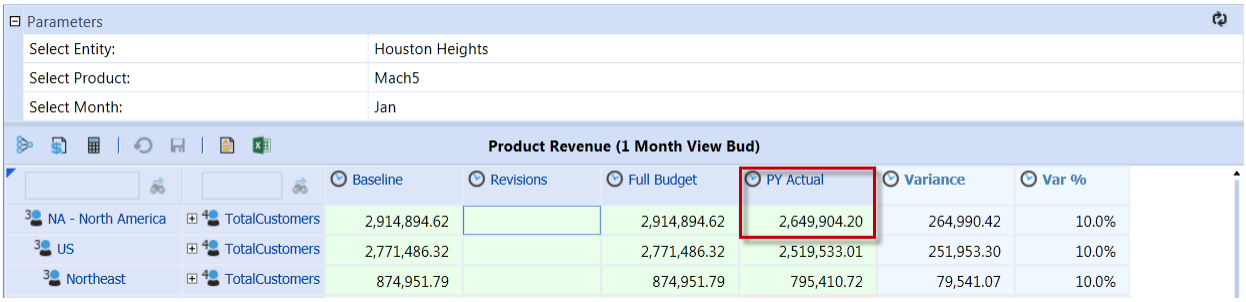
This determines if the Form Input Type is enabled for the Scenario Type. If not, the Form Input Type is disabled and cannot update cells from a Cube View in the web, Cube View, or XFSetCell function from Excel.

If Form Input Type is enabled, the Analytic Engine checks the full Workflow Status for the active Form Input Type. If the Workflow is locked or the Parent Workflow is certified, cells are not updated. If the Workflow indicates updates can occur, the Process Cube task of the Workflow and all ancestor Parent Workflows are impacted.

Form Allocations

Advanced Distribution

In the example below, an advanced distribution is used on a Product Revenue Form. This allocation will take the previous year's actual data, increase it by 20% and populate the current year's revenue budget revisions for all regions and customers.



Parameters

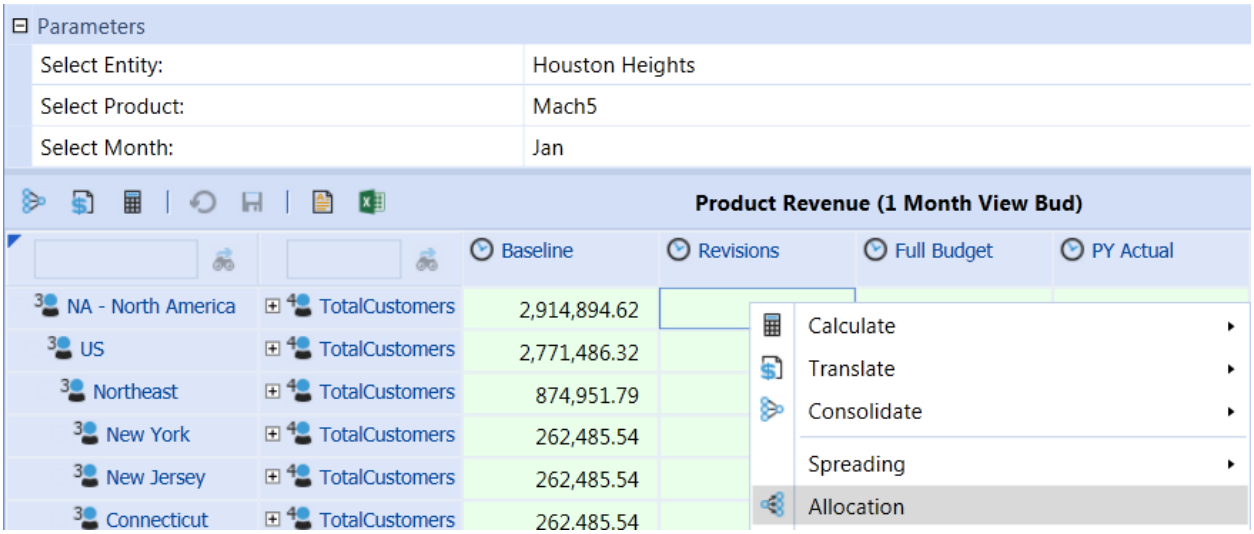
Select Entity:	Houston Heights
Select Product:	Mach5
Select Month:	Jan

Product Revenue (1 Month View Bud)

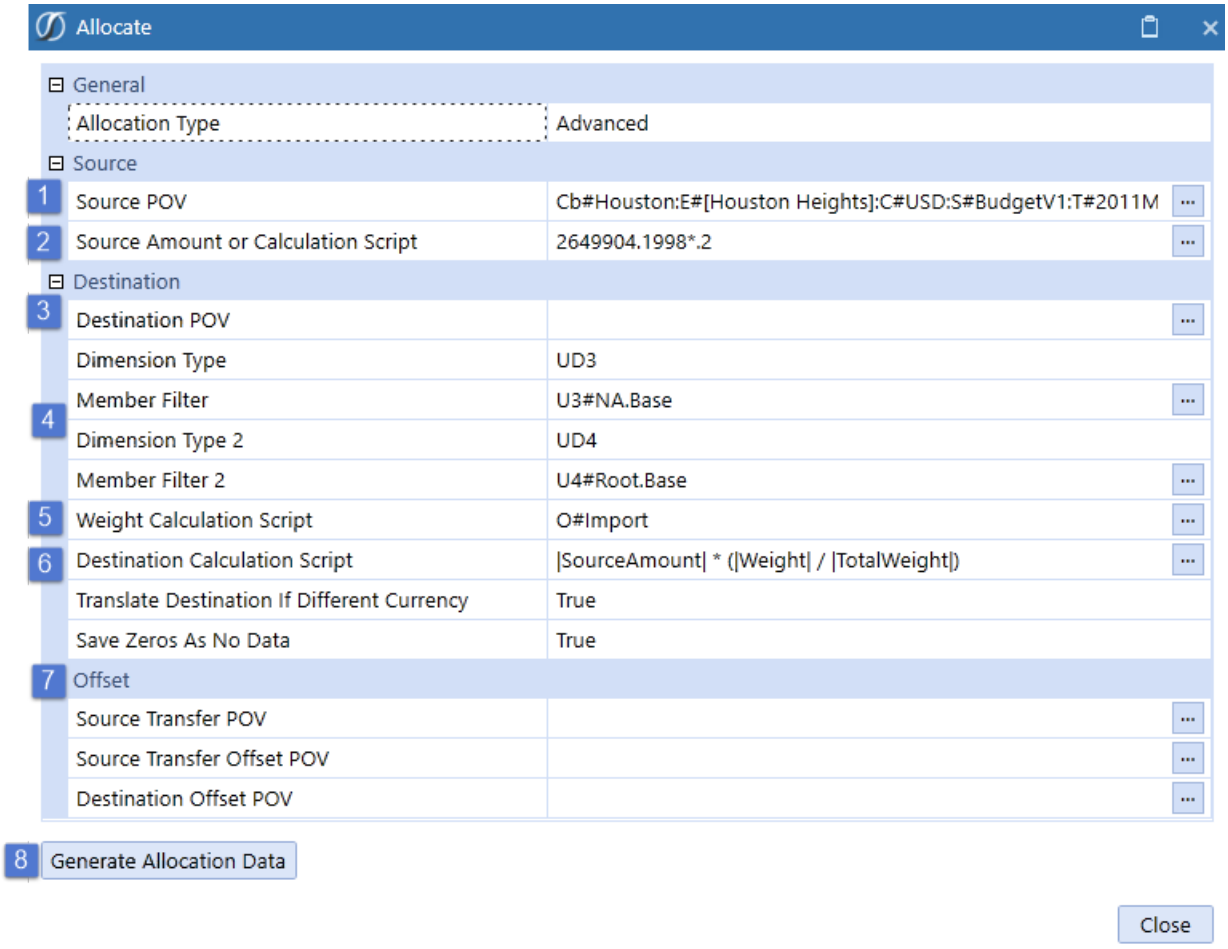
		Baseline	Revisions	Full Budget	PY Actual	Variance	Var %
NA - North America	TotalCustomers	2,914,894.62		2,914,894.62	2,649,904.20	264,990.42	10.0%
US	TotalCustomers	2,771,486.32		2,771,486.32	2,519,533.01	251,953.30	10.0%
Northeast	TotalCustomers	874,951.79		874,951.79	795,410.72	79,541.07	10.0%

The allocation data is being written to a form which will then populate the Revisions column. The sum of the Baseline and Revisions will then create the new Full Budget for each Region and Customer.

Right-click the first data cell in the Revisions column and select Allocation. This helps create the Source and Destination POV.



By default, the Allocation dialog will open to the last Allocation processed. Select the Allocation Type desired (e.g., Advanced).



1. **Source POV**

   The Source POV defaults to the last cell selected for Allocation. Every Dimension is represented in the POV. In this example, it defaults to the data cell under Revisions because that is where the allocation option was selected.

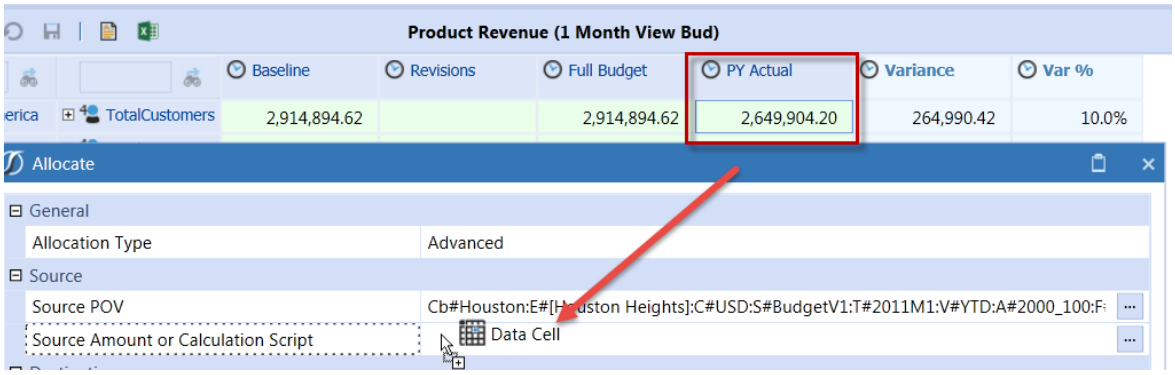
   Cb#Houston:

   ```
   E#[HoustonHeights]:C#USD:S#BudgetV1:T#2011M1:V#YTD:A#2000_
   100:F#None:O#Forms:I#None:U1#None:U2#Mach5:U3#NA:U4#TotalCustomers:U5#None:U6#None:U7#
   None:U8#None
   ```

Users can also select a data cell from the grid and drag and drop the cell's POV into this field. The Source POV is the default Source Amount for the allocation.

2. **Source Amount or Calculation Script**

   To override the Source POV value, enter a source amount or a calculation script. In this example, the PY Actual value is used for the Source Amount. Click on the cell in the grid and drag and drop the value into the Source Amount property. To increase this amount by 20%, the value is multiplied by .2.



3. **Destination POV**
   This is where the allocation is applied. In this example, the Destination POV is blank because it is using the same Members from the Source POV. Users can also drag and drop a data cell's POV.

4. **Dimension Type/Member Filters**
   These properties override the Destination POV and allow allocations to occur to several Members at a time. In this example, the UD3 (Regions) and UD4 (Customers) Dimensions are specified and therefore will override the UD3 and UD4 Members in the Destination POV.

5. **Weight Calculation Script**
   The Weight Calculation Script determines how the allocation is weighted. Any Members not specified in the script derive from the Destination POV. In this example, the weight for each Member is determined by the imported value in the Baseline column.
6. **Destination Calculation Script**
   The default Destination Calculation Script is |SourceAmount|*(|Weight|/|TotalWeight|). Additional calculations may be added to this field to customize how the weight calculation is performed. This example uses the default calculation.
7. **Offset**
   The offset properties are optional and not used in this example.
8. **Generate Allocation Data**
   After the allocation dialog is complete, click this button to see the allocation data before applying it.

The allocation results dialog provides information on all the allocation destinations, weight information, and displays all the data rows that will be updated upon selecting Save Allocation Data. Check the Show All Dimensions box to see every Dimension intersection for each data row. After the allocation data is saved, the form data will update and store the data to the cube.

Allocation Results

Source POV

Cb#Houston:E#[Houston Heights]:P#?:C#USD:S#BudgetV1:T#2011M1:V#YTD:A#2000_100:F#None:O#Forms:I#None:U1#None:U2#Mach5:U3#NA:U4#TotalCustomers:U5#None:U6#None:U7#None:U8#None

Source POV Amount 0.00

|SourceAmount| 529,980.84

|NumDestinations| 483

|TotalWeight| 2,914,894.62

Show All Dimensions

Amount	Is NoData	|Weight|	Type	UD3	UD4
4,980.47		27,392.60	Destination	Michigan	Active Hub
6,445.32		35,449.24	Destination	Michigan	Double Eagle
161.13		886.23	Destination	Michigan	East Sports
336.91		1,853.03	Destination	Michigan	EZ Sporting Goods
146.48		805.66	Destination	Michigan	Golf Hub
981.45		5,397.95	Destination	Michigan	Golf Pro

Save Allocation Data | Cancel

Results:

Product Revenue (1 Month View Bud)

		Baseline	Revisions	Full Budget	PY Actual	Variance	Var %
Midwest	TotalCustomers	397,676.06	72,304.74	469,980.80	361,523.69	108,457.11	30.0%
Michigan	TotalCustomers	99,419.01	18,076.18	117,495.20		117,495.20	
	Active Hub	27,392.60	4,980.47	32,373.07		32,373.07	
	Double Eagle	35,449.24	6,445.32	41,894.56		41,894.56	
	East Sports	886.23	161.13	1,047.36		1,047.36	
	EZ Sporting Goods	1,853.03	336.91	2,189.94		2,189.94	
	Golf Hub	805.66	146.48	952.15		952.15	
	Golf Pro	5,397.95	981.45	6,379.40		6,379.40	

> **NOTE:** The Var % column updated itself to 30% from 10% because of the additional 20% added to the allocation. The Full Budget column also updated itself with the new total from the Baseline and Revisions columns.

Form Templates

Form templates can be setup to allow manual data entry. The entries can be done in a Cube View from an Excel file or from the Spreadsheet feature (OneStream Windows Application only). Each form template group has an assigned Cube View, dashboard or Excel file. Forms can also be loaded using an Excel or CSV template. See Loading Form Data for more information.

Applying Literal Value Parameters to Form Templates

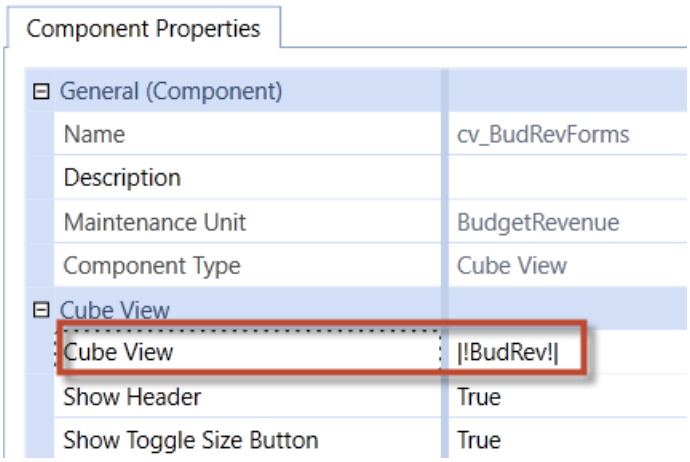
A Delimited List Parameter containing several Cube View or spreadsheet names can be applied to a dashboard to use the dashboard in multiple Form Templates via the Name Value Pairs Form Template Property. Instead of creating multiple dashboards to assign to multiple Form Templates, users can define the Parameter name thus defining which Cube View or spreadsheet the specific form should use. This approach helps in achieving a common toolbar and look for all data entry forms.

The example below uses Cube Views, however, if the forms are driven from spreadsheets, a Spreadsheet Dashboard Component can also be used.

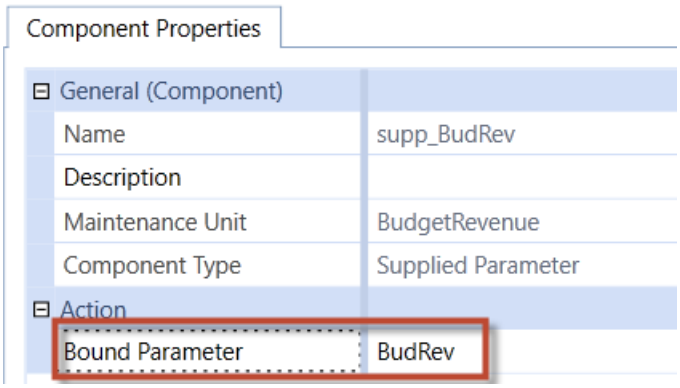
1. Design the Cube Views necessary for data entry. After the Cube Views are complete, create a Dashboard Maintenance Unit.
2. Within the Dashboard Maintenance Unit, create a Delimited List Dashboard Parameter specifying all Cube View names in both the Value Items Property.

General (Parameter)	
Name	BudRev
Description	Budget Revenue Input Forms NA Course Mgmt
User Prompt	
Maintenance Unit	BudgetRevenue
Sort Order	0
Data Source	
Parameter Type	Delimited List
Default Value	
Display Items (comma delimited)	Design NA Revenue (1 Month View),Event Coordination NA Revenue (1 Month View),Services NA Revenue (1 Month View)
Value Items (comma delimited)	Design NA Revenue (1 Month View),Event Coordination NA Revenue (1 Month View),Services NA Revenue (1 Month View)

3. Create a Cube View Dashboard Component and enter the Parameter name in Cube View Property enclosed in Pipes and Exclamation Marks.



4. Create a Supplied Parameter Dashboard Component to pass the Parameter value from the Dashboard to the Form Template. Specify the Parameter Name in the Bound Parameter property.



5. Create a Dashboard with a Uniform Layout Type and assign the Cube View and Supplied Parameter Components to it.
6. Create a Form Template and set the Form Type to Dashboard and assign the desired Dashboard.
7. Define which Cube View this specific Form should use in the Name Value Pairs property by hardcoding a specific Cube View name from the Delimited List Parameter.

   When the Form Template is used in the Workflow, the specified Cube View will display for data entry.

Form Template Profiles

After the form template groups are created, they are organized into form template profiles. The profiles are then assigned to workflow profiles.

- Click  to create a new profile.
- Click  to assign a group to a profile. This allows the user to select which groups will be in the profile.

- Assign a profile to a workflow profile by selecting **Application** > **Workflow Profiles | Form Settings**, or Journal Settings for journal template profiles.

> **NOTE:** Click  to access the Security screen to modify users and groups before assigning them to a form template profile. To assign groups, click  and begin typing the group name. Groups are filtered, so you can select the group, press CTRL and then double-click. This specifies the correct name into the appropriate field.

Security

Access Group
Members of this group have access to the Form Template group.

Maintenance Group
Members of this group have the authority to maintain the Form Template group.

> **NOTE:** Click  to navigate to the Security page so you can modify users or groups before assigning them. Click  and begin typing the name of the Security group in the blank field. As the first few letters are typed, groups are filtered, making it easier to find the right group. Select the group, click CTRL and double-click.

Form Template Properties

Form details can also be loaded using an Excel or CSV template. See Loading Form Data in Data Collection Guides for more information.

General

Name
Name of the form template.

Description
This allows for a more descriptive definition of the form template.

Form Type

Cube View
Select this to utilize a cube view for the form's data entry method.

Dashboard
Select this to utilize a dashboard for the form's data entry method.

Spreadsheet (OneStream Windows App Only)
Select this for the Form to be visible using the Spreadsheet feature in OneStream Windows App. When selected, it is the same functionality as attaching an Excel file, but the spreadsheet only exists within OneStream.

Cube View/Dashboard

Select the Cube View or dashboard that will be associated with this form template. Click  and begin typing the name of the cube view or dashboard in the blank field. Names are filtered making it easier to select items. If the name is unknown, click Cube View or Dashboard Group and scroll through the list. Select a Cube View or dashboard, click CTRL and then double-click.

Spreadsheet (OneStream Windows Application Only)

This spreadsheet should have been built with an embedded Cube View or retrieve functions in order to interact with form data. Upload a pre-created Excel file for data entry when this option is selected. Click  to upload an Excel file. Once selected, the file name in the Excel File space will appear.

- Click  to delete an uploaded Excel file.
- Click  to download a copy of the uploaded Excel file.
- Click  to open the uploaded Excel file.

Excel File (Optional)

An Excel file may be used for data entry. Click  to upload an Excel file. Once selected, the file name in the Excel File space will appear. You can also:

- Click  to delete an uploaded Excel file.
- Click  to download a copy of the uploaded Excel file.
- Click  to open the uploaded Excel file.

Workflow

Form Requirement Level

Not Used
This setting is used if the form is no longer in use and shows the form as Deprecated in the Workflow.

Optional
This setting will allow the user to enter data via the form if desired.

Required
This setting will make the form mandatory for any process to which it is assigned

Form Frequency

All Time Periods
This allows the form to display every period

Monthly
This allows the form to display every month. If this is for a weekly application, they will display the last week of each month.

Quarterly
This allows the form to display every quarter, or four times a year.

Half Yearly
This allows the form to display two times a year; once in June and December.

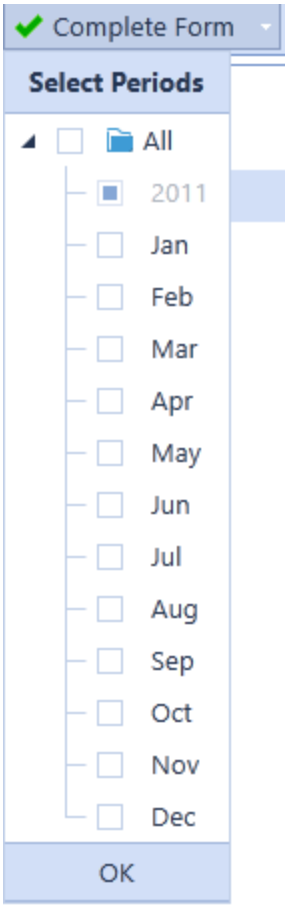
Yearly
This allows the form to display once a year in December.

Member Filter
This turns on the Frequency Member Filter. Filters can then be defined in that section.

Frequency Member Filter
This only becomes available when Member Filter is chosen in the Forms Frequency options above, otherwise this is gray for all the others. The purpose of this option is to allow the ability to filter by time.

Time Filter for Complete Form

The Time filter will dictate how often the form is required. The example below is what will be shown when selecting Complete Form using the filter T#WFYear.Base.



Literal Parameter Values

If the dashboard selected for the form, contains a parameter for the cube view or Spreadsheet name, enter the name/value pair which specifies which cube view the form will use in the Workflow.

Example: ParameterName = CubeViewName

Parameters (Cube View Forms Only)

Parameter Type (for Parameters 1 through 6)
Parameter types allow for different values or variables to be passed to the Cube View.

Literal Value
The value you specify will be hard coded.

Input Value
This allows the user to enter or change the value.

Delimited List
This provides a distinct list of values populated in the Parameter Type.

Member List
This will produce a flat list of Members to show to the user.

Member Dialog
Similar to the Member list, this allows the user to select a member, but through a pop-up Member Selection dialog which also has search capabilities. This is more appropriate for a dimension such as Accounts or Entities where the user can choose a base or Parent member by traversing a hierarchy.

Variable Name from Cube View
Place a parameter in this field to replace another parameter used in a Cube View. For example, a prompt such as |!MyEntityName!| will suppress the Parameter dialog from appearing when opening the form.

User Prompt
This prompts the user based on the question entered here.

Default Type
Set the default value so it is not blank.

Dimension Type
Choose the dimension being prompted

Member Filter
Allows the ability to put in a defined filter. For example, E#Root.WFProfileEntities.

Form Template Group Properties

This section includes general group properties.

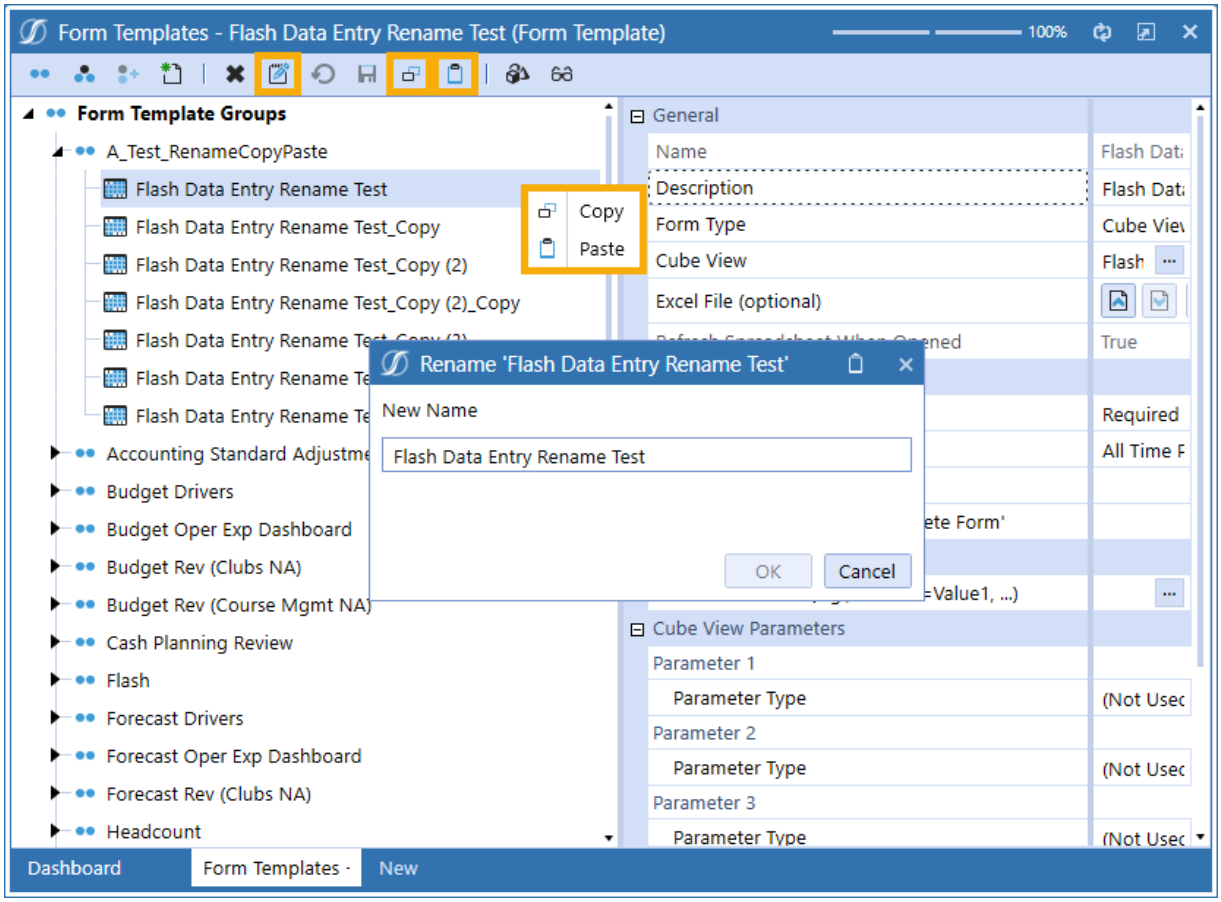
General (Form Template Group)

Name
The name of the form template group.

Description
A short description of the template group such as how or where it is used.

Managing Form Templates

Form Templates can be renamed, copied, and pasted from the toolbar. Form templates can be copy and pasted using a right-click.



Rename a form template by clicking on the toolbar. All associated links for the Workflow Profile and Dashboard Groups will remain valid as the form template is based on a unique identifier, not the name.

Copy a form template by clicking on the toolbar or right-clicking the form template name and selecting Copy.

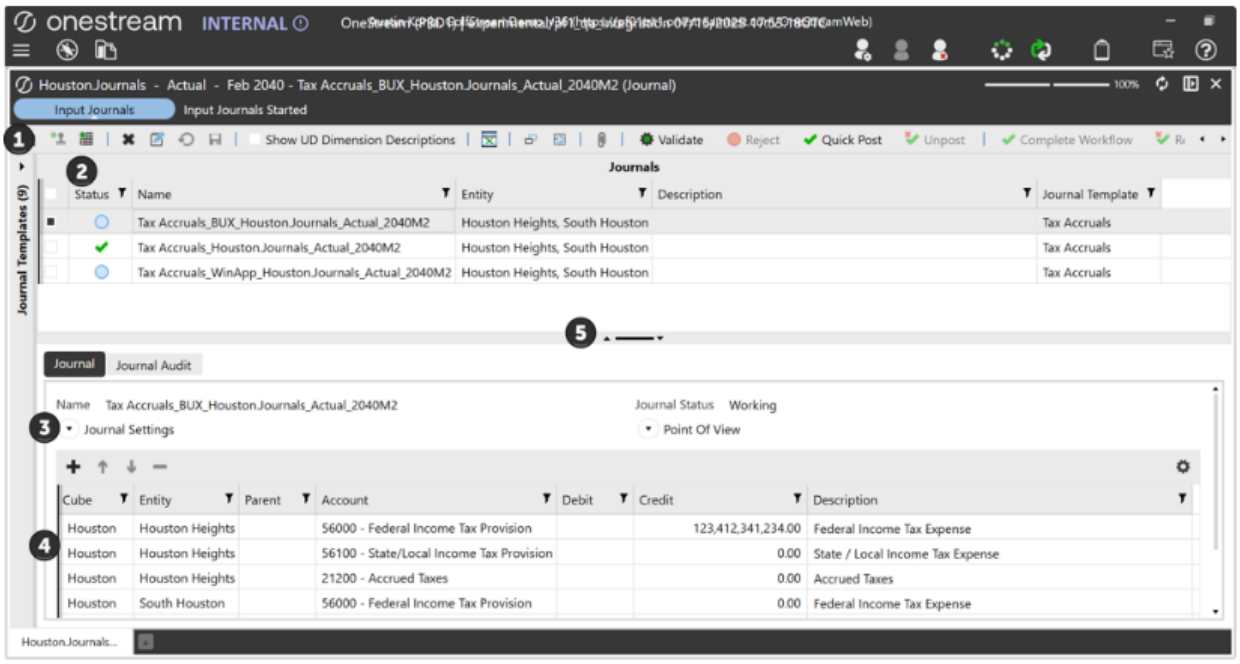
Paste a form template by clicking on the toolbar or right-clicking the form template name and selecting Paste.

Journals Overview

Journals are used to record business transactions in the accounting records of a business from an enterprise resource planning perspective. Journals are typically used in financial consolidation projects to record closing adjustments. Journals display account amounts and indicate whether the accounts are balanced. In the OneStream app, journals are written to the AdjInput member of the origin dimension.

You can open or create journals from the OnePlace Workflow tab. Select a Workflow Profile, Scenario, Time Period, and then click the journal you want to open.

The following image is an example of the Input Journals workflow interface.



1. The Journal Templates pane is collapsible.
2. The Journals Grid is a listing of available journals. It is located above the Journal Header/Journal Details area.
3. Entities are located at the top of the Journals Grid. You can filter the grid by entity.
4. Journal Header properties are displayed in collapsible panes.
5. You can drag the horizontal divider up or down, or snap it to a location using the arrow buttons.

Create a New Journal

There are three ways to create a journal as shown in the image below.



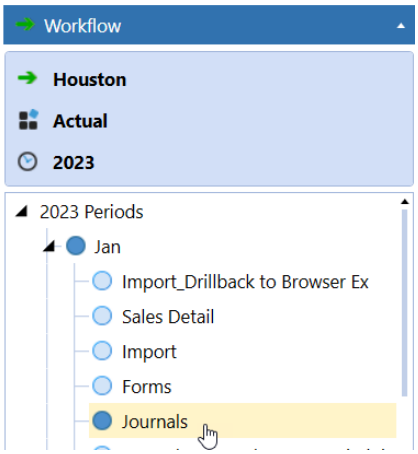
1. **Create Journal**: Create a new blank journal.
2. **Create Journal Using Selected Template**: Create a new journal based on the selected template.
3. **Create Journal Using File**: Create a new journal using data from either an Excel file or a CSV file.

In the following sections, you will create journals using the three methods listed above.

Create Journal

To display the Journals toolbar and create a new blank journal:

1. In the OneStream app, click the **OnePlace** tab.
2. Click **Workflow** to expand.
3. Click the existing **Workflow Profile** to change it if necessary.
    a. In the **Select Workflow View** dialog, click the **Workflow Profile** tab and make a selection.
    b. Click the **Scenario** tab and make a selection.
    c. Click the **Year** tab and make a selection.
    d. Click **OK**.
4. Expand the period tree, locate the time period, and click **Journals**.



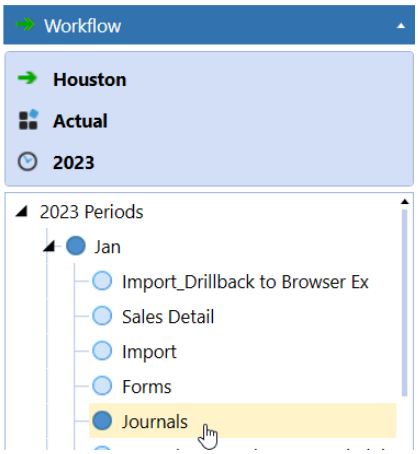
5. On the Journals toolbar, click **Create Journal**.
6. In the Journal Properties editor, under **General**, type a name for the new journal.
7. Under **Journal**, set properties as needed.
8. Under **Point of View**, set properties as needed.

9. Click **Save**. The new journal is added to the **Journals** pane to the right of the journal properties section.

Create Journal Using Selected Template

Templates can be built for journals to accelerate repetitive journal creation. They allow for control over which properties are configured by the user. They can be used to create new journals.

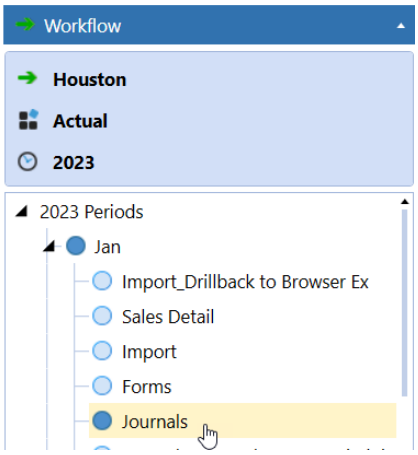
1. In the OneStream app, click the **OnePlace** tab.
2. Click **Workflow** to expand.
3. Click the existing **Workflow Profile** to change it if necessary.
    a. In the **Select Workflow View** dialog, click the **Workflow Profile** tab and make a selection.
    b. Click the **Scenario** tab and make a selection.
    c. Click the **Year** tab and make a selection.
    d. Click **OK**.
4. Expand the period tree, locate the time period, and click **Journals**.



5. In the **Workflow Journal Templates** pane, click a journal template to select. This will be the journal on which you will base the template.
6. On the Journals toolbar, click **Create Journal Using Selected Template**. It will automatically be given a name but you can change this if necessary.
7. Under **Journal Template**, set properties as needed.
8. Under **Journal**, set properties as needed.
9. Under **Point of View**, set properties as needed.
10. Click **Save**. The new journal is added to the **Journals** pane to the right of the journal properties section.

Create Journal Using File

1. In the OneStream app, click the **OnePlace** tab.
2. Click **Workflow** to expand.
3. Click the existing **Workflow Profile** to change it if necessary.
    a. In the **Select Workflow View** dialog, click the **Workflow Profile** tab and make a selection.
    b. Click the **Scenario** tab and make a selection.
    c. Click the **Year** tab and make a selection.
    d. Click **OK**.
4. Expand the period tree, locate the time period, and click **Journals**.



5. On the Journals toolbar, click **Create Journal Using Excel (xlsx) or Comma Separated Values (csv) File**.
6. In the **Open** dialog, browse to the file and click **Open**.
7. Under **Journal Template**, set properties as needed.
8. Under **Journal**, set properties as needed.
9. Under **Point of View**, set properties as needed.
10. Click **Save**. The new journal is added to the **Journals** pane to the right of the journal properties section.

Open a Journal

You can open any journals you have created from the **Journals** pane in the OneStream app.

1. In the OneStream app, click the **OnePlace** tab.
2. Click **Workflow** to expand.
3. Click the existing **Workflow Profile** to change it if necessary.

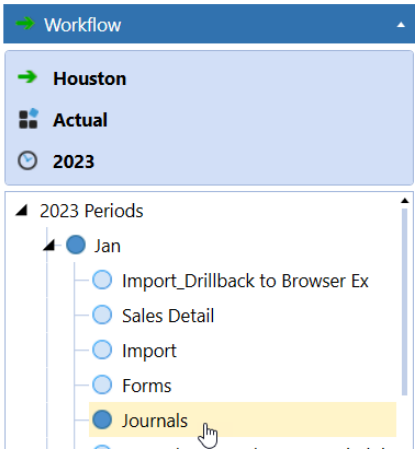
a. In the **Select Workflow View** dialog, click the **Workflow Profile** tab and make a selection.

b. Click the **Scenario** tab and make a selection.

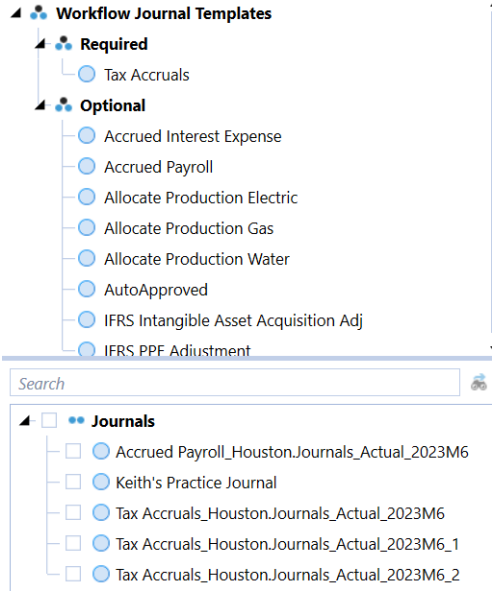
c. Click the **Year** tab and make a selection.

d. Click **OK**.

4. Expand the period tree, locate the time period, and click **Journals**.



The **Workflow Journal Templates** pane and the **Journals** pane display.

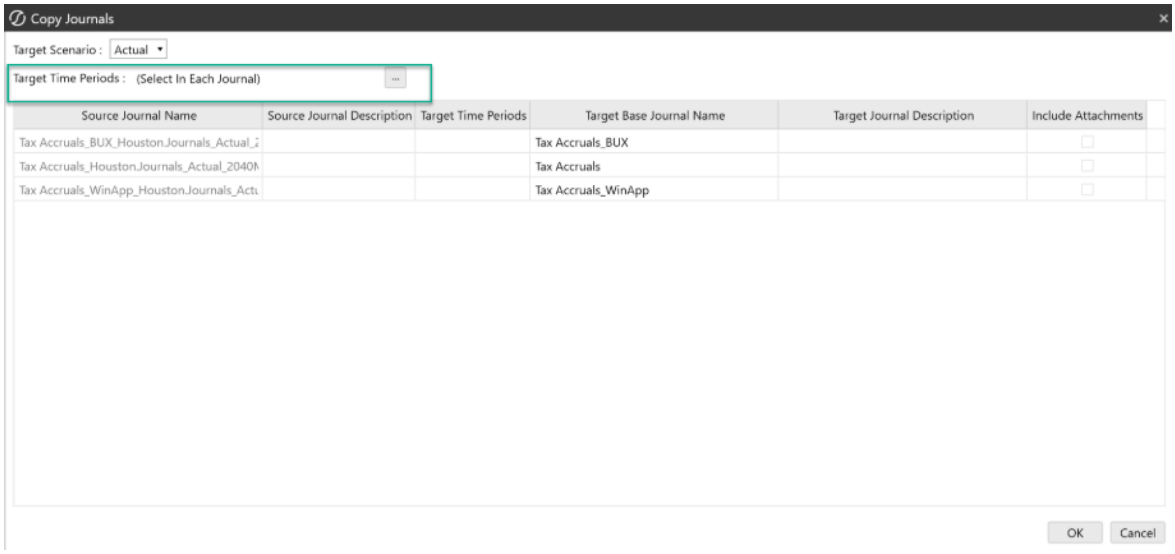


5. In the **Journals** pane, click the journal you would like to view. You can new set journal properties as needed and enter line items into the journal.

Bulk Selection of Time Periods for Copying

You can select time periods for all journals in a copy job simultaneously. This streamlines the journal copy process, and avoids having to select time periods individually for each journal.

1. Navigate to the **Input Journals** page.
2. Select the journals you want to copy.
3. Click **Copy Journals**.
4. At the top of the **Copy Dialog** box, select the appropriate target time periods.



5. Click **OK**.

Rename a Journal

Journals that are in Working or Rejected status can be renamed by administrators or members of the workflow profile Journal Process Group security role.

Follow the steps below to rename a journal:

1. Select a journal that is in Working or Rejected status. If you select a journal of another status, or more than one journal, the Rename Selected Journal button is disabled.
2. Click **Rename Selected Journal** in the toolbar .
3. In the **Rename Journal** dialog box, type a unique journal name.
4. Click **Save**. The journal name is updated. If you type a name that already exists, you will be prompted to type a unique name.

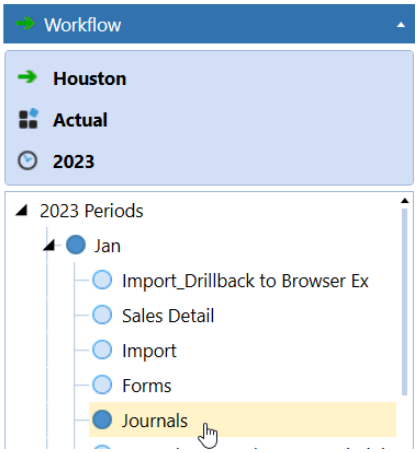
> **NOTE:** The system will not append any additional workflow profile information to the new journal name.

Delete a Journal

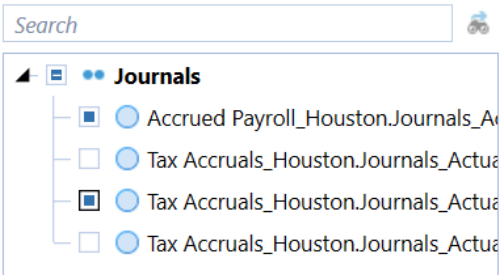
You can delete journals that you no longer use or need.

1. Navigate to the journal you would like to delete:
    a. In the OneStream app, click the **OnePlace** tab.
    b. Click **Workflow** to expand.
    c. Click the existing **Workflow Profile** to change it if necessary.
        i. In the **Select Workflow View** dialog, click the **Workflow Profile** tab and make a selection.
        ii. Click the **Scenario** tab and make a selection.
        iii. Click the **Year** tab and make a selection.
        iv. Click **OK**.

d. Expand the period tree, locate the time period, and click **Journals**.



2. In the **Journals** pane, place a tick into the box next to the journal you want to delete. You can delete more than one journal at a time by ticking all of the necessary boxes.



3. Click **Delete Selected Item**.
4. In the confirmation dialog, click **Yes**.

Loading Data into Journals

There are several ways to load data into a journal, including the following:

- Journal Excel template
- Journal CSV template
- OneStream Workflow process
- Extensibility Business Rules (BRApi)

Loading Journal Data

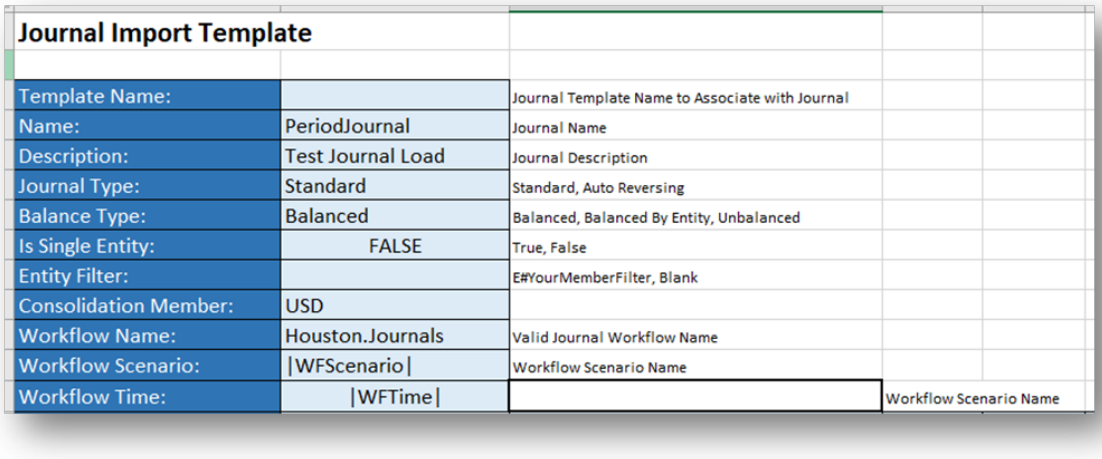
Journal Excel Template

When loading journal entries via Excel a specific template must be created to determine the journal properties, the Dimensions to which the entries are being made, and the debit and credit amounts.

OneStream reads this template using a specific Named Range which is explained later in this section. Ensure the following information is included in the Named Range.

Property Tokens

The first eleven rows of the Named Range in the Excel template must include the following token definitions:



Journal Import Template			
Template Name:		Journal Template Name to Associate with Journal	
Name:	PeriodJournal	Journal Name	
Description:	Test Journal Load	Journal Description	
Journal Type:	Standard	Standard, Auto Reversing	
Balance Type:	Balanced	Balanced, Balanced By Entity, Unbalanced	
Is Single Entity:	FALSE	True, False	
Entity Filter:		E#YourMemberFilter, Blank	
Consolidation Member:	USD		
Workflow Name:	Houston.Journals	Valid Journal Workflow Name	
Workflow Scenario:	|WFScenario|	Workflow Scenario Name	
Workflow Time:	|WFTime|		Workflow Scenario Name

Template Name

Enter a template name is applicable. If this is a free form journal, leave this blank.

Name

Enter the name of the journal.

Description

If desired, enter a description for the journal.

Journal Type*

Standard or Auto-reversing. Allocation is not supported for Excel or CSV import.

Balance Type*

Balanced, Balanced by Entity, or Unbalanced

Is Single Entity*

True or False

Entity Filter*

Use a Member Filter to specify the Entities used with this journal.

Consolidation Member

Enter the specific currency or Local Member of Consolidation.

Workflow Name

Enter the Journal Workflow name. For example, if the name of the Workflow Profile is Houston, and the Adj input type is named Journals, enter Houston.Journals.

Workflow Scenario

Enter the Workflow Scenario or make the template dynamic by entering the |WFScenario| Substitution Variable.

Workflow Time

Enter the Workflow Time or make the template dynamic by entering the |WFTime| Substitution Variable. Workflow Time support two fields. The available cell immediately to the right is option representing the CubeTimeName. This field can be used when the Scenario's Workflow Tracking Frequency is Yearly, and the Input Frequency is Monthly. For example, the Workflow Time would be |WFTime| or 2019 and the CubeTimeName would be the period to post, 2019M7.

Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	|WFTime|	

When Tracking Frequency is Yearly and Input Frequency is Monthly:

Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	2019M7	

Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	|WFTime|	2019M7

Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	|WFTime|	|WFYear|M7

*See Journal Templates Data Collection for details on these Journal properties.

Dimension Tokens

Next, create the Dimension Tokens necessary to load the journal to the correct Dimensions in OneStream. The Dimension tokens need to be the column header for each data row. The standard tokens used determine the Cube, Entity, Parent, Account, Flow, Intercompany, the User Defined Members, and a Label if needed. Refer to Loading Stage Data for the syntax. The journal specific tokens are as follows:

AMTDR#

This indicates the debited amount.

AMTCR#

This indicates the credited amount.

Header Abbreviations

Static Value

Use :[] to fix a specific Member to the entire column creating a Static Value for the specified Source Dimension. For example, F#:[None] imports the None Flow Member for every Flow row within the Named Range. This syntax applies to all Dimension Tokens.

After the Dimension Tokens are setup, enter the data in the corresponding column.

Template Example:

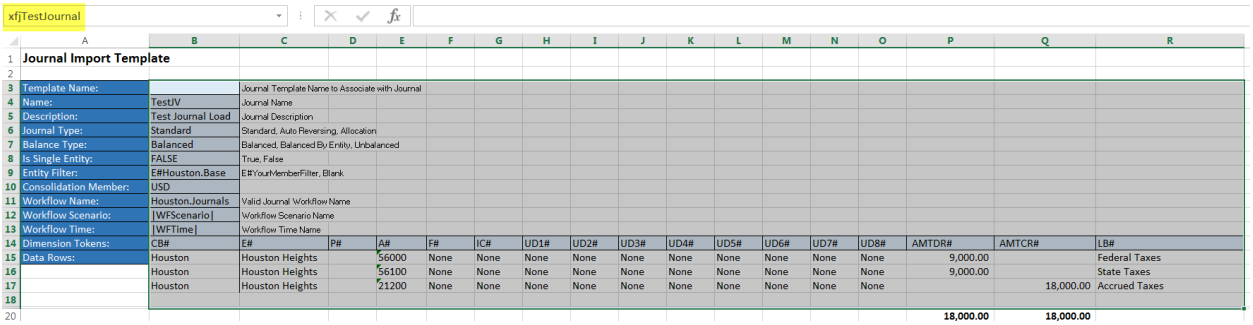
Journal Import Template

Template Name:		Journal Template Name to Associate with Journal														
Name:	TestJV	Journal Name														
Description:	Test Journal Load	Journal Description														
Journal Type:	Standard	Standard, Auto Reversing, Allocation														
Balance Type:	Balanced	Balanced, Balanced By Entity, Unbalanced														
Is Single Entity:	FALSE	True, False														
Entity Filter:	E#Houston.Base	E#YourMemberFilter, Blank														
Consolidation Member:	USD															
Workflow Name:	Houston.Journals	Valid Journal Workflow Name														
Workflow Scenario:	|WFScenario|	Workflow Scenario Name														
Workflow Time:	|WFTime|	Workflow Time Name														
Dimension Tokens:	CB#	E#	P#	A#	F#	IC#	UD1#	UD2#	UD3#	UD4#	UD5#	UD6#	UD7#	UD8#	AMTDR#	AMTCR#	LB#
Data Rows:	Houston	Houston Heights		56000	None	None	None	None	None	None	None	None	None	None	9,000.00		Federal Taxes
	Houston	Houston Heights		56100	None	None	None	None	None	None	None	None	None	None	9,000.00		State Taxes
	Houston	Houston Heights		21200	None	None	None	None	None	None	None	None	None	None		18,000.00	Accrued Taxes
															18,000.00	**18,000.00**	

The final step is to create a XFJ Named Range making sure to include the definition of each property, the Dimension tokens, and the data rows. The Named Range must begin with XFJ for OneStream to read and load it correctly. Multiple XFJ Named Ranges can be used within the template over multiple tabs.

> **NOTE:** Loading of Journal Templates or previously exported Journal data only requires a Parent (P#) column value to be populated if the target Consolidation dimension member being updated is OwnerPreAdj or OwnerPostAdj. Otherwise, this entry can be left blank.

Named Range Example:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	**Journal Import Template**																	
2																		
3	Template Name:		Journal Template Name to Associate with Journal															
4	Name:	TestJV	Journal Name															
5	Description:	Test Journal Load	Journal Description															
6	Journal Type:	Standard	Standard, Auto Reversing, Allocation															
7	Balance Type:	Balanced	Balanced, Balanced By Entity, Unbalanced															
8	Is Single Entity:	FALSE	True, False															
9	Entity Filter:	E#Houston.Base	E#YourMemberFilter, Blank															
10	Consolidation Member:	USD																
11	Workflow Name:	Houston.Journals	Valid Journal Workflow Name															
12	Workflow Scenario:	|WFScenario|	Workflow Scenario Name															
13	Workflow Time:	|WFTime|	Workflow Time Name															
14	Dimension Tokens:	CB#	E#	P#	A#	F#	IC#	UD1#	UD2#	UD3#	UD4#	UD5#	UD6#	UD7#	UD8#	AMTDR#	AMTCR#	LB#
15	Data Rows:	Houston	Houston Heights		56000	None	None	None	None	None	None	None	None	None	None	9,000.00		Federal Taxes
16		Houston	Houston Heights		56100	None	None	None	None	None	None	None	None	None	None	9,000.00		State Taxes
17		Houston	Houston Heights		21200	None	None	None	None	None	None	None	None	None	None		18,000.00	Accrued Taxes
18																		
20																18,000.00	18,000.00	

NOTE: The Named Range only covers each property definition.

Journal CSV Template

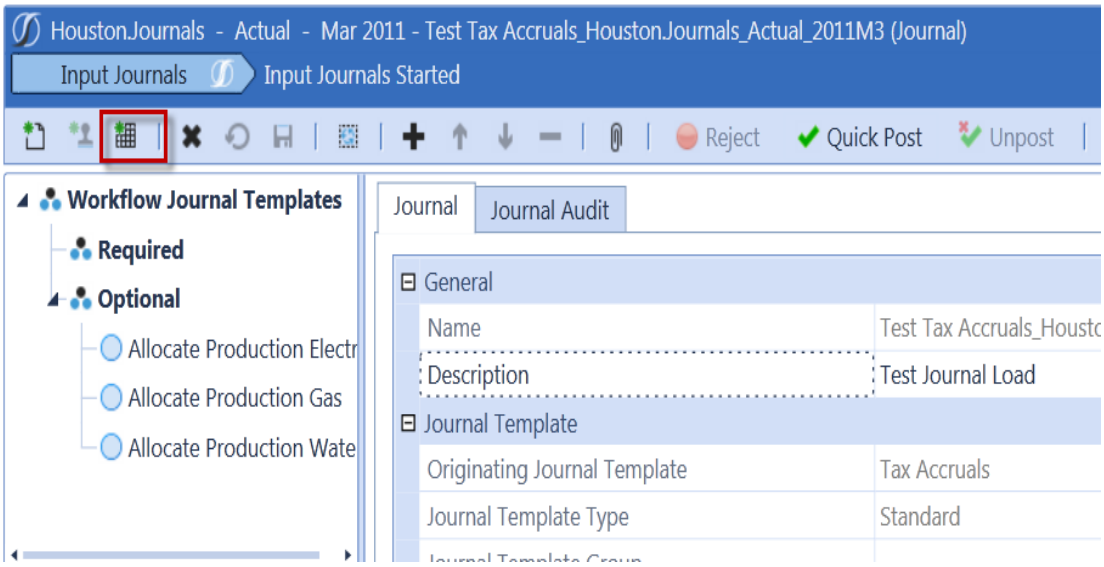
To set up a CSV template for a Journal, the Header and Detail values must be specified.

	A	B	C	D	E	F
2	!RowType (H=Header)	JournalName	OriginatingTemplateName	JournalDescription	JournalType	JournalBalanceType
3	!RowType (D=Detail)	JournalName	CubeName	EntityName	ParentName	AccountName
	H	Test Tax Accruals_Houston.Journals_Actual_2011M3	Tax Accruals	Test Journal Load	Standard	Balanced
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3	Houston	Houston Heights		56000
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3	Houston	Houston Heights		56100
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3	Houston	Houston Heights		21200
4	H	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Tax Accruals		Standard	Balanced
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	Houston Heights		56000
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	Houston Heights		56100
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	Houston Heights		21200
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	South Houston		56000
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	South Houston		56100
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	South Houston		21200

1. Column A Specifies Row Type
   In the first two rows of Column A, create two Row Type Parameters specifying the Header and the Detail. In the example above, !RowType (H=Header) and !RowType (D=Detail) are used to tag the corresponding rows with H or D identifying where the Header and Detail information is located in the CSV file.
2. Row One Specifies the Headers
   After the Header Parameter is configured, enter the Journal Column Headers. The required Journal Headers are JournalName, OrigniatingTemplateName, JournalDescription,JournalType, JournalBalanceType, IsSingleEntity, EntityMemberFilter, ConsName, WFProfileName, WFScenarioName, WFTimeName and CubeTimeName. For more details on these, refer to the Journal Excel Template.
3. Row Two Specifies the Details
   After the Detail Parameter is configured, enter the Journal Detail Headers. The required Journal Detail Headers are JournalName, CubeName, EntityName, ParentName, AccountName, FlowName, ICName, all UDNames, DebitAmount, CreditAmount, and LineDescription. For more details on these, refer to Journal Excel Template.
4. Header and Detail Tags
   The Journal data is driven by how each row in column A is tagged. Any rows tagged with H load as the Headers and any row tagged with D load as the details.

Loading Journal Details via Workflow

After the template is configured, users can load it directly into OneStream during the Workflow process. While the Journal Input type is selected, click the  icon in the Journal toolbar. This lets you select the desired Excel or CSV template and load it into OneStream. After the file is loaded, the journal line items will appear in the journal and the user can save it to the Cube.



Extracting/Loading Journal Details via BRApi

Users can export journal details into a CSV or XSLX file and load journals from a CSV or XSLX file by configuring an Extensibility Business Rule. This allows journal details to be extracted from one application and loaded into another. Users can also extract the journal, make changes, and re-load.

Extract Journals to CSV

Use BRApi.Journals.Data.ExportJournalstoCSV and define the session, filepath, Workflow Profile, Scenario, Time Filter, and Journal Status.

BRApi.Journals.Data.ExportJournalsToCsv(si, filePath, "Houston", "Actual", "T#|WFYear|.Base", "Posted")

Extract Journals to XSLX

Export journals associated with defined parameters to an XSLX file using the ExportJournalsToXSLX BRApi. It is sometimes easier to read and modify journal data in an XSLX format as compared to CSV files that are generated by the ExportJournalsToCSV BRApi. Use the parameters listed in the following table.

Parameter	Description
si	Session information
serverFilePath	Name and location to save the file
wfProfileName	Name of the Workflow Profile
wfScenarioName	Name of the Scenario
wfTimeMemberFilter	Filters workflow profile times. For example, if the scenario's Workflow Tracking Frequency is set to Quarterly, and Input Frequency is set to Monthly, the filter searches for journals associated with the Workflow Tracking Frequency. Time must be prefixed with T#. You can type a comma separated list of times such as T#2022H1, T#2022H2.
journalName	Filters journals that contain the specified text
journalStatus	Status of the journal. You can search by status or use All to return all statuses. You cannot type a comma separated list.

The following code example uses the ExportJournalsToXslx BRApi:

```
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.LInq
```

```
Imports Microsoft.VisualBasic
Imports OneStream.Finance.Database
Imports OneStream.Finance.Engine
Imports OneStream.Shared.Common
Imports OneStream.Shared.Database
Imports OneStream.Shared.Engine
Imports OneStream.Shared.vcf
Imports OneStream.Stage.Database
Imports OneStream.Stage.Engine

Namespace OneStream.BusinessRule.Extender.ExportJournalstoXSLX
    Public Class MainClass
        Public Function Main(ByVal s1 As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Object, ByVal args as ExtenderArgs) As Object
            Try
                Dim configSettings As AppServerConfigSettings = AppServerConfig.GetSettings
(si)
                Dim sourceFolder As String = BRApi.Utilities.GetFileShareFolder
(si,FileShareFolderTypes.FileShareRoot,Nothing) & "\System\Contents\Export"
                Dim journalFilePath As String = sourcefolder & "\" & "Journal_Export.xlsx"
                Dim filePath As String = BRApi.Journals.Data.ExportJournalsToXlsx(si,
journalFilePath, "Houston", "Actual", "T#2023M5", "", All)

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

Load Example

Use BRApi.Journals.Data.ImportAndProcessJournals and define the session, filepath, and journal tasks to complete upon loading the journal details.

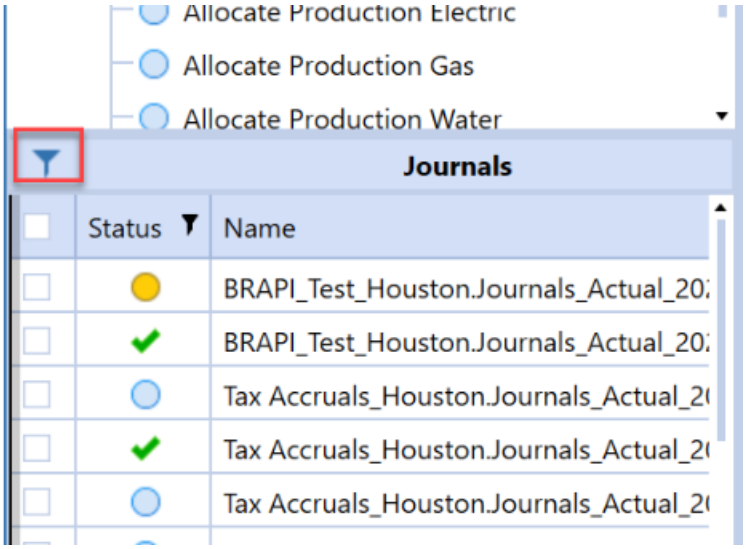
'BRApi.Journals.Data.ImportAndProcessJournals(si, filePath, save, submit, approve, post, unpostAndOverwrite, throwOnError)

Filter Journals

You can filter journal entries on the entity name and columns, and you can combine multiple filters to refine the list of journals according to your needs. For example, you can filter journals by entity and journal type to see only those journals that match the filter criteria. Filtering helps auditors, accountants, and other stakeholders, quickly identify specific transactions and reduces the time spent on searching for information, especially when investigating entries.

To filter journals by entity:

1. On the Journals list toolbar, click the **Filter** button at the top left corner.



2. In the filter dialog box, type all or part of the entity name in the entity field. This filter is a “Contains Text” filter, meaning it will match any entity name that contains the entered text.
3. Click **Apply**. The journal list displays only those journals that contain the specified filter string in their entity name.

To filter journals by column:

1. Click the **Filter** icon in the column header that you want to filter.
2. In the **Filter** dialog box, choose between the following two options:
    - In the upper part of the dialog box, select one or more items to filter by distinct values found in the respective column.
    - In the lower part of the dialog box, set your own filtering criteria.
3. Click the **Filter** icon to apply the filter.

To clear filters:

1. Click on the filter icon that you want to clear.
2. Click the **Clear Filter** button.

Process a Journal

After you have created or opened a journal and entered data into the ledger, there are several actions you can perform on the journal.



1. **Validate a journal**: Ensure that data is correct and that cube intersections are valid.
2. **Reject a journal**: Discard the journal data.
3. **Quick post a journal**: Apply the journal data to the associated cube.
4. **Unpost**: Remove the journal data from the associated cube.

In the following sections, you will move a journal through the above processes.

Validate a Journal

You can ensure that dimension members selected in the journal are valid cube intersections based on constraints configured in the dimension library. If validation errors exist, they will display in a Validation pane so that you can view them and make edits to the journal. This helps you save time by validating entries earlier in the process.

> **NOTE:** You must have the Journal Processing security role to use this feature and dimension member constraints must first be configured in the dimension library.

To validate a journal:

1. Open a Journal that you would like to validate.
2. Click **Validate**.

Journal data will be evaluated and if invalid data is found, the **Validation** pane will display a message that details each entry error.

Reject a Journal

You can reject a journal after it has been submitted for approval.

To reject a journal:

1. Open a Journal that you would like to reject.
2. Click **Reject**.
3. In the **Reject Journals** prompt, type an optional reason or comment.
4. Click one of the following:

a. **Cancel**- the prompt closes and the rejection is canceled.

b. **OK**- the prompt closes, selected journals are rejected, and the comment (if entered) is saved to all journals.

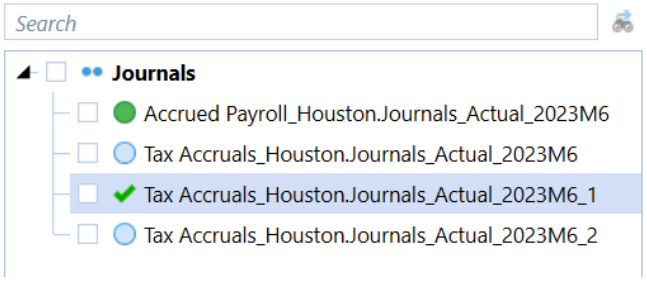
The journal status changes to Rejected. In the **Journal Audit**, rejection comments are displayed in the **Rejected Comment** field. If multiple rejections occur, only the last rejection comment is displayed.

Post a Journal

Posting a journal in the OneStream app means adding the ledger data into the associated cube, similar to moving transaction data into a general ledger.

To post journal data:

1. Open a Journal that you would like to post.
2. Enter line item journal data as necessary.
3. Optional. If you would like to post more than one journal at a time, select the journals in the **Journals** pane by placing a tick in the selection box next to the journals.
4. Click **Quick Post**. A green checkmark next to the journal in the **Journals** pane indicates the journal has been posted.



Journal data is moved into the associated cube.

Unpost a Journal

Unposting a journal in the OneStream app means removing the ledger data from the associated cube.

To unpost journal data:

1. Open a Journal that you would like to unpost.
2. Optional. If you would like to unpost more than one journal at a time, select the journals in the **Journals** pane by placing a tick in the selection box next to the journals.
3. Click **Unpost**. The green checkmark next to the journal in the **Journals** pane is removed, indicating the journal has been unposted.

Journal data is removed from the associated cube.

Central Workflow Processing

Parent adjustment workflows can post journals to entities that belong to already-certified base workflows. When a central journal input post occurs to an entity owned by a certified base workflow, the system will:

- Temporarily un-certifiy the impacted base workflow (and its ancestors) to In Process status
- Allow the journal to post to the appropriate adjustment workflow unit (including Level 2 and Level 3 channel-based units)
- Attempt to re-certify impacted workflows after posting

All certification changes are logged in the workflow audit trail.

> **NOTE:** The parent workflow must be a journal input workflow, and the base workflow must be a central input journal.

Central workflow processing is controlled by a new property in the controlling parent adjustment workflow profile called, **Perform Central Workflow Processing**, and is located in the **Data Quality** section of the **Adj** property grid. It supports both tree-structured workflows and named dependent workflows that are assigned to review levels.

Central Workflow Processing Usage

The following procedure walks you through central journal input setup.

1. Enable central workflow processing on the controlling parent workflow:
    a. Open the controlling parent adjustment workflow profile.
    b. In the **Integration Settings** section, locate **Perform Central Workflow Processing** and set to **True**. This applies to scenarios where the parent workflow should post entries assigned to certified base workflows.



    c. If you deploy via XML, make sure that XML and metadata templates are updated.
2. Configure central journal input:
    a. On the base input adjustment workflow where journals are received centrally, make sure **Central Journal Input** is assigned.
3. Run the base workflow as usual.

    Base entities complete their required tasks (Import, Forms, and Adjustments). Base workflows complete (Lock, Confirm, and Certify).
4. Post a parent adjustment using central journal input:

a. From the parent journal input workflow, Create a New Journal and Post a Journal a journal to an entity whose base workflow is already certified. When you are posting, the system will:

   - Identify the impacted base workflow and ancestors (including named dependent workflows assigned to review levels)
   - Set their certification status to **In Process**, which removes the implicit certification restriction for the impacted units
   - Post the journal to the appropriate workflow unit and data intersection

Re-Certification Behavior

After posting, the system attempts to re-certify the impacted workflows. Certification changes are written to the Workflow Audit Log, which include quick certify actions and certification profile question-based certification actions

Re-certification is only attempted for steps that can complete quickly. Where re-certification would require longer-running tasks, such as workspace or process tasks (including cube processing that would take an extended amount of time), the journal remains posted, but the impacted base workflows are left un-certified. The system reports which workflows could not be re-certified, and you must complete those tasks and re-certify those workflows manually.

For parent adjustments that reset the process task on child workflows, process tasks must be completed on the child workflows before re-certification can be completed.

Security and Access

Security ensures that when you are posting journals and triggering overrides, you have the proper rights to post journals to the parent journal input and to un-certify and re-certify workflows.

All actions respect entity, scenario, data cell access, and workflow security by managing data application permissions.

To ensure security and control, only authorized users can trigger overrides. A locked workflow remains protected, and posting to it fails with a clear message. Certification alone does not block posting under this feature. When a base workflow is certified but not locked, the journal posts and re-certification is attempted afterward. If re-certification cannot be completed automatically, posting still succeeds and the system reports the workflows that require manual re-certification.

Benefits of Central Journal Input

Central journal input adjustments can be made after base workflows are certified, without forcing administrators to manually un-certify and re-certify entire workflow trees. Additionally:

- Audit risk in Actuals is reduced by maintaining full audit trails of certification status changes, quick certify and certification profile question actions, and journal postings that impact certified workflows.
- Base workflow responsibility and timing is preserved. Base entities can complete, lock, and certify while allowing corporate-level adjustments on frozen data.
- Complex workflow designs are supported including multi-period processing, named dependent workflows, level 2 and 3 workflow units
- Manual administration work and errors are minimized by automating the temporary un-certification process of impacted workflows, and by automating attempted re-certification after corporate posting.

Recurring Journals

A recurring journal is a journal entry that repeats at regular intervals (monthly or annually) for routine transactions. It is commonly used for expenses that repeat over time on a consistent basis. Recurring journal entries can be automated in accounting systems to:

- maintain consistency of financial statements
- ensure accurate recording of routine transactions
- reduce effort while minimizing the risk of errors

Create a Standard Recurring Journal

There is a new journal type called Standard Recurring. This type lets you define the frequency of automatic journal creation.

1. Create Journal.
2. For journal type, select **Standard Recurring**.
3. For **Recurring Journal Frequency**, select one of the following:
    - All Time Periods
    - Monthly
    - Quarterly
    - Half Yearly
    - Yearly

    > **NOTE:** The journal cannot recur more frequently than input periods allow. For example, if you are using quarterly input periods, you cannot have a monthly journal recurrence as Q1 would contain three months.

4. Post a Journal. A new journal is created in the specified time period, and is a copy of the previous journal with the same naming scheme, settings, POV, and line items.

    > **NOTE:** You must have process rights, as you are the creator of the recurring

> journal.

You will receive a confirmation notice upon successful creation.

Journal import and export does not handle the **Recurring Journal Frequency** setting. Imports and exports that include standard recurring journals will default to All Time Periods.

Final Recurring Period

The final recurring period controls how the last occurrence of a recurring journal entry is handled. It defines the logic used when a recurring journal series reaches its final period (for example, the final month or final fiscal period of the schedule). This ensures that the series ends correctly, with the correct dates, the number of occurrences, and the amounts applied in the final period.

1. Create or open a recurring journal entry.
2. In **Journal Settings**, specify the **Final Recurring Period** in the **Time Period Selection** dialog box.

> **NOTE:** After this setting is configured, no recurring journals are created after the **Final Recurring Period** setting.

Journal Templates

Create pre-set Journal Templates in Journal Template Groups. After the Journal Template Groups are created, they are organized into Journal Template Profiles and then assigned to Workflow Profiles. For more details on Profiles, see Form Template Profiles.

Journal details can also be loaded via an Excel or CSV template. See Loading Journal Data for more information.

Journal Template Properties

This section includes various journal template properties.

General

Name
Name of the journal template.

Description
A detailed description of the journal template.

Journal Template

Journal Template Type
A journal template can either be standard or auto approved. If auto approved, a user can create a journal from the template with limited editing ability meaning permission to change name, description, and so forth. After pressing Save, the line items are validated, and the journal skips the approval process and becomes approved.

If the journal needs to be edited, it is handled like any other journal from that point forward. Someone in the Approve Journals group must reject it for the original user to edit it. Then, the regular Submit, Approve steps will occur. This approach is needed because the person who created the auto approved template only pre-approved certain numbers to be put in the system.

The user does not need to be in the ApproveJournals security group to do this. They only need to be in the ProcessJournals security group which is normally the group for end users to save or submit journals.

Journal Template Group
This displays the journal template group that was created under the new template.

Journal Requirement Level
This setting works together with Frequency and determines when a journal is required. The options available are Not Used (anymore), Optional, or Required.

Journal Frequency

All Time Periods
This allows the journal to display every period.

Monthly
This allows the journal to display every month. If this is for a weekly application, it will display the last week of each month.

Quarterly
This allows the journal to display every quarter, or four times a year.

Half Yearly
This allows the journal to display two times a year; once in June and December.

Yearly
This allows the journal to display once a year in December.

Member Filter
This turns on the Frequency Member filter and then the filters can be defined in that section. For example, if this journal only needs to be completed in September, use the filter T#WFYearM9.

Frequency Member Filter
This becomes available when Member Filter is chosen in the Forms Frequency options above, otherwise this is gray for all the others. The purpose of this option is to allow the ability to filter by time.

The following settings control what the user can modify when creating a journal. Settings are True or False.

- Can Change Journal Settings
- Can Change Journal POV
- Can change Journal Line Items
- Can change Journal Amounts

Journal

Journal Type
The options are Standard, Auto Reversing, or Allocation. When posting an Auto Reversing journal, the auto reversal journal is automatically created in the next time period and set to the Approved state. The Auto Reversal journal has all the debits and credits reversed.

When un-posting an Auto Reversing journal, check to make sure the auto reversal is not posted first. If not, delete the Auto Reversal journal from the next time period and un-post.

An Allocation journal can be set up to perform simple or more intricate allocations, such as creating the weighting of the allocation, previewing the actual allocation entries, and un-posting them if they need to be run again.

Journal Balance Type

Balanced
The entire set of journal lines must balance between the debits and credits.

Balance by Entity
Debits and credits in a multi-Entity journal must balance for each entity.

Unbalanced
Balance check will not be performed. This is normally used for one-sided journals.

Is Single Entity
If True, the entity name is entered in the Journal POV and all journal lines relate to this one entity. If False, the Cube, Entity and Parent columns must be filled out for every line in the journal instance.

Entity Member Filter
This Member Filter will help limit the list of entities presented to the user in the Journal POV and journal lines.

Point of View
In order to limit the amount of setup for every journal line, the items that remain constant (e.g. Flow = None) can be set in the Journal POV instead of in every line Item.

> **TIP:** A Journal template can be repeated on a regular basis if values are placed in the journal lines and journal's settings require repeating upon a certain frequency.

Journal Template Group Properties

This section lists the general group properties.

General (Journal Template Group)

Name
The name of the Journal Template group.

Description
A short description of the Template group such as how or where it is used.

Security

Access Group
Members of this group have access to the Journal Template group

Maintenance Group

Members of this group have the authority to maintain the Journal Template group

> **NOTE:** Click  to use the Security page to modify users and groups before assigning them. Click  and begin typing the name of the group. As the first few letters are typed, groups are filtered making it easier to find and select the desired group. Select the group, press CTRL and then double-click.

Loading Data With Excel Templates or CSV Files

Use Excel templates or CSV files to load data into stage using the Import method in Workflow, Forms data, Journals, Cell Detail, Data Attachments or to Custom Tables. An Excel file can be imported without having to configure the data source completely by setting Allow Dynamic Excel Loads to True and configuring an Excel template. This section describes the configuration required for this integration.

Loading Stage Data

Import Excel Template

When importing data into the Stage via Excel, a specific template must be used for OneStream to read and load the data. The user must first create Dimension Tokens such as, E#, A# or S#, to organize the data correctly. The Dimension Token specifies the specific type of data in any given column. For example, if the column header is E#, OneStream will read every row in that column as an Entity name when loading into Stage. After the Dimension Tokens are specified, create a Named Range beginning with xfd. There can multiple xfd Named Ranges across multiple tabs within an Excel workbook. The Dimension Tokens must be the Named Range's first row.

> **NOTE:** To use zeroes in an XFD template, you must implement a parser rule to facilitate. Otherwise, Excel ignores suppression settings and attribute settings.

The following Dimension Tokens are used within an Import Excel Template. Please note these tokens can be in any order on the Excel template.

Dimension Tokens	Meaning
A#	Account: each row below will list the accounts to be imported
AMT#	Amount: using the AMT.ZS# header will automatically apply zero suppression to this import. **TIP:** Apply zero suppression to a Matrix-style Excel import template by using this same .ZS extension on the AMT column.
F#	Flow
IC#	Intercompany
E#	Entity
C#	Consolidation
S#	Scenario
T#	Time Period
V#	View
O#	Origin

Dimension Tokens	Meaning
UD1#-UD8#	Each row must have a value even if a User Defined Member is not used in the application. Create a Static Value of None for any UD Members where this applies. Ex. UD5#:[None] UX# or UDX# can be used for all User Defined Dimensions.
LB#	Label: This is used for an Account description related to a line of data. It is imported just for reference purposes and not stored in the cube.
SI#	Source ID: This is a key for data imported into Stage. This typically includes a reference to the Entity being loaded but depends on the implementation. It is a best practice to have only one Source ID per Named Range and these can be the same or different for every Named Range imported for one Excel workbook.
TV#	Text Value: this is used to store large amounts of textual data.
A1# through A20#	Attribute Dimensions: these 20 dimensions can each store 100 characters of text.
AV1# through AV12#	Attribute Value Dimensions: these 12 dimensions can store numeric data.

Header Abbreviations

Static Value

Use :[] to fix a specific Member to the entire column creating a Static Value for the specified Source Dimension. For example, F#:[None] imports the None Flow Member for every Flow row within the Named Range. This syntax applies to all Dimension Tokens.

Data Sources allow text values to be loaded as a View Member from the same row as the numeric value. Specify #Annotation, #VarianceExplanation, #AuditComment, #Footnote, or #Assumption as the Static Text Value of the TextValue Source Dimension and a new row will be created for the comment row. For example, use TV#:[#Annotation] to add an additional Annotation row.

Business Rule

Pass a Business Rule for any specified Source Dimension to set a specific value.

AMT#:[]:[BusinessRuleNameThatSetsAValue]

Matrix Member

This repeats for each Member. For example, if there were twelve time periods in the named range the syntax would be as follows:

T#:[]:[]:[2012M3]

To use Current/Global Scenario and Time, use .C# and .G# which creates a Static Value for the Time and Scenario within the Named Range. T.C# and S.C# returns the current Workflow Time and Scenario. T.G# and S.G# returns the Global Time and Scenario.

Import Data Extracted via Data Management

Any type of data (import, forms, or journals) extracted to a CSV file through a Data Management Job can be imported into Stage via an Extensibility Business Rule. This simplifies the migration of data between applications.

Example

Dim objXFResult As XFResult = BRApi.Finance.Data.SetDataCellsUsingCsvFile(si, filePath, delimiter, originFilter, targetOriginMember, loadZeros)

When using this BRApi make sure to specify the Origin Filter which determines the type of data desired from the file (import, forms or adjustments), and the Target Origin Member which determines where the data will be stored upon loading the file.

Loading Form Data

This section describes how to load form data from various sources.

Form Excel Template

When loading form data via Excel, a specific template must be created to determine the form properties, the dimensions to which the data is loaded, the data entry amount, and any data attachment information.

OneStream reads this template using a specific Named Range which is explained later in this section. Ensure the following information is included in the Named Range.

Property Tokens
The first four rows of the Named Range in the Excel template must include the following token definitions:

Form Import Template

FormTemplateName:	Accounting Standard Adjustments	Form Template Name
Workflow Name:	Houston.Forms	Valid Form Workflow Name
Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	|WFTime|	Workflow Time Name

Form Template Name

Enter the Form Template name intended for the Form data load.

Workflow Name

Enter the Form Workflow name. For example, if the name of the Workflow Profile is Houston, and the Form input type is named Forms, enter Houston.Forms.

Workflow Scenario

Enter the current Workflow Scenario such as Actual, Budget, etc. To dynamically use the current Workflow Scenario, use the |WFScenario| Substitution Variable.

Workflow Time

Enter the current Workflow Time Period. To dynamically use the current Workflow Time, use the |WFTime| Substitution Variable.

Dimension Tokens

Next, create the Dimension Tokens necessary to load the form data to the correct Dimensions in OneStream. The Dimension tokens need to be the column header for each data row. The standard tokens used determine the Cube, Entity, Parent, Account, Flow, Intercompany, the User Defined Members, and an Amount. Refer to Loading Stage Data for the syntax. The form specific tokens are as follows:

HD#

Has Data

Enter Yes or No to specify whether the row has data.

AN#

Annotation

AS#

Assumption

AD#

Audit Comment

FN#

Footnote

VE#

Variance Explanation

Header Abbreviations

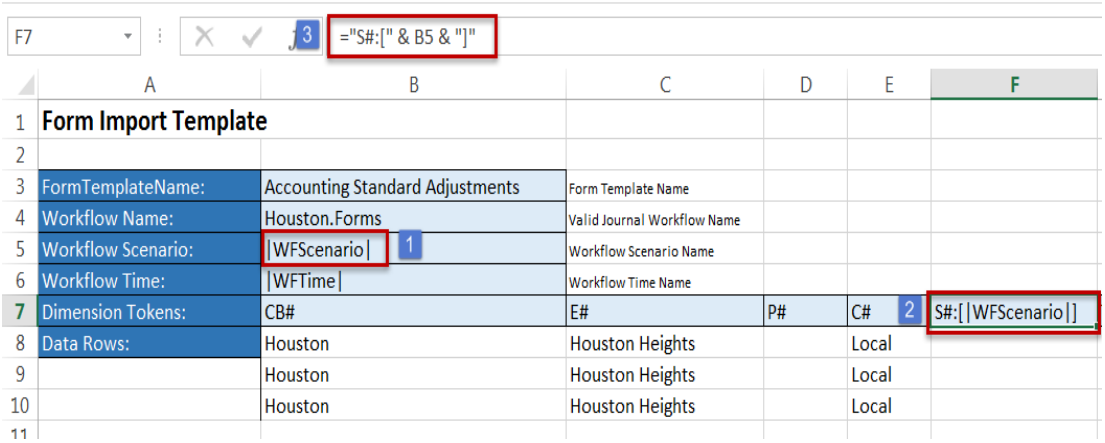
Static Value

Use :[] to fix a specific Member to the entire column creating a Static Value for the specified Source Dimension. For example, F#:[None] imports the None Flow Member for every Flow row within the Named Range. This syntax applies to all Dimension Tokens.

Using Substitution Variables

If a Substitution Variable is used to define the Workflow Scenario or Workflow Time Tokens, link the Scenario and Time Dimension Tokens using the Ampersand (&) Excel Function and referencing the Excel cell.

Example:



1. The Workflow Scenario Token, located in cell B5, is using a Substitution Variable to dynamically reference the user's current Scenario.
2. The Scenario Dimension Token needs to reference that Substitution Variable to ensure the correct Scenario is used and the template functions properly.
3. The syntax to reference the cell B5 is =”S#:[“ & B5 &”]”
   This references the correct variable and displays it in the proper cell.

After the Dimension Tokens are configured, enter the data in the corresponding column. The Dimensions can be in any order.

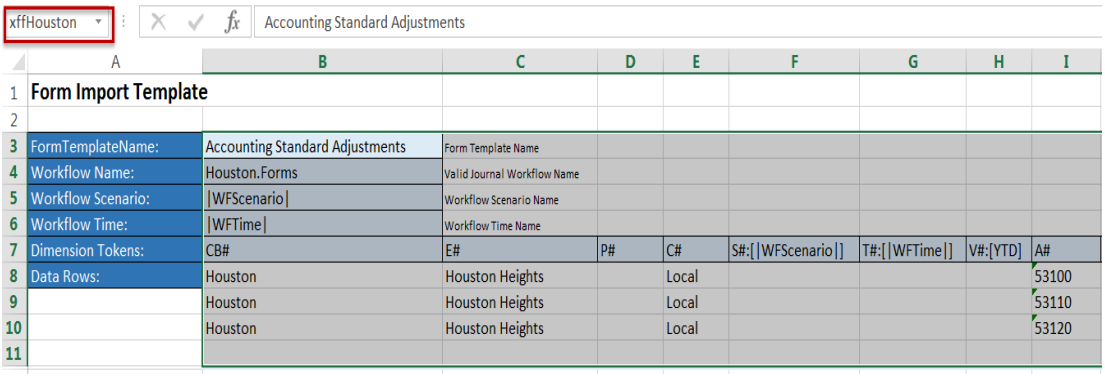
Form Import Template

FormTemplateName:	Accounting Standard Adjustments	Form Template Name						
Workflow Name:	Houston.Forms	Valid Journal Workflow Name						
Workflow Scenario:	|WFScenario|	Workflow Scenario Name						
Workflow Time:	|WFTime|	Workflow Time Name						
Dimension Tokens:	CB#	E#	P#	C#	S#:[|WFScenario|]	T#:[|WFTime|]	V#:[YTD]	A#
Data Rows:	Houston	Houston Heights		Local				53100
	Houston	Houston Heights		Local				53110
	Houston	Houston Heights		Local				53120

F#	O#:[Forms	IC#	UD1#	UD2#	UD3#	UD4#	UD5#:[IFRS Adj]	UD6#	UD7#	UD8#	AMT#
None		None	None	None	None	None		None	None	None	9,000.00
None		None	None	None	None	None		None	None	None	9,000.00
None		None	None	None	None	None		None	None	None	12,000.00

HD#:[Yes]	AN#	AS#	AD#	FN#	VE#

The final step is to create a Named Range beginning with XFF making sure to include the definition of each property, the Dimension tokens, and the data rows. The Named Range must begin with XFF for OneStream to read and load the form data correctly. Multiple XFF Named Ranges can be used across multiple tabs.

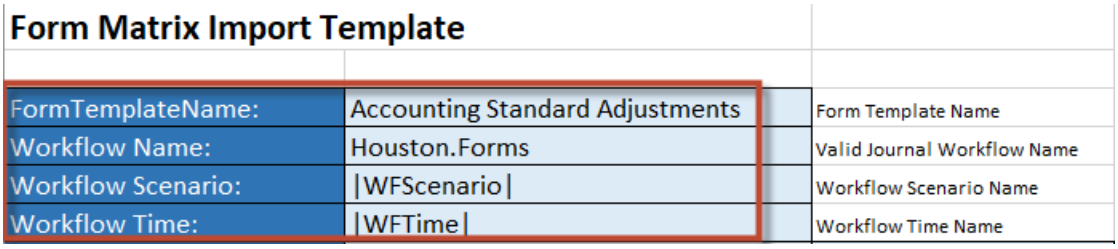


	A	B	C	D	E	F	G	H	I
1	Form Import Template								
2									
3	FormTemplateName:	Accounting Standard Adjustments	Form Template Name						
4	Workflow Name:	Houston.Forms	Valid Journal Workflow Name						
5	Workflow Scenario:	|WFScenario|	Workflow Scenario Name						
6	Workflow Time:	|WFTime|	Workflow Time Name						
7	Dimension Tokens:	CB#	E#	P#	C#	S#:[|WFScenario|]	T#:[|WFTime|]	V#:[YTD]	A#
8	Data Rows:	Houston	Houston Heights		Local				53100
9		Houston	Houston Heights		Local				53110
10		Houston	Houston Heights		Local				53120
11									

Form Matrix Excel Template

A form matrix template is used to load multiple amount columns to multiple time periods at once. In this template, the Time Dimension Token is combined with the amount to identify which amount should load to which period.

This template uses the same property tokens as a regular Excel Form template shown above.



Form Matrix Import Template

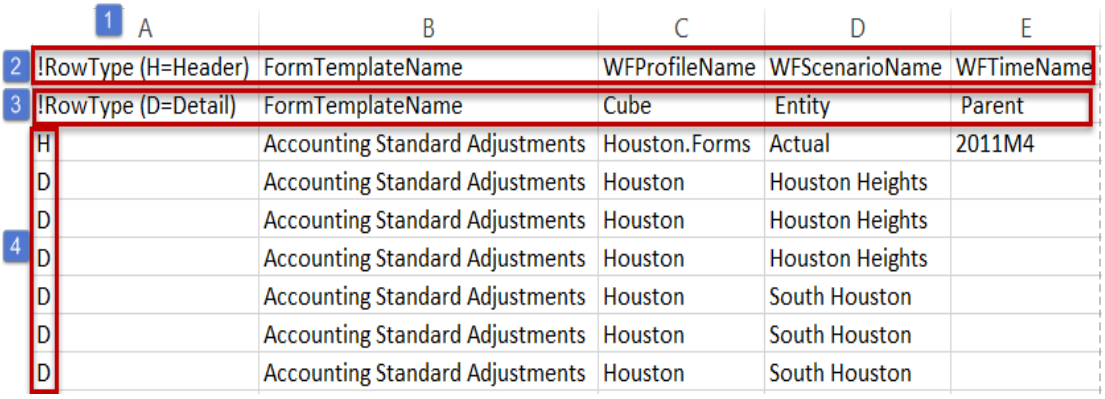
FormTemplateName:	Accounting Standard Adjustments		Form Template Name
Workflow Name:	Houston.Forms		Valid Journal Workflow Name
Workflow Scenario:	|WFScenario|		Workflow Scenario Name
Workflow Time:	|WFTime|		Workflow Time Name

In the Matrix Form template, Amount and Time must be specified in the same column. A third Dimension can be specified such as Scenario, if desired. The example below is indicating the Amount Column using AMT# and then specifying to which Time Members the Amount detail belongs.

AMT#::T#2011M1	AMT#::T#2011M2	AMT#::T#2011M3	
1,000.00	2,000.00	3,000.00	
2,000.00	4,000.00	6,000.00	
3,000.00	6,000.00	9,000.00	

Form CSV Template

To set up a CSV template for a Form, the Header and Detail values must be specified.



	A	B	C	D	E
2	!RowType (H=Header)	FormTemplateName	WFProfileName	WFScenarioName	WFTimeName
3	!RowType (D=Detail)	FormTemplateName	Cube	Entity	Parent
	H	Accounting Standard Adjustments	Houston.Forms	Actual	2011M4
	D	Accounting Standard Adjustments	Houston	Houston Heights	
	D	Accounting Standard Adjustments	Houston	Houston Heights	
4	D	Accounting Standard Adjustments	Houston	Houston Heights	
	D	Accounting Standard Adjustments	Houston	South Houston	
	D	Accounting Standard Adjustments	Houston	South Houston	
	D	Accounting Standard Adjustments	Houston	South Houston	

1. Column A Specifies Row Type

   In the first two rows of Column A, create two Row Type Parameters specifying the Header and the Detail. In the example above, !RowType (H=Header) and !RowType (D=Detail) are used to tag the corresponding rows with H or D identifying where the Header and Detail information is located in the CSV file.

2. Row One Specifies the Headers

   After the Header Parameter is configured, enter the form column headers. The required Form Headers are FormTemplateName, WFProfileName, WFScenarioName, and WFTimeName. For more details on these, refer to Form Excel Template.

3. Row Two Specifies the Details

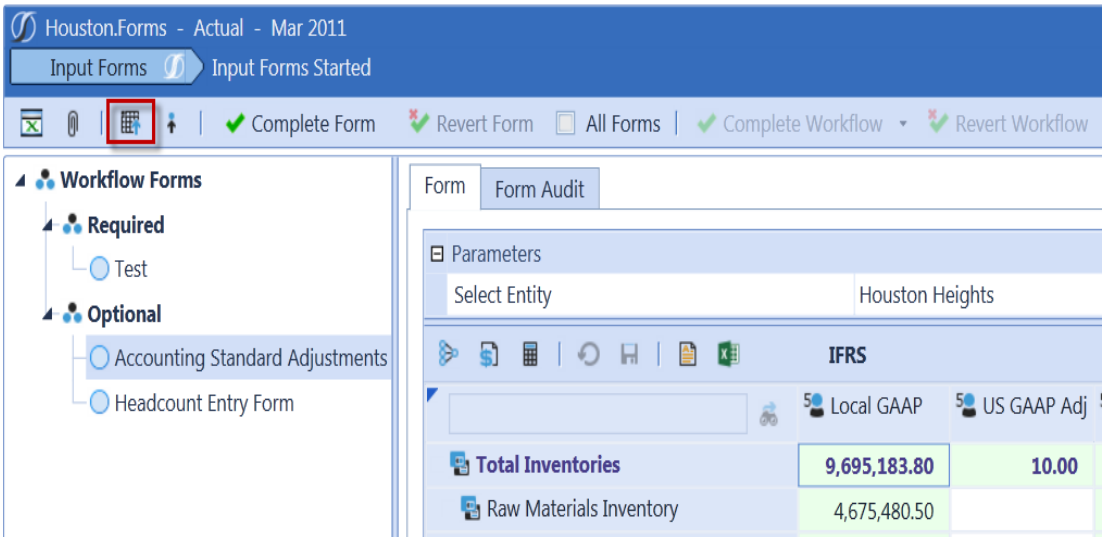
   After the Detail Parameter is configured, enter the form detail headers. The required Form Detail Headers are FormTemplateName, all 18 standard Dimensions, Amount, HasData, Annotation, Assumptions, AuditComment, Footnote, and VarianceExplanation. For more details on these, refer to Form Excel Template.

4. Header and Detail Tags

   The Form data is driven by how each row in column A is tagged. Any rows tagged with H load as the Headers and any row tagged with D load as the details.

Loading Form Details via Workflow

After the template is configured, users can load it directly into OneStream during the Workflow process. While the Form Input type is selected, click the  icon in the Form toolbar. This allows the user to select the desired Excel or CSV template and load it into OneStream. After the file is loaded, the data will appear in the Form grid and it auto-saves upon importing to the Cube.



Loading Form Details via Business Rules

Users can load Form details from an Excel Template or CSV file by configuring an Extensibility Business Rule and using the ImportandProcessForms BRApi. Within the BRApi, define the Session Info, file path, and Form actions.

Load Example

'BRApi.Forms.Data.ImportAndProcessForms(si, filePath, save, complete, throwOnError)

Loading Journal Data

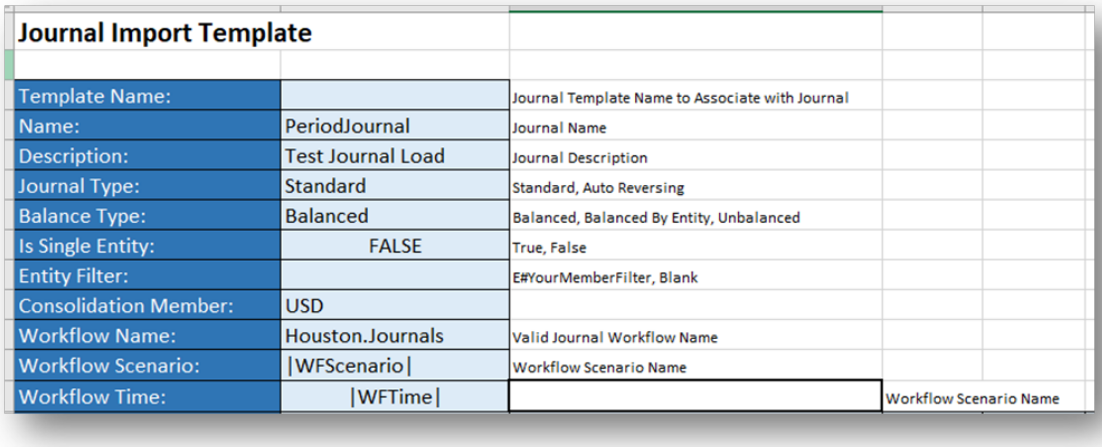
This section describes how to load journal data from various sources.

Journal Excel Template

When loading journal entries via Excel a specific template must be created to determine the journal properties, the Dimensions to which the entries are being made, and the debit and credit amounts.

OneStream reads this template using a specific Named Range which is explained later in this section. Ensure the following information is included in the Named Range.

Property Tokens
The first eleven rows of the Named Range in the Excel template must include the following token definitions:



Journal Import Template			
Template Name:		Journal Template Name to Associate with Journal	
Name:	PeriodJournal	Journal Name	
Description:	Test Journal Load	Journal Description	
Journal Type:	Standard	Standard, Auto Reversing	
Balance Type:	Balanced	Balanced, Balanced By Entity, Unbalanced	
Is Single Entity:	FALSE	True, False	
Entity Filter:		E#YourMemberFilter, Blank	
Consolidation Member:	USD		
Workflow Name:	Houston.Journals	Valid Journal Workflow Name	
Workflow Scenario:	|WFScenario|	Workflow Scenario Name	
Workflow Time:	|WFTime|		Workflow Scenario Name

Template Name

Enter a template name is applicable. If this is a free form journal, leave this blank.

Name

Enter the name of the journal.

Description

If desired, enter a description for the journal.

Journal Type*

Standard or Auto-reversing. Allocation is not supported for Excel or CSV import.

Balance Type*

Balanced, Balanced by Entity, or Unbalanced

Is Single Entity*

True or False

Entity Filter*

Use a Member Filter to specify the Entities used with this journal.

Consolidation Member

Enter the specific currency or Local Member of Consolidation.

Workflow Name

Enter the Journal Workflow name. For example, if the name of the Workflow Profile is Houston, and the Adj input type is named Journals, enter Houston.Journals.

Workflow Scenario

Enter the Workflow Scenario or make the template dynamic by entering the |WFScenario| Substitution Variable.

Workflow Time

Enter the Workflow Time or make the template dynamic by entering the |WFTime| Substitution Variable. Workflow Time support two fields. The available cell immediately to the right is option representing the CubeTimeName. This field can be used when the Scenario's Workflow Tracking Frequency is Yearly, and the Input Frequency is Monthly. For example, the Workflow Time would be |WFTime| or 2019 and the CubeTimeName would be the period to post, 2019M7.

Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	|WFTime|	

When Tracking Frequency is Yearly and Input Frequency is Monthly:

Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	2019M7	

Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	|WFTime|	2019M7

Workflow Scenario:	|WFScenario|	Workflow Scenario Name
Workflow Time:	|WFTime|	|WFYear|M7

*See Journal Templates for details on these Journal properties.

Dimension Tokens

Next, create the Dimension Tokens necessary to load the journal to the correct Dimensions in OneStream. The Dimension tokens need to be the column header for each data row. The standard tokens used determine the Cube, Entity, Parent, Account, Flow, Intercompany, the User Defined Members, and a Label if needed. Refer to Loading Stage Data for the syntax. The journal specific tokens are as follows:

AMTDR#

This indicates the debited amount.

AMTCR#

This indicates the credited amount.

Header Abbreviations

Static Value

Use :[] to fix a specific Member to the entire column creating a Static Value for the specified Source Dimension. For example, F#:[None] imports the None Flow Member for every Flow row within the Named Range. This syntax applies to all Dimension Tokens.

After the Dimension Tokens are setup, enter the data in the corresponding column.

Template Example:

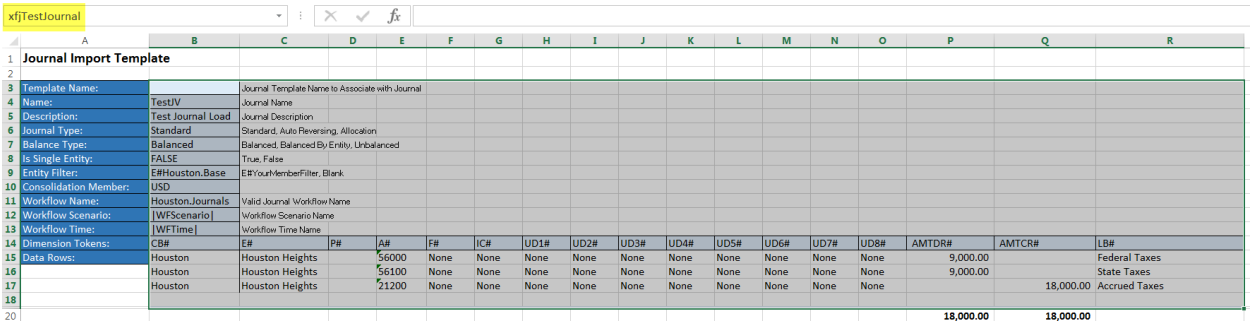
Journal Import Template

Template Name:		Journal Template Name to Associate with Journal															
Name:	TestJV	Journal Name															
Description:	Test Journal Load	Journal Description															
Journal Type:	Standard	Standard, Auto Reversing, Allocation															
Balance Type:	Balanced	Balanced, Balanced By Entity, Unbalanced															
Is Single Entity:	FALSE	True, False															
Entity Filter:	E#Houston.Base	E#YourMemberFilter, Blank															
Consolidation Member:	USD																
Workflow Name:	Houston.Journals	Valid Journal Workflow Name															
Workflow Scenario:	|WFScenario|	Workflow Scenario Name															
Workflow Time:	|WFTime|	Workflow Time Name															
Dimension Tokens:	CB#	E#	P#	A#	F#	IC#	UD1#	UD2#	UD3#	UD4#	UD5#	UD6#	UD7#	UD8#	AMTDR#	AMTCR#	LB#
Data Rows:	Houston	Houston Heights		56000	None	None	None	None	None	None	None	None	None	None	9,000.00		Federal Taxes
	Houston	Houston Heights		56100	None	None	None	None	None	None	None	None	None	None	9,000.00		State Taxes
	Houston	Houston Heights		21200	None	None	None	None	None	None	None	None	None	None		18,000.00	Accrued Taxes
															18,000.00	**18,000.00**	

The final step is to create a XFJ Named Range making sure to include the definition of each property, the Dimension tokens, and the data rows. The Named Range must begin with XFJ for OneStream to read and load it correctly. Multiple XFJ Named Ranges can be used within the template over multiple tabs.

> **NOTE:** Loading of Journal Templates or previously exported Journal data only requires a Parent (P#) column value to be populated if the target Consolidation dimension member being updated is OwnerPreAdj or OwnerPostAdj. Otherwise, this entry can be left blank.

Named Range Example:

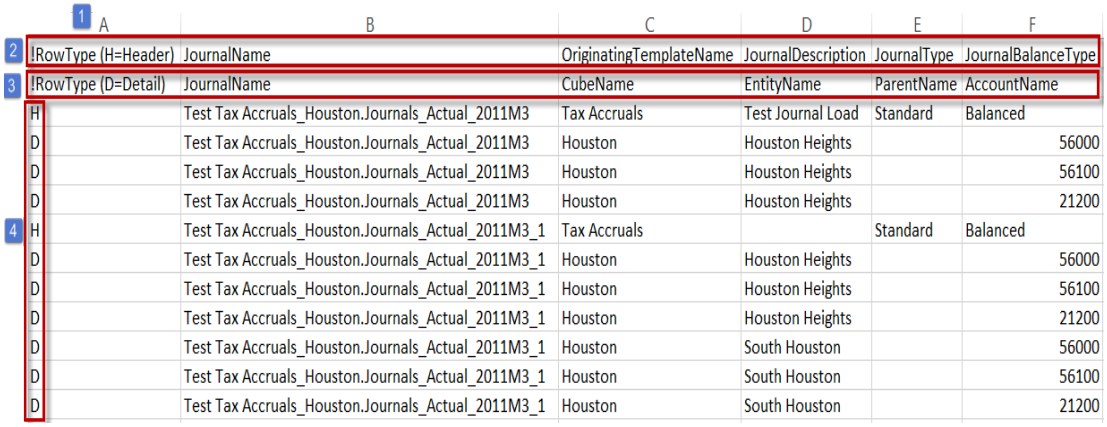


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	**Journal Import Template**																	
2																		
3	Template Name:		Journal Template Name to Associate with Journal															
4	Name:	TestJV	Journal Name															
5	Description:	Test Journal Load	Journal Description															
6	Journal Type:	Standard	Standard, Auto Reversing, Allocation															
7	Balance Type:	Balanced	Balanced, Balanced By Entity, Unbalanced															
8	Is Single Entity:	FALSE	True, False															
9	Entity Filter:	E#Houston.Base	E#YourMemberFilter, Blank															
10	Consolidation Member:	USD																
11	Workflow Name:	Houston.Journals	Valid Journal Workflow Name															
12	Workflow Scenario:	|WFScenario|	Workflow Scenario Name															
13	Workflow Time:	|WFTime|	Workflow Time Name															
14	Dimension Tokens:	CB#	E#	P#	A#	F#	IC#	UD1#	UD2#	UD3#	UD4#	UD5#	UD6#	UD7#	UD8#	AMTDR#	AMTCR#	LB#
15	Data Rows:	Houston	Houston Heights		56000	None	None	None	None	None	None	None	None	None	None	9,000.00		Federal Taxes
16		Houston	Houston Heights		56100	None	None	None	None	None	None	None	None	None	None	9,000.00		State Taxes
17		Houston	Houston Heights		21200	None	None	None	None	None	None	None	None	None	None		18,000.00	Accrued Taxes
18																		
20																18,000.00	18,000.00	

> **NOTE:** The Named Range only covers each property definition.

Journal CSV Template

To set up a CSV template for a Journal, the Header and Detail values must be specified.



	A	B	C	D	E	F
2	!RowType (H=Header)	JournalName	OriginatingTemplateName	JournalDescription	JournalType	JournalBalanceType
3	!RowType (D=Detail)	JournalName	CubeName	EntityName	ParentName	AccountName
	H	Test Tax Accruals_Houston.Journals_Actual_2011M3	Tax Accruals	Test Journal Load	Standard	Balanced
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3	Houston	Houston Heights		56000
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3	Houston	Houston Heights		56100
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3	Houston	Houston Heights		21200
4	H	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Tax Accruals		Standard	Balanced
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	Houston Heights		56000
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	Houston Heights		56100
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	Houston Heights		21200
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	South Houston		56000
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	South Houston		56100
	D	Test Tax Accruals_Houston.Journals_Actual_2011M3_1	Houston	South Houston		21200

1. Column A Specifies Row Type

   In the first two rows of Column A, create two Row Type Parameters specifying the Header and the Detail. In the example above, !RowType (H=Header) and !RowType (D=Detail) are used to tag the corresponding rows with H or D identifying where the Header and Detail information is located in the CSV file.

2. Row One Specifies the Headers

   After the Header Parameter is configured, enter the Journal Column Headers. The required Journal Headers are JournalName, OrigniatingTemplateName, JournalDescription,JournalType, JournalBalanceType, IsSingleEntity, EntityMemberFilter, ConsName, WFProfileName, WFScenarioName, WFTimeName and CubeTimeName. For more details on these, refer to the Journal Excel Template.

3. Row Two Specifies the Details

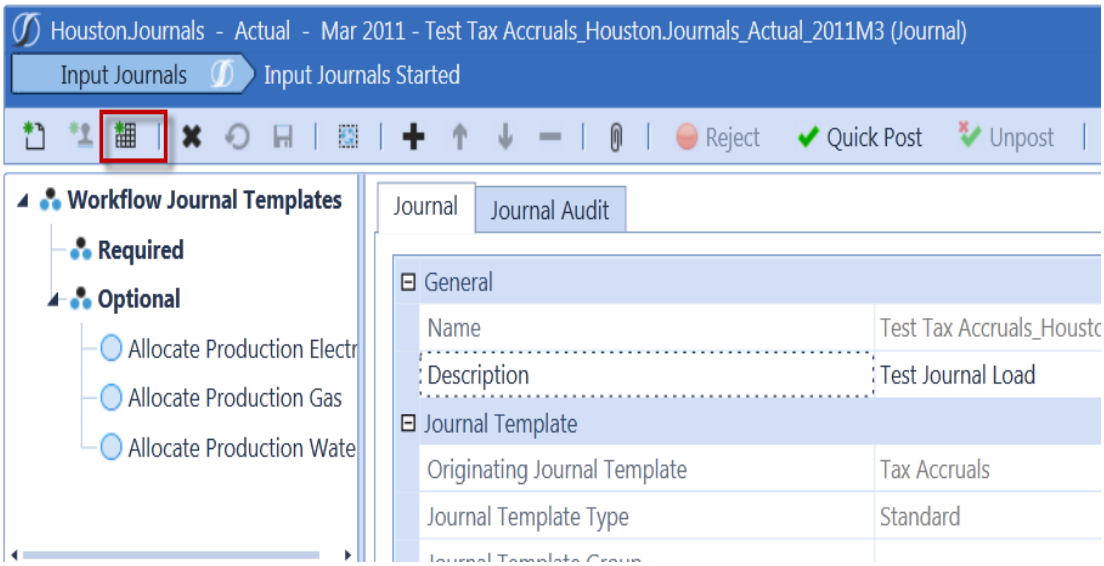
   After the Detail Parameter is configured, enter the Journal Detail Headers. The required Journal Detail Headers are JournalName, CubeName, EntityName, ParentName, AccountName, FlowName, ICName, all UDNames, DebitAmount, CreditAmount, and LineDescription. For more details on these, refer to Journal Excel Template.

4. Header and Detail Tags

   The Journal data is driven by how each row in column A is tagged. Any rows tagged with H load as the Headers and any row tagged with D load as the details.

Loading Journal Details via Workflow

After the template is configured, users can load it directly into OneStream during the Workflow process. While the Journal Input type is selected, click the  icon in the Journal toolbar. This lets you select the desired Excel or CSV template and load it into OneStream. After the file is loaded, the journal line items will appear in the journal and the user can save it to the Cube.



Extracting/Loading Journal Details via BRApi

Users can export journal details into a CSV or XSLX file and load journals from a CSV or XSLX file by configuring an Extensibility Business Rule. This allows journal details to be extracted from one application and loaded into another. Users can also extract the journal, make changes, and re-load.

Extract Journals to CSV

Use BRApi.Journals.Data.ExportJournalstoCSV and define the session, filepath, Workflow Profile, Scenario, Time Filter, and Journal Status.

BRApi.Journals.Data.ExportJournalsToCsv(si, filePath, "Houston", "Actual", "T#|WFYear|.Base", "Posted")

Extract Journals to XSLX

Export journals associated with defined parameters to an XSLX file using the ExportJournalsToXSLX BRApi. It is sometimes easier to read and modify journal data in an XSLX format as compared to CSV files that are generated by the ExportJournalsToCSV BRApi. Use the parameters listed in the following table.

Parameter	Description
si	Session information
serverFilePath	Name and location to save the file
wfProfileName	Name of the Workflow Profile
wfScenarioName	Name of the Scenario

Parameter	Description
wfTimeMemberFilter	Filters workflow profile times. For example, if the scenario's Workflow Tracking Frequency is set to Quarterly, and Input Frequency is set to Monthly, the filter searches for journals associated with the Workflow Tracking Frequency. Time must be prefixed with T#. You can type a comma separated list of times such as T#2022H1, T#2022H2.
journalName	Filters journals that contain the specified text
journalStatus	Status of the journal. You can search by status or use All to return all statuses. You cannot type a comma separated list.

The following code example uses the ExportJournalsToXslx BRApi:

```
Imports System.Data
Imports System.Data.Common
Imports System.Globalization
Imports System.IO
Imports System.LInq
Imports Microsoft.VisualBasic
Imports OneStream.Finance.Database
Imports OneStream.Finance.Engine
Imports OneStream.Shared.Common
Imports OneStream.Shared.Database
Imports OneStream.Shared.Engine
Imports OneStream.Shared.vcf
Imports OneStream.Stage.Database
Imports OneStream.Stage.Engine

Namespace OneStream.BusinessRule.Extender.ExportJournalstoXSLX
    Public Class MainClass
        Public Function Main(ByVal s1 As SessionInfo, ByVal globals As BRGlobals, ByVal api
As Object, ByVal args as ExtenderArgs) As Object
            Try
```

```
                Dim configSettings As AppServerConfigSettings = AppServerConfig.GetSettings
(si)
                Dim sourceFolder As String = BRApi.Utilities.GetFileShareFolder
(si,FileShareFolderTypes.FileShareRoot,Nothing) & "\System\Contents\Export"
                Dim journalFilePath As String = sourcefolder & "\" & "Journal_Export.xlsx"
                Dim filePath As String = BRApi.Journals.Data.ExportJournalsToXlsx(si,
journalFilePath, "Houston", "Actual", "T#2023M5", "", All)

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

Load Example

Use BRApi.Journals.Data.ImportAndProcessJournals and define the session, filepath, and journal tasks to complete upon loading the journal details.

'BRApi.Journals.Data.ImportAndProcessJournals(si, filePath, save, submit, approve, post, unpostAndOverwrite, throwOnError)

Loading Cell Detail

Cell Detail Excel Template

When loading Cell Detail via Excel, a specific template must be created to determine the Cell Detail Dimension Tokens and each Cell Detail line data.

OneStream reads this template using a specific Named Range which is explained later in this section. Ensure the following information is included in the Named Range.

Dimension Tokens

The first 19 rows of the Named Range in the Excel template must include the following token definitions:

CubeName	Houston
EntityName	Houston Heights
ParentName	
ConsName	USD
ScenarioName	ActualTest
TimeName	2011M3
AccountName	56100
FlowName	None
Origin	Forms
ICName	None
UD1Name	None
UD2Name	None
UD3Name	None
UD4Name	None
UD5Name	IFRS Adj
UD6Name	None
UD7Name	None
UD8Name	None
DetailViewType	Periodic

The Cube and each Dimension Member must be specified. All User Defined Members must be specified. If a specific User Defined Member is not used in the application, enter None.

Next, create the Dimension Tokens necessary to load each Cell Detail line. The specific tokens are as follows:

Dimension Tokens:		AMT#	LIT#	AW#	CL#	LB#
Data Rows:		700.00	Default	1.00		Description or Detail

AMT#

Amount

LIT#

Line Item Type

AW#

Aggregation Weight

CL#

Classification Type

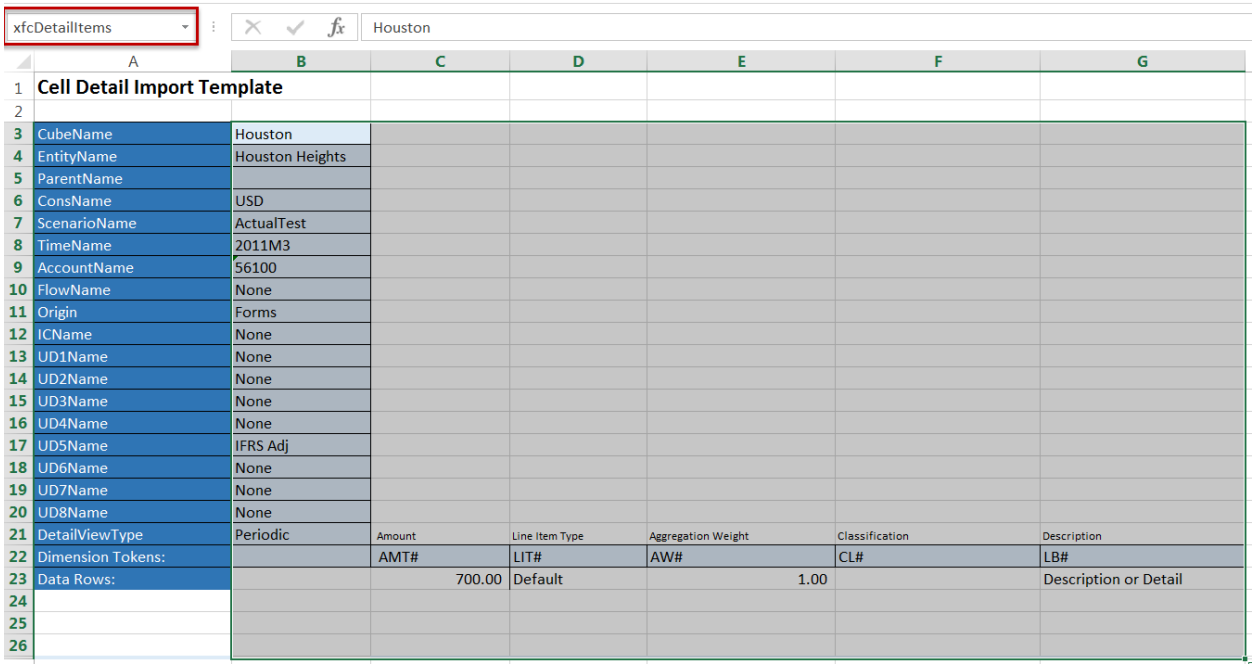
LB#

Label allowing users to add a description or additional detail.

After the Dimension Tokens are configured, enter the data in the corresponding column. The Dimensions can be in any order.

CubeName	Houston					
EntityName	Houston Heights					
ParentName						
ConsName	USD					
ScenarioName	ActualTest					
TimeName	2011M3					
AccountName	56100					
FlowName	None					
Origin	Forms					
ICName	None					
UD1Name	None					
UD2Name	None					
UD3Name	None					
UD4Name	None					
UD5Name	IFRS Adj					
UD6Name	None					
UD7Name	None					
UD8Name	None					
DetailViewType	Periodic	Amount	Line Item Type	Aggregation Weight	Classification	Description
Dimension Tokens:		AMT#	LIT#	AW#	CL#	LB#
Data Rows:		700.00	Default	1.00		Description or Detail

The final step is to create a Named Range beginning with XFC making sure to include the definition of Dimension token, and the data rows. The Named Range must begin with XFC for OneStream to read and load the Cell Detail correctly. Multiple XFC Named Ranges can be used across multiple tabs.



xfcDetailItems | Houston

	A	B	C	D	E	F	G
1	Cell Detail Import Template						
2							
3	CubeName	Houston					
4	EntityName	Houston Heights					
5	ParentName						
6	ConsName	USD					
7	ScenarioName	ActualTest					
8	TimeName	2011M3					
9	AccountName	56100					
10	FlowName	None					
11	Origin	Forms					
12	ICName	None					
13	UD1Name	None					
14	UD2Name	None					
15	UD3Name	None					
16	UD4Name	None					
17	UD5Name	IFRS Adj					
18	UD6Name	None					
19	UD7Name	None					
20	UD8Name	None					
21	DetailViewType	Periodic	Amount	Line Item Type	Aggregation Weight	Classification	Description
22	Dimension Tokens:		AMT#	LIT#	AW#	CL#	LB#
23	Data Rows:		700.00	Default	1.00		Description or Detail
24							
25							
26							

Cell Detail CSV Template

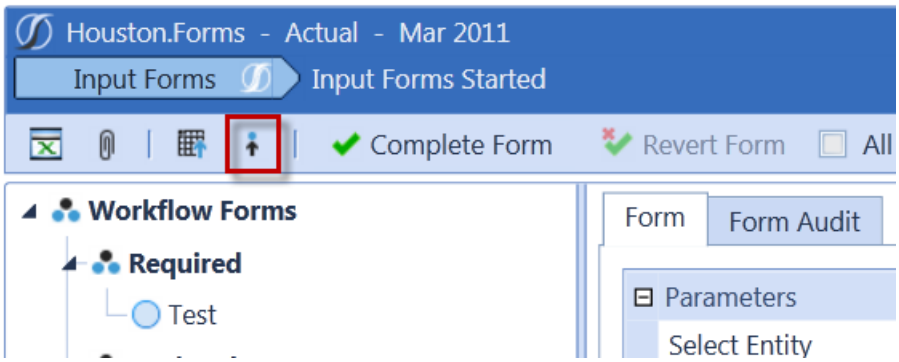
To set up a CSV template for Cell Detail, the Header and Detail values must be specified.

	1 A	B	C	D	E	F	G	H
2	!H (RowType Header)	CubeName	EntityName	ParentName	ConsName	ScenarioName	TimeName	AccountName
3	!D (RowType Detail)	Amount	LineItemType	AggregationWeight	Classification	Description		
4	H	Houston	Houston Heights		USD	Actual	2011M3	56100
	D	700	Default	1	Class1	Desc1		
	H	Houston	South Houston		USD	Actual	2011M3	56100
	D	2000	Default	1	Rebates Received on Supplies	Desc2		

1. Column A Specifies Row Type
   In the first two rows of Column A, create two Row Type Parameters specifying the Header and the Detail. In the example above, !RowType (H=Header) and !RowType (D=Detail) are used to tag the corresponding rows with H or D identifying where the Header and Detail information is located in the CSV file.
2. Row One Specifies the Headers
   After the Header Parameter is configured, enter the Cell Detail column headers. The required Cell Detail Headers specify the cube and all 18 Dimension Members. For more details on these, refer to Cell Detail Excel Template.
3. Row Two Specifies the Details
   After the Detail Parameter is configured, enter the Cell Detail detail headers. The required Detail Headers are Amount, LineItemType, AggregationWeight, Classification, and Description. For more details on these, refer to Cell Detail Excel Template.
4. Header and Detail Tags
   The Cell Detail's data is driven by how each row in column A is tagged. Any rows tagged with H load as the Headers and any row tagged with D load as the details.

Loading Cell Detail via Workflow

After the template is configured, users can load it directly into OneStream during the Workflow process. While the Form Input type is selected, click the  icon in the Form toolbar. This allows the user to select the desired Excel or CSV template and load it into OneStream. After the file is loaded, it has been successfully stored to the cube.



Extracting/Loading Cell Detail via Business Rules

Users can export Cell Details into a CSV file and load Cell Details from a CSV file by configuring an Extensibility Business Rule. This allows Cell Detail to be extracted from one application and loaded into another. Users can also extract the Cell Detail, make changes, and re-load.

Extract

Use BRApi.Finance.Data.ExportCellDetailtoCSV and define the session, filepath, the Entity Dimension, the Entity Member Filter, the Scenario and the Time Member Filter.

BRApi.Finance.Data.ExportCellDetailToCsv(si, filePath, entityDimensionName, entityMemberFilter, scenarioName, timeMemberFilter)

Load Example

Use BRApi.Finance.Data.ImportCellDetail and define the session and filepath.

BRApi.Finance.Data.ImportCellDetail(si, filePath, throwOnError)

Exporting Data Attachments

Exporting Data Attachments via Business Rule

You can export data attachments to perform variety of actions such as retrieve the file name, read the contents, store them in a File Explorer folder, or import them into another application. This process is configured in an Extensibility Business Rule using the following API.

Example:
```
Dim objDataAttachmentList As
DataAttachmentList = BRApi.Finance.Data.
GetDataAttachments(si, memberScript,
includeFileBytes)
```

Exporting Data Attachment Text

You can export the Data Attachment Text field to a CSV file using an Extensibility Business Rule. For example, use BRApi.Finance.Data.ExportCellTextToCSV and define the session, filepath, the Entity Dimension, the Entity Member Filter, the Scenario and the Time Member Filter.

```
BRApi.Finance.Data.ExportCellTextToCsv(si, cellTextFilePath,
entityDimensionName, entityMemberFilter, scenarioName,
timeMemberFilter)
```

Loading Excel Templates to Custom Tables

OneStream Solutions typically have related SQL Server tables. Other custom solutions may also include adding custom SQL Server tables. This allows users to load data to these custom tables using an Excel template. The mechanism for loading these tables could be through the user interface of a OneStream Solution or through OneStream Extensibility Rules.

OneStream reads this template using a specific Named Range which is explained later in this section. Ensure the following information is included in the Named Range.

In the first three rows of the Named Range in Column A, specify the following:

Database Location
Application or System specifies which database contains the custom tables.

Table Name
Custom tables only; enter the Table name.

Load Method
The load method determines the action and any additional criteria for the action.

The syntax is:
Action:[Where Clause Criteria] (Where Clause Criteria is optional)

Load Method Definitions

Merge
If there are no criteria, Merge updates the data if it finds a matching key, otherwise it inserts it.

Merge Where Clause Criteria Example
First, this will clear the values for emp1 and then Merge.
Merge:[EmployeeID = ‘emp1’]

Merge Where Clause Criteria with Substitution Variable Example

Substitution Variables can be used in the Where Clause Criteria.

Merge:[WFProfileName = '|WFProfile|']

Replace

If there are no criteria, Replace clears everything first. By default, instead of merging, it clears the entire table. This will perform better for high volume because it does not try to match rows from the file to the table. An error will occur if it finds a match.

Replace Where Clause Criteria Example

This does not try to locate, it only does inserts and appends.

Replace:[EmployeeID = 'emp1']

Replace Where Clause Criteria with Substitution Variable Example

Replace:[WFProfileName = '|WFProfile|']

Next, define the Field Types and Field Names beginning in Column A Row 4 and spanning as many columns as necessary.

The column definition syntax is:

```
FieldType#:[FieldName]:StaticValue(optional):DefaultValue(optional)
```

Field Type

This relates to the column name in the table.

xfGuid

Unique identifier [SQL = uniqueidentifier]

xfText

Text defined column in the table [SQL = nvarchar, nchar, ntext]

xfInt

Short integer (4 byte integer) [SQL = int]

xfBit

0,1 (True, False) [SQL = bit]

xfDec

Decimal [SQL = Decimal (28,9)]

xfDbl

Floating point number (8 byte floating) [SQL = Float]

xfDateTime

Date [SQL = datetime]

Field Name

This is specific to the SQL table to be loaded.

StaticValue

Whatever is specified as the Static Value will override every row for that column regardless if it is blank or not.

StaticValueExample

This example will override all rows and enter 50,000 as the Static Value.

xfDec#:[Salary]:50,000

DefaultValue

This only applies to blank rows.

> **NOTE:** If something is specified in the Static Value, it will ignore whatever is in the DefaultValue.

Default Value Example

This example will enter a New Guid for all blank rows in the column.

xfGuid#:[EmployeeID]::NewGuid

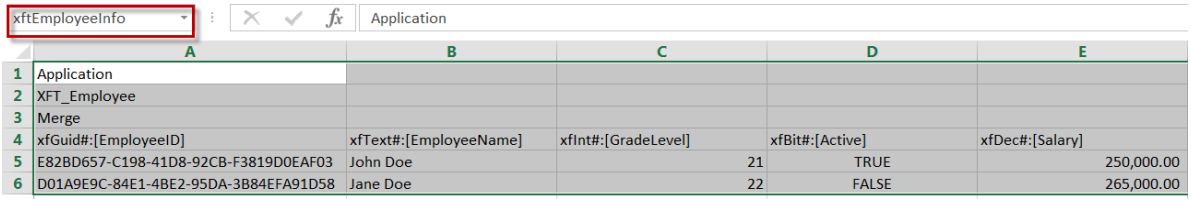
Substitution Variable Example

Substitution Variables can be used in both StaticValue and DefaultValue.

xfText#:[EmployeeName]::|Username|

	A	B	C	D	E
1	Application				
2	XFT_Employee				
3	Merge				
4	xfGuid#:[EmployeeID]	xfText#:[EmployeeName]	xfInt#:[GradeLevel]	xfBit#:[Active]	xfDec#:[Salary]
5	E82BD657-C198-41D8-92CB-F3819D0EAF03	John Doe	21	TRUE	250,000.00
6	D01A9E9C-84E1-4BE2-95DA-3B84EFA91D58	Jane Doe	22	FALSE	265,000.00

Finally, create a Named Range beginning with XFT making sure to include the entire template.



	A	B	C	D	E
1	Application				
2	XFT_Employee				
3	Merge				
4	xfGuid#:[EmployeeID]	xfText#:[EmployeeName]	xfInt#:[GradeLevel]	xfBit#:[Active]	xfDec#:[Salary]
5	E82BD657-C198-41D8-92CB-F3819D0EAF03	John Doe	21	TRUE	250,000.00
6	D01A9E9C-84E1-4BE2-95DA-3B84EFA91D58	Jane Doe	22	FALSE	265,000.00

After the template is complete, it is ready to be loaded into the custom table. If this is being used in conjunction with a OneStream Solution, refer to the Solution for further instructions on how to load the template to the table. If this is being loaded via an Extensibility Business Rule, refer to the following example.

Example:

```
Dim fieldTokens As New List(Of String)
fieldTokens.Add("xfGuid#:[EmployeeID]::NewGuid")
'fieldTokens.Add("xfGuid#:[EmployeeID]")
fieldTokens.Add("xfText#:[EmployeeName]")
'fieldTokens.Add("xfText#:[EmployeeName]::|Username|")
fieldTokens.Add("xfInt#:[GradeLevel]")
fieldTokens.Add("xfBit#:[Active]")
fieldTokens.Add("xfDec#:[Salary]")
fieldTokens.Add("xfDbl#:[VacationDays]")
fieldTokens.Add("xfDateTime#:[HireDate]")
BRApi.Utilities.LoadCustomTableUsingDelimitedFile(si, SourceDataOriginTypes.FromFileShare,
filePath,
    Nothing, ",", dbLocation, tableName, loadMethod, fieldTokens, True)
```

Importing Text-based Data

To map cell text comments from a file into OneStream, here are a few tips to get started:

1. On the Cube's Integration Tab, ensure that the TextValue field is enabled for the desired Scenario Type. This is needed to import the actual text.
2. Create a Data Source to import data for the Scenario Type.

   > **NOTE:** TextValue is one of the mappable Dimensions.

3. Hardcode the View Dimension to import to the Annotation Member, or one of the other View Dimension Comment Members such as VarianceExplanation.
4. Map the comment text to the TextValue Dimension.
5. There may not be an Amount to bring in but select a column that has a decimal value in each row and a comment and link the Data Source to that column. These numbers will come into the Stage but will not end up in the cube because they are mapped to an Annotation-type View Member.