Varigence Documentation

Cover Page sub header

Table of Contents

Software and Hardware requirements for installation components

BimlStudio

- Internet Access (required for installing and running the application)
- Same requirements as recommended for the Windows version in use (For Windows 10: https://www.microsoft.com/en-au/windows/windows-10-specifications)
- Same requirements as approved for the Visual Studio version and edition in use (For Visual Studio 2015: https://www.visualstudio.com/en-us/productinfo/vs2015-sysrequirements-vs)
- BimlStudio supports Visual Studio 2010, 2012, 2013, 2015. The version of SQL Server Data Tools (SSDT BI) and Visual Studio used depends on the SQL Server version used for the SSIS Server
- A valid Varigence BimlStudio Product key is required for use

BimlFlex Excel Plugin

- Internet Access (required for installing and running the application)
- Same minimum requirements as the Windows version in use (For Windows 10: https://www.microsoft.com/en-au/windows/windows-10-specifications)
- Same requirements as the Excel version and edition in use (For Excel 2016: https://products.office.com/en-au/office-system-requirements)
- The current plugin supports Excel 2013 and 2016, 32 or 64-bit versions
- A valid Varigence BimlFlex Product key is required for use

Databases

- Requires a supported SQL Server version
- Same requirements as the SQL Server environment it is running in (For SQL Server 2016 Enterprise Edition: https://msdn.microsoft.com/en-us/library/ms143506.aspx)
- CPU, Memory and disk space to host the BimlFlex databases as well as the DW databases

Developer installation

A developer installation is assumed to be either on a development database server with SQL Server installed or a client Windows installation (local or hosted) where the developer will use both BimlStudio, BimlFlex Excel Metadata editor, Visual Studio SSDT BI and SQL Server Management Studio to develop the Data Warehouse solution.

The developer installation required full development licensing including both BimlStudio and BimlFlex for each developer.

Considering the recurring requirement to be able to run all these applications at once it is recommended that the machine used is high-powered.

The developer also needs internet access for license key validations as well as network access to all required databases. This normally includes source systems, BimlFlex databases, Data Warehouse databases. All BimlFlex work requires access to the BimlFlex metadata database.

More information on the Developer Installation can be found here

Analyst Installation

An Analyst installation is assumed to be on a client Windows installation where the Analyst uses BimlFlex Excel Metadata editor to prepare the required metadata and source to target mappings so that the destination Data Warehouse aligns with the necessary business requirements.

It is the Analysis job to translate the system of origin schema and data into the target models such as Raw and Business Data Vaults

and presentation layer. This typically requires access to modeling and analysis tools such as SQL Server Management Studio. Some modeling can be done without access to source/destination databases, but it is recommended to maintain access to the origin and target databases for analyst work. All BimlFlex work required access to the BimlFlex metadata database.

More information on the Analyst Installation can be found here

Server installation

The base requirements for installing Microsoft SQL Server can be found here: https://docs.microsoft.com/en-us/sql/sql-server/install/hardware-and-software-requirements-for-installing-sql-server

The SSIS Server where SQL Server SSIS is used to execute SSIS packages requires the installation of the Varigence SSIS Custom Components using the Varigence Product Suite installer.

The Varigence SSIS Components can also be installed using the XCOPY scripts provide by contacting BimlFlex support.

The SQL Server where the Data Warehouse databases are deployed requires no specific installations or configurations. It is necessary that the developers and analysts have adequate access to building the Data Warehouse.

The SQL Server where the BimlFlex metadata databases are deployed requires no specific installations or configurations. It is required that the developers and analysts have adequate access to work with the metadata and target databases.

More information on the Server installation can be found here

Database requirements

BimlFlex

The BimlFlex environment requires a metadata database installed/configured and available on SQL Server.

All developers and analysts using either BimlStudio or the Excel-based metadata management plugin need access to and rights to the metadata database. These access rights can be defined through AD accounts or SQL Logins.

BimlCatalog

BimlFlex also requires an orchestration database for all orchestration, logging and auditing to be installed/configured and available on SQL Server.

For installations and configurations using SSIS with package deployment methods (such as SQL Server 2008R2) there is also a requirement for an SSIS logging table. This is integrated into the BimlCatalog database.

Developer Installation

Developer installations provide the complete developer and SSIS generation environment for building BimlFlex solutions.

Other Installation Types are:

- Analyst
- Server

Installation Media

Varigence provides two installers:

1. BimlToolsSetup_(Version).exe
2. BimlFlexSetup_(Version).exe

BimlTools Setup Installation Wizard

The BimlTools Setup installation file contains the BimlStudio application.

Run the BimlTools installation and choose what options to install.

To be able to build out 64 b packages the corresponding 64 b SSIS components needs to be available (e.g. through a local SQL Server installation). If only the 32 b SSDT BI/Visual Studio development tools are installed BimlStudio will only be able to build out in 32 b. To be able to run the 64 b BimlStudio version and still build successfully, install both BimlStudio versions and choose to build 32 b from the 64 b version.

BimlFlex Setup Installation Wizard

The BimlFlex Setup Installation file contains the BimlFlex add-ins, BimlFlex Excel Add-in and BimlFlex SSIS Custom Components.

Run the BimlFlex installation and choose what options to install.

Developers need to have both BimlStudio and the BimlFlex components installed.

SSIS Components

The Varigence Custom SSIS Components are required for building and testing the SSIS Packages for developers opening the generated SSIS packages in Visual Studio and for running the packages on an SSIS Server.

Uninstallation

If the Applications or components require uninstallation, they can use the standard "Uninstall a Program" option in Control Panel, Programs, Programs and Features.

Analyst installation

Analyst installations provide an Excel-based metadata editor environment for defining the metadata required to build BimlFlex solutions.

Other Installation Types are:

- Developer Installation
- @bimlflex-server-installation

Installation Media

Varigence provides a single BimlFlex metadata installer:

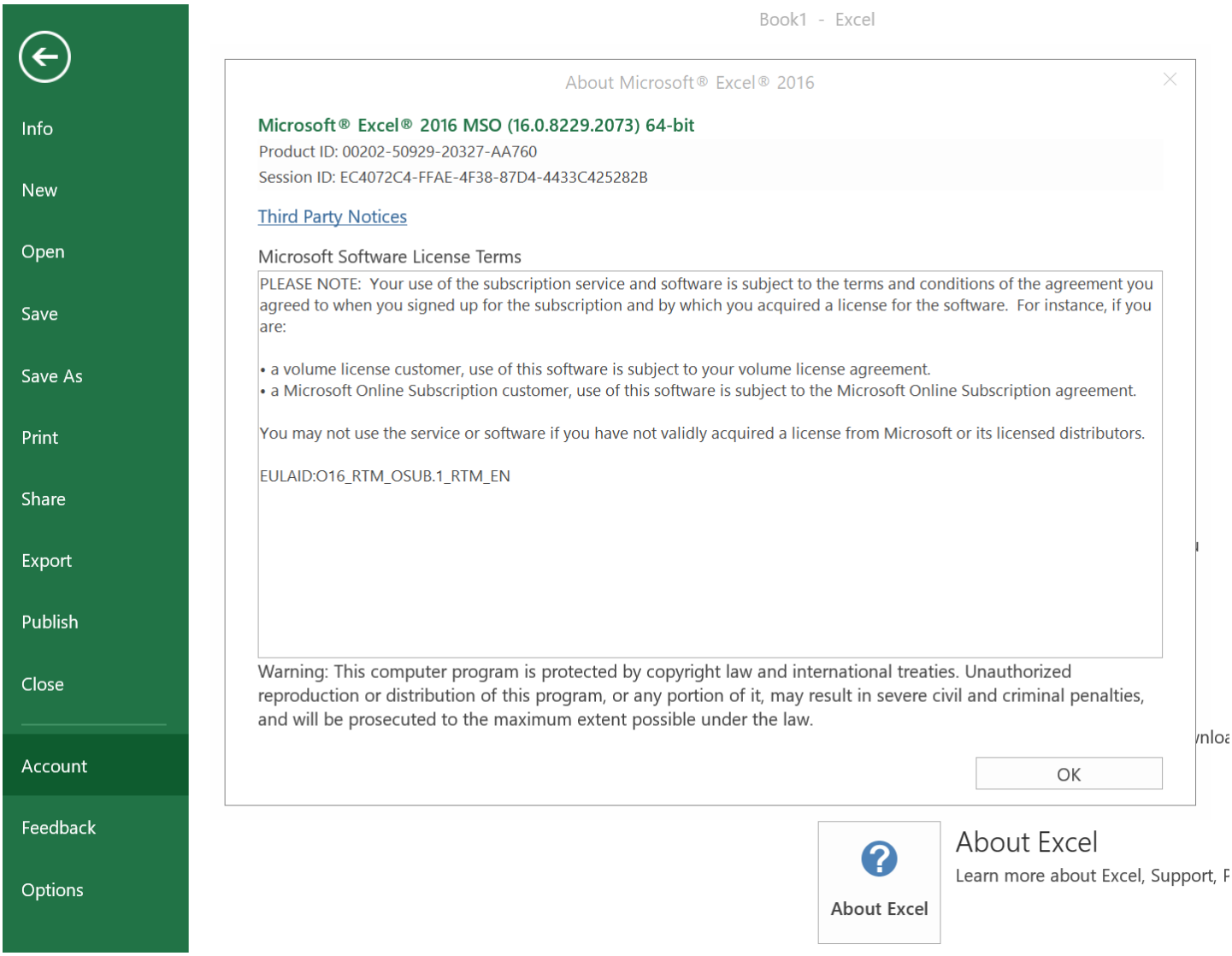
- `BimlFlexSetup_(Version).exe`

The BimlFlexSetup Installation file contains the BimlFlex Excel Add-in.

Excel Version

The BimlFlex Excel Add-in installation needs to match the installed version of Excel.

To find the installed version and its bitness, open an Excel spreadsheet, click File, Account and About Excel. The version specification includes either 32 bit or 64 bit.



For more information, review Microsoft's information here

BimlFlex Setup Installation Wizard

The BimlFlex Setup Installation file contains the BimlFlex Excel Add-in.

Run the BimlFlex installation and choose what options to install. Only a single version of the Excel Add-in matching the installed bitness of Excel should be installed.

Analysts only need to have the BimlFlex metadata functionality installed to interact with metadata through Excel. If building and BimlStudio project management is required, follow the steps in the Developer installation guide.

Uninstallation

If the Applications or components require uninstallation, they can be uninstalled from the normal "Uninstall a Program" option in Control Panel, Programs, Programs and Features.

Server Installation

For SSIS Server installations, the Varigence Custom SSIS components used in BimlFlex generated SSIS packages are required to be available on the server.

This installation is required for SQL/SSIS Servers that run BimlFlex created packages. These custom components are also required for opening and running the generated packages in Visual Studio.

Other Installation Types are:

- Developer
- Analyst

Installation Media

The Varigence BimlFlex custom SSIS Components are open source and available for download from the Varigence BimlCatalog GitHub repository

Varigence provides a Xcopy installer for the components suitable for deployments to servers. The installer is packaged in SQL server version-dependent zip.

Direct download links:

- SQL Server 2008 R2
- SQL Server 2012
- SQL Server 2014
- SQL Server 2016
- SQL Server 2017

Xcopy based deployment

Use the provided script files to deploy the Custom SSIS Components to the server.

Uninstallation

Use the provided script files to uninstall the Custom SSIS Components to the server.

Initial Setup and Configuration

The setup includes the necessary steps to set up the environments, including:

- the projects
- the databases
- the components

that are required for the full BimlFlex framework to work.

Once the BimlStudio and BimlFlex installations are completed, it is time to set up the environment for a BimlFlex project.

The development side is focused around the BimlStudio-based BimlFlex project. Open the BimlStudio Application and create a new, empty project.

The first time a new installation starts it requires the License Key. This key is provided as part of the engagement with Varigence.

If the key is missing, please contact Varigence BimlFlex Support at bimlflex-support@varigence.com.

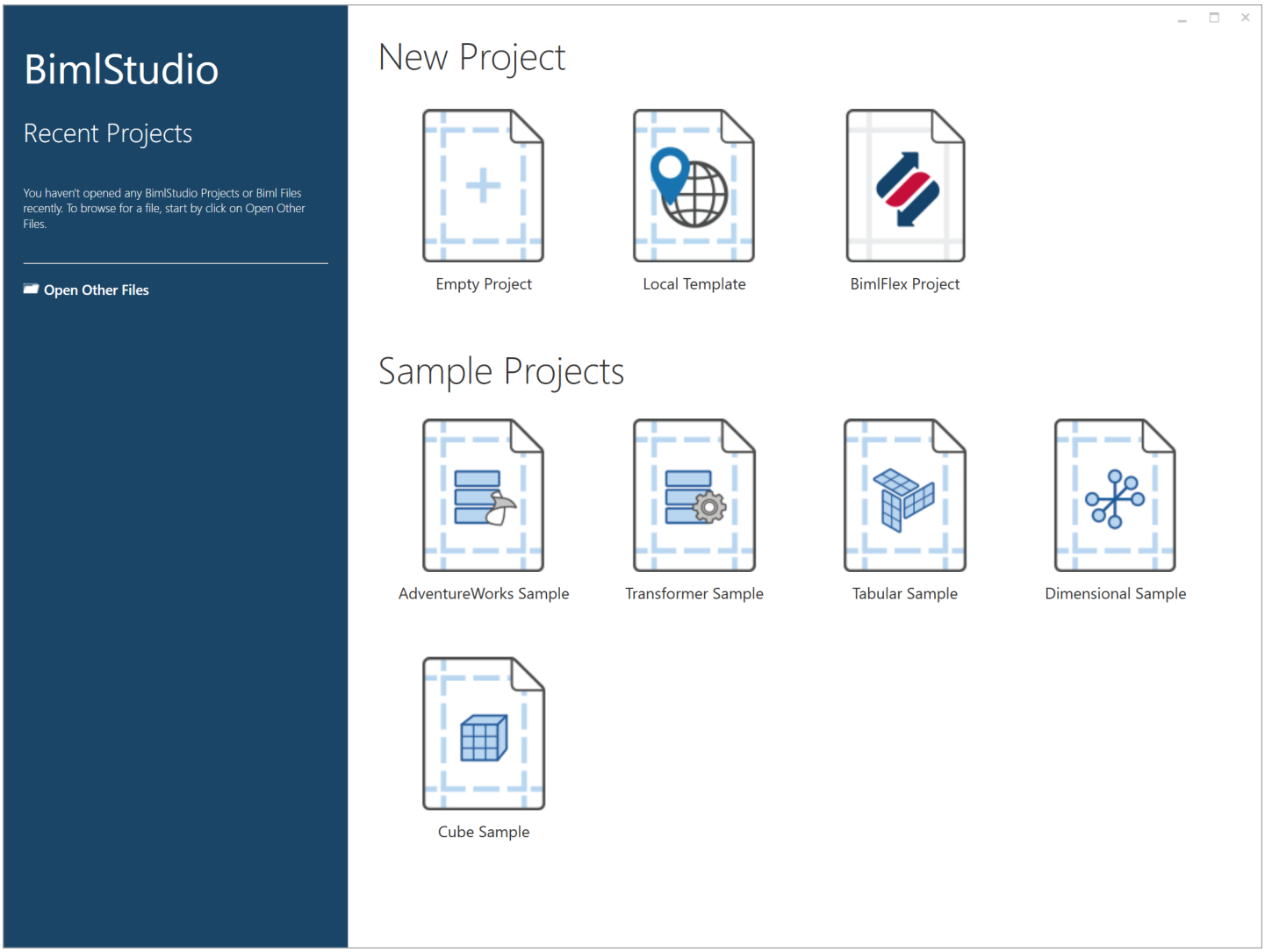
Watch Recordings

01. BimlFlex - Initial Configuration

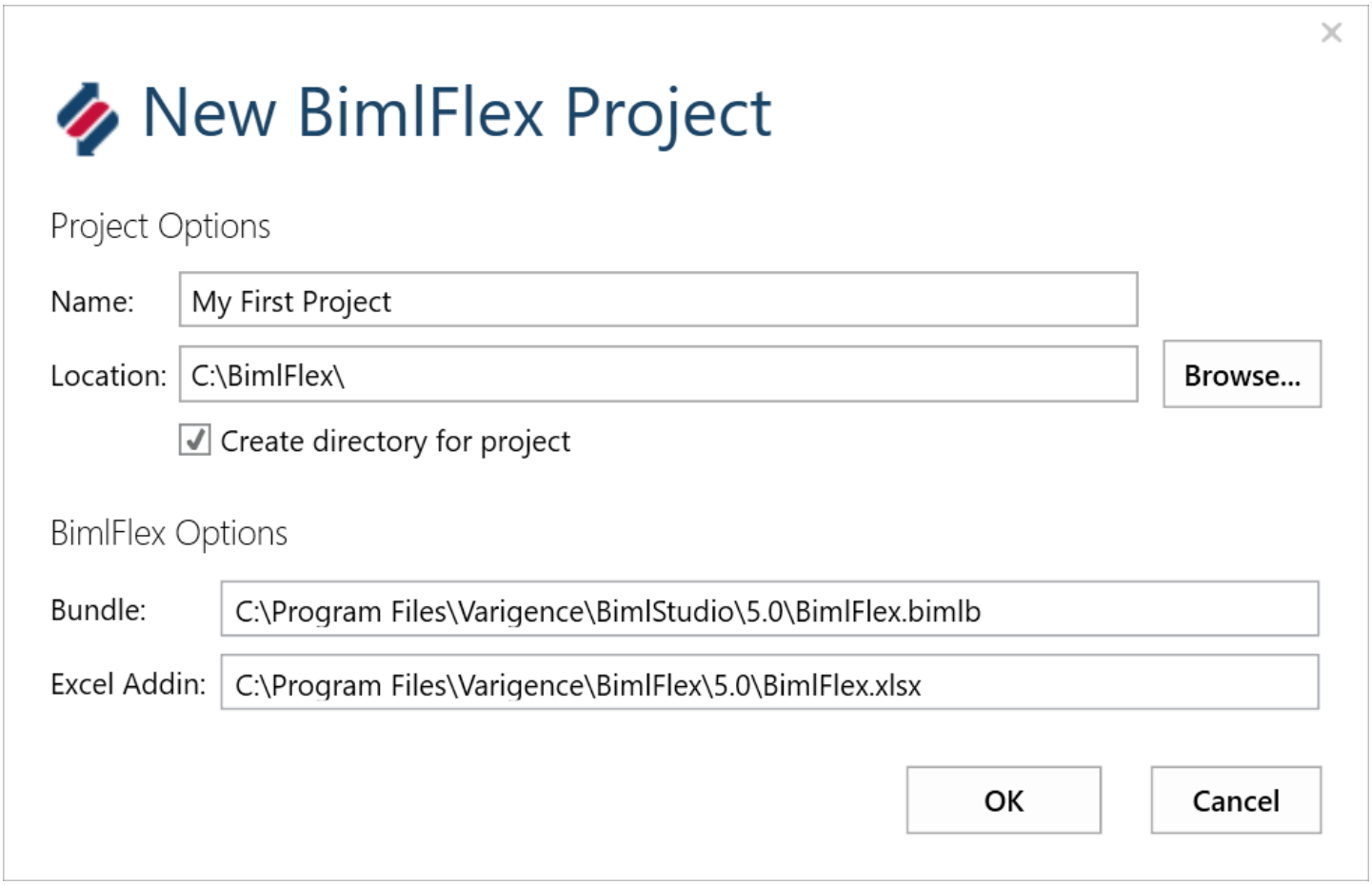
In this session, we go through the initial configuration of the BimlFlex project. We will cover the initial deployment of BimlFlex and BimlCatalog databases, create our first metadata instance.

Creating the BimlFlex project

From the start page, create a new BimlFlex project



Options for creating a new BimlFlex Project.



The project creation options include project file name and location and also the templates for the BimlFlex Bundle and the Excel Add-in file. These are stored in the default installation locations of the software:

32-bit installations

```
Bundle: C:\Program Files (x86)\Varigence\BimlStudio\5.0\BimlFlex.bimlb
Excel Add-in: C:\Program Files (x86)\Varigence\BimlFlex\5.0\BimlFlex.xlsx
```

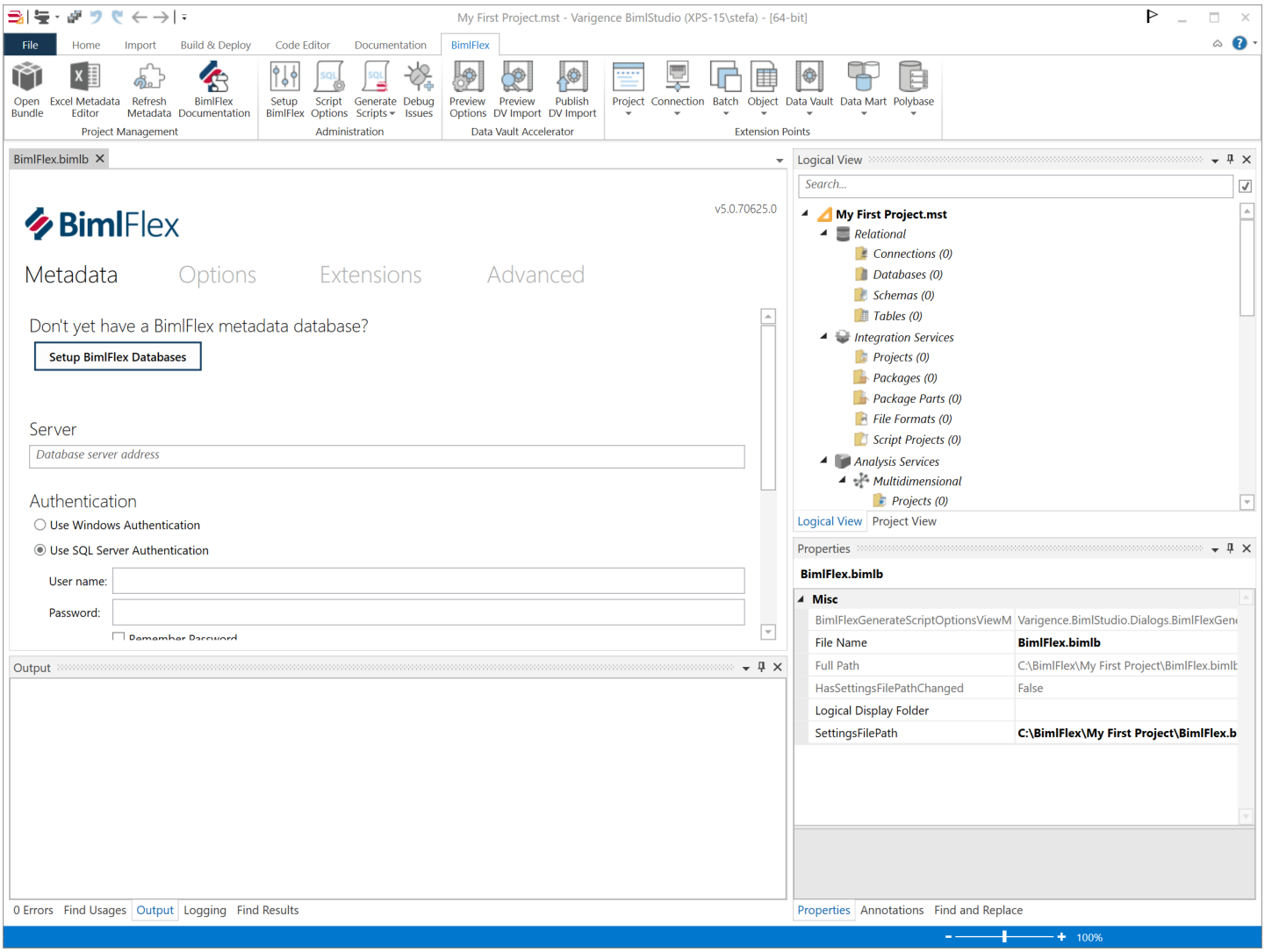
64-bit installations

```
Bundle: C:\Program Files\Varigence\BimlStudio\5.0\BimlFlex.bimlb
Excel Add-in: C:\Program Files\Varigence\BimlFlex\5.0\BimlFlex.xlsx
```

The Bundle contains the logic, Biml scripts, and code that enables the BimlFlex features in BimlStudio. The Excel Add-in is the Excel spreadsheet with the Add-in code for the Metadata editor.

The Bundle and the Excel Add-in are versioned and are copied to the destination folder with the other project files. That means different versions of the bundle and the Excel Add-in can be used depending on the project. It also means they can be updated by just exchanging these files for new versions.

Once the project is created, it is possible to open it directly from the folder or by opening the project file from BimlStudio.

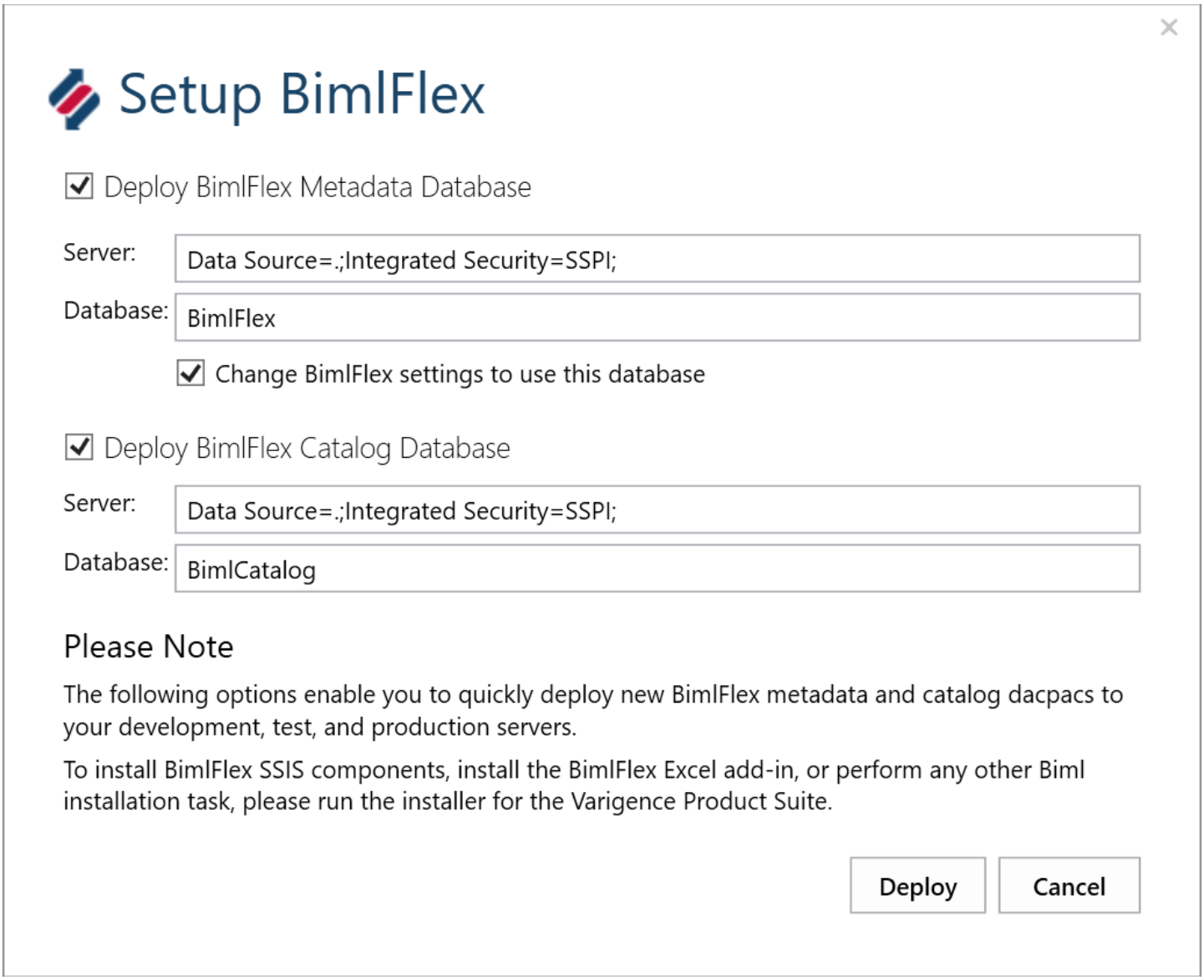


Set up Databases

The next step is to create or connect to the BimlFlex databases. BimlFlex uses a metadata database and an orchestration catalog database. All metadata generated in BimlFlex is stored in the metadata database. A metadata database can contain many projects (customers) and versions.

The orchestration catalog database contains run-time information about package executions, auditing and logging of any errors encountered.

Create the required BimlFlex databases by clicking Setup BimlFlex in the Ribbon.



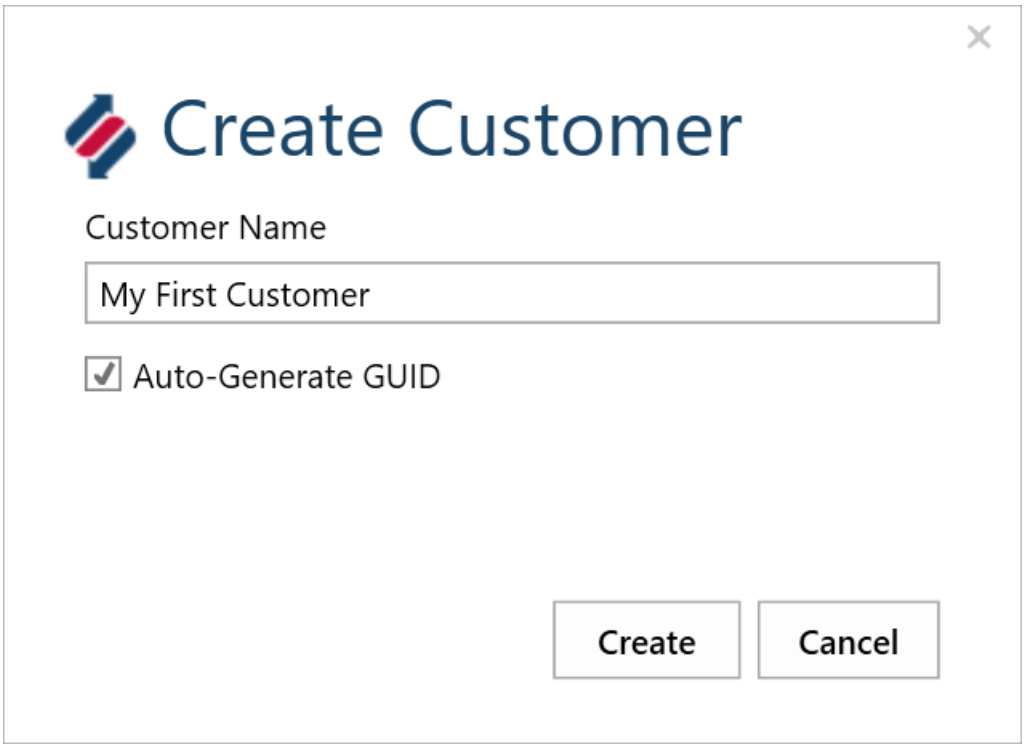
The default naming convention for the database names can be overridden to identify projects or environments.

To be able to complete this step, the account used to access the database engine needs the necessary rights to create and configure the new databases.

It is also possible to deploy the databases through the BimlFlex Support Utility Application or by implementing Dacpacs.

Setting up a Customer

Once the databases are available, it is time to create a customer. The Customer entity holds a separate set of metadata in the database and can be used to identify different projects, versions, customers, variations.



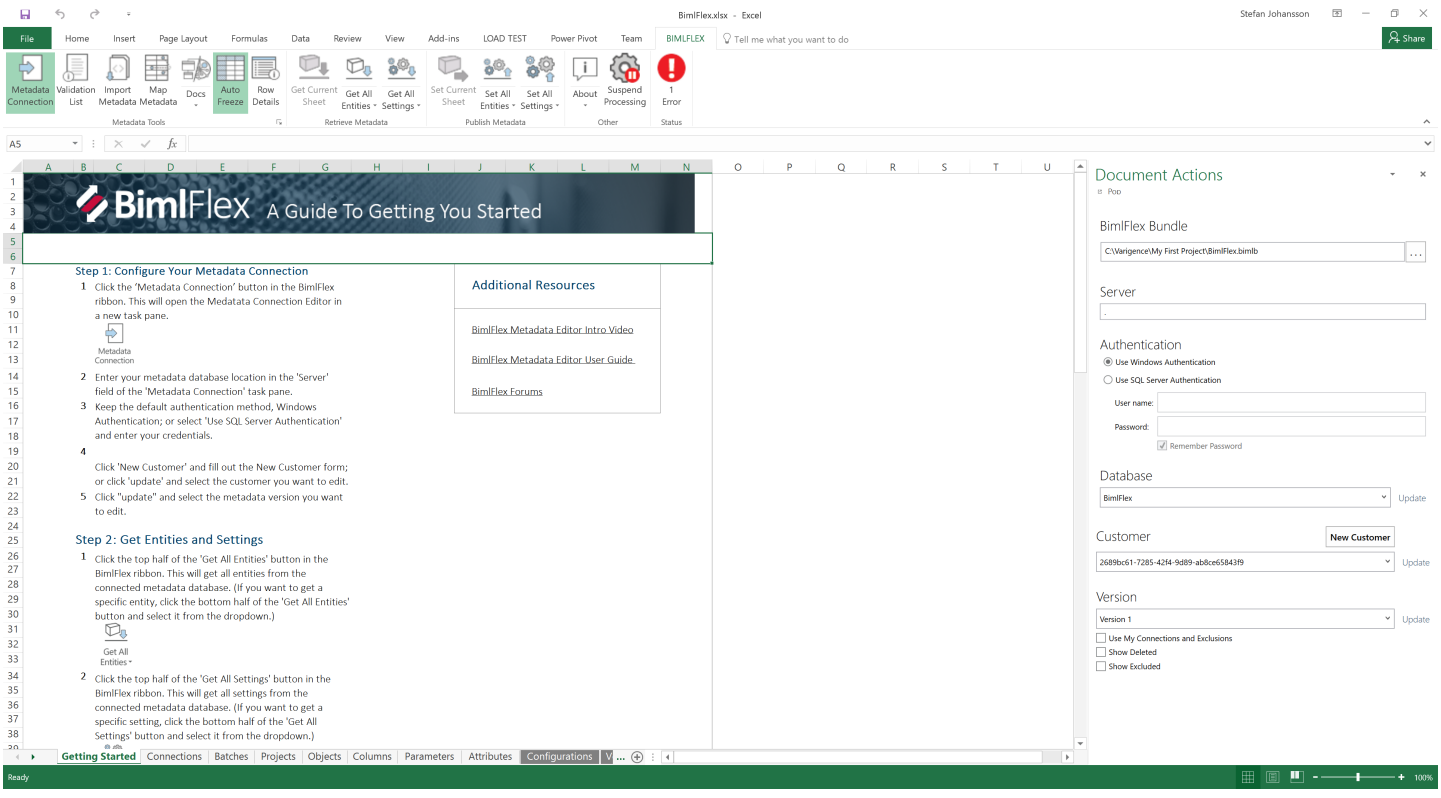
The customer is represented internally in the database with a GUID, so if metadata is reused or manually managing the data in the database, it is possible to specify it here. For typical creation, it is recommended to auto-generate it.

A Customer can have multiple versions of the related metadata. An initial Version is created with the Customer but throughout a project, it is possible to work with multiple different versions.

Once the Project, Databases, Customer, and Version are setup the Excel-based Metadata Editor can be opened to populate the Project with metadata.

Setting Up the Excel Metadata Editor

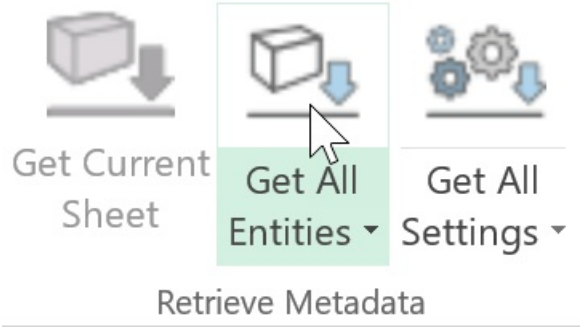
The Excel-based Metadata Editor is opened from the BimlFlex Ribbon tab in BimlStudio. The editor provides a familiar and productive metadata management environment.



The BimlFlex Excel metadata editor comes with an additional Excel tab in the Ribbon for interacting with BimlFlex. The first step is to open the metadata connection pane and connect to the BimlFlex metadata database (default name BimlFlex).

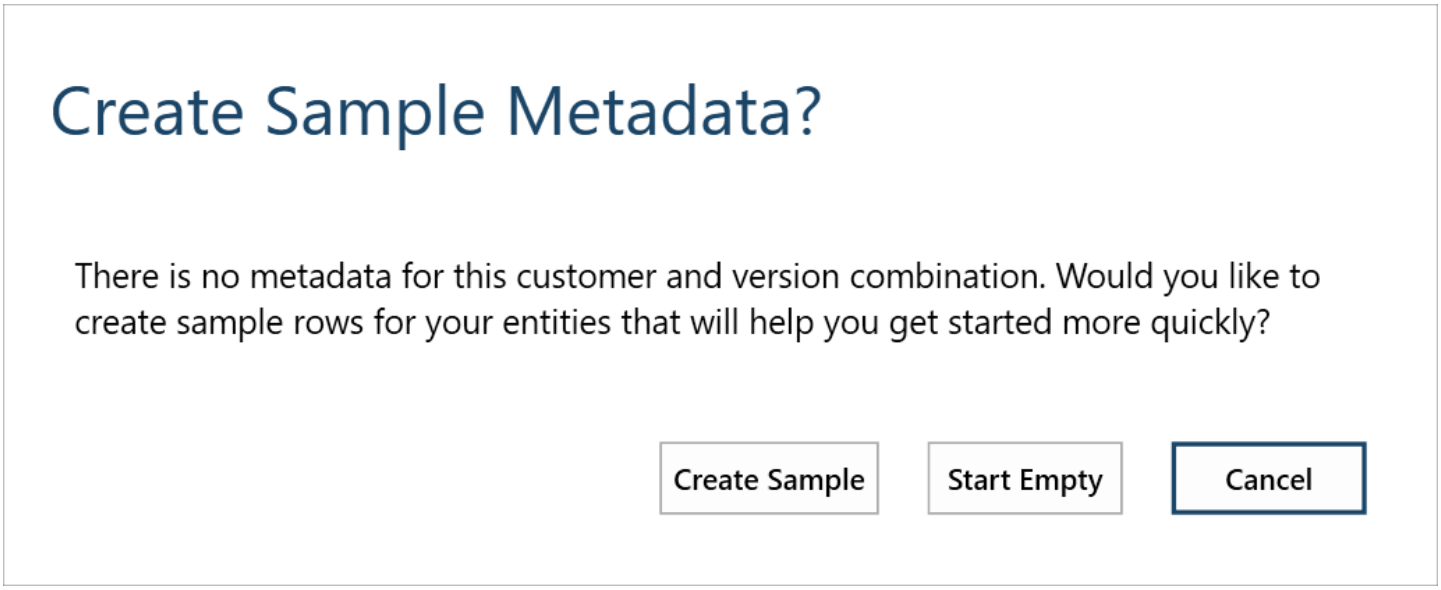
Once the connection is defined the metadata can be read into Excel. The master data is stored in the database and the working set is cached and displayed in Excel. The data is not updated in real time, so for collaborative projects, it is important to commit changes to the database and keep communicating within the team.

Retrieve all metadata into Excel by clicking the `Get All Entities` and `Get All Settings` buttons in the Excel UI.



The first time an empty customer is opened there is an option to create sample metadata.

This helps guide the first projects and provides a base set of metadata to build on.



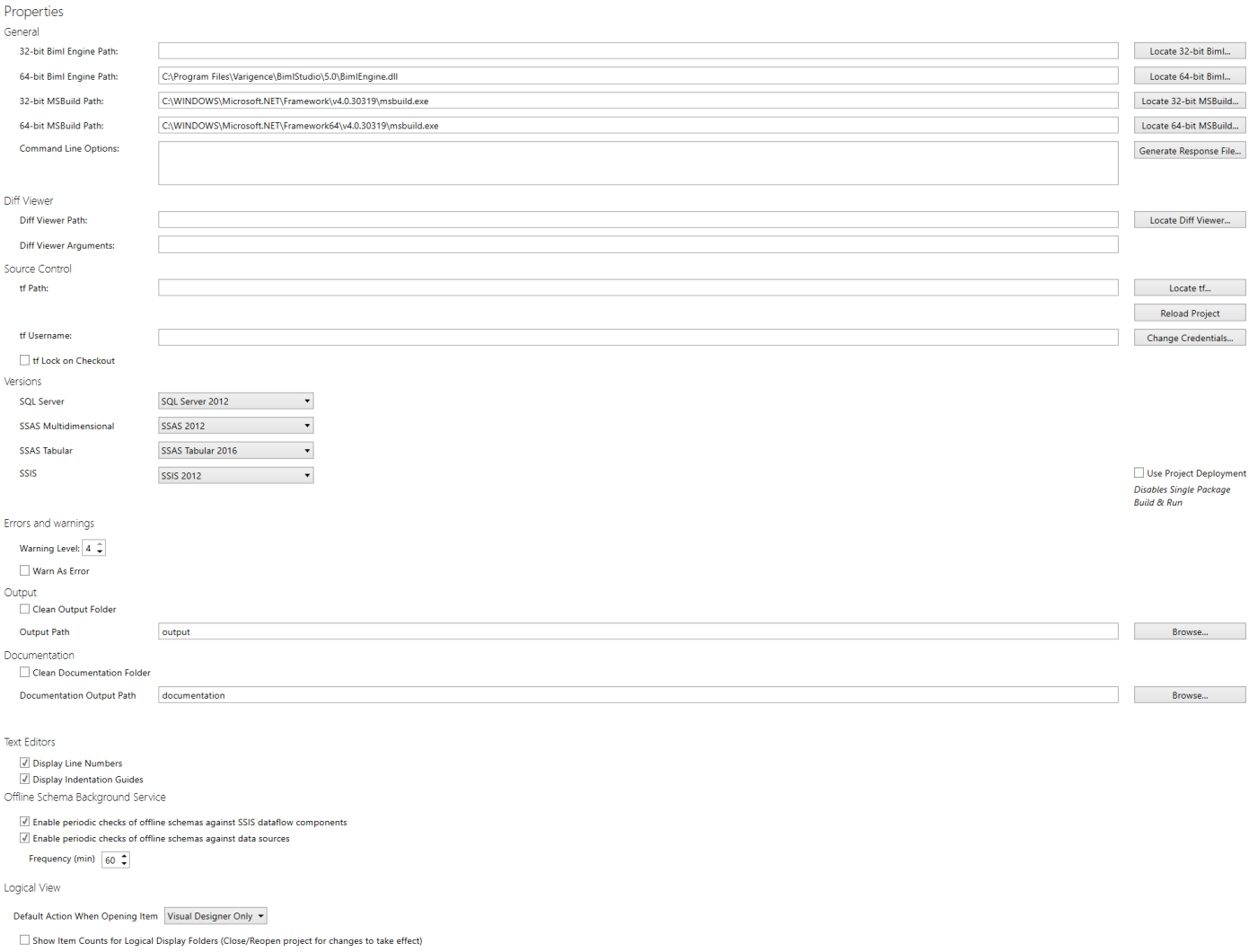
Once the metadata entities and settings are refreshed, it is possible to review the sample data and default settings.

Configuration

This step includes the required and optional configurations available for the BimlFlex framework.

BimlStudio Project configuration

BimlStudio project options include settings such as the target SQL Server and SSIS versions and project or package deployments for SSIS. Most of these are optional to change, but the target version and deployment method needs to be correct for the packages to run on the destination server.



Metadata configuration

The Configurations and Settings sheets in the Metadata Editor contains options for the generation and behaviour of the project.

This is described in more detail in:

- Configurations
- Settings

ConfigurationKey	ConfigurationValue	ConfigurationDatatype	ConfigurationDefaul	ConfigurationGrouping	ConfigurationOrde	SsisExpression	IsNullable	StagingAttribute	PersistentStagingAttribute	HubAttribute	SatelliteAttribute	LinkAttribute	DimAttribute	FactAttribute	Description	IsDeleted
RowEffectiveFromDa	FlexRowEffectiveFrom	DataType="DateTime2" Sc	0001-01-01 00:00:00.00	10	1	@[User::ParentBatchStartTime		Derived	Derived	Source	Source	Source				
RowEffectiveToDate	FlexRowEffectiveToDa	DataType="DateTime2" Sc	31/12/9999	10	2	(DT_DBTIMESTAMP2, 7)"9999-		Derived	Derived		Derived					
RowLastSeenDate	FlexRowLastSeenDate	DataType="DateTime2" Scale="7"		10	3	(DT_DBTIMESTAMP2, 7)"1900-01-01"										
RowStartDate	FlexRowStartDate	DataType="DateTime2" Scale="7"		20	1	(DT_DBTIMESTAMP2, 7)GETDATE()							Derived			
RowEndDate	FlexRowEndDate	DataType="DateTime2" Scale="7"		20	2	(DT_DBTIMESTAMP2, 7)"9999-12-31 00:00:00.000"							Derived			
RowAuditId	FlexRowAuditId	DataType="Int64"		20	3	@[User::ExecutionID]		Derived	Derived	Derived	Derived	Derived	Derived	Derived		
RowChangeType	FlexRowChangeType	DataType="AnsiString" Ler	I	20	4	(DT_STR,1,1252)"I"		Derived	Derived		Source					
RowRecordSource	FlexRowRecordSource	DataType="AnsiString" Ler	FLX	20	5	(DT_STR,10,1252)"@@this"		Derived	Derived	Source	Source	Source				
RowSourceId	FlexRowSourceId	DataType="Int32"	-1	20	6			Derived	Derived		Source					
RowIsCurrent	FlexRowIsCurrent	DataType="Boolean"	1	30	1	TRUE		Derived	Derived		Derived		Derived			
RowIsDeleted	FlexRowIsDeleted	DataType="Boolean"	1	30	2	(DATEPART("ms", Y		Derived	Derived		Source					
RowIsInferred	FlexRowIsInferred	DataType="Boolean"	1	30	3	FALSE	Y						Derived			
RowHash	FlexRowHash	DataType="AnsiString" Ler	0	40	1	[vck@@this1]+[v	Y	Hash	Hash							
RowHashKey	FlexRowHashKey	DataType="AnsiString" Ler	0	40	2	[vck@@this1]	Y	Hash	Hash							
RowHashSat	FlexRowHashSat	DataType="AnsiString" Ler	0	40	3	[@@this1]	Y				Hash					
RowHashType1	FlexRowHashType1	DataType="AnsiString" Ler	0	40	4	[vck@@this1]	Y						Hash			
RowHashType2	FlexRowHashType2	DataType="AnsiString" Ler	0	40	5	[vck@@this1]	Y						Hash			
RowLoadSequence	FlexRowLoadSequence	DataType="Int32"	0	40	6											
ExportPath	C:\Varigence\Export			90	1											
ImportPath	C:\Varigence\Import			90	2											
RootPath	C:\Varigence\BimlFlex			90	3											
ConfigurationPath	C:\Varigence\Configurations			90	4											
UseBimlCatalog	Y			90	5											
7ZipPath	C:\Program Files\7-Zip			90	6											
AzCopyPath	C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy			90	7											
KeyEndsWith	Id,Code,No,Key			90	8											
SuffixOrPrefixColum	S			90	9											
SuffixOrPrefixObject	P			90	10											
UseRecordSourceAs	N			90	11											
UseRecordSourceAs	Y			90	12											
UseColumnModelOv	N			90	13											
UseObjectModelOve	N			90	14											
HashBusinessKey	N			90	15											
ConcatenatorBusine	~			90	16											
BusinessKeyNullValu				90	17											
AppendBusinessKey	BK	DataType="AnsiString" Length="40"		91	1											
AppendSurogateKey	SK	DataType="AnsiString" Length="40"		91	2											
AppendRecordSourc	N			91	3											
AppendSchemaDm	N			91	4											
AppendSchemaRdv	N			91	5											
AppendSchemaPsa	ods			91	6											
AppendSchemaStg	N			91	7											
AppendDomain	N			91	8											
DisplayDatabaseNan	N			91	9											
DisplaySchemaName	N			91	10											
DisplayDatabaseNan	N			91	11											
LookupCachePath	C:\Varigence\Cache			91	12											
DisplaySchemaName	N			91	13											
LookupAddFilterTabl	N			91	14											
DisplayDatabaseNan	N			91	15											
LookupTablePattern	"lkp." + column.Name.MakeSsisSafe			91	16											
DisplaySchemaName	N			91	17											
DmAppendDim	DIM			92	1											
DmAppendFact	FACT			92	2											
DmAppendExternal	EXT			92	3											
DmAppendStaging	STAGE			92	4											
DvAppendBridge	BRD			93	1											
DvAppendHub	HUB			93	2											
DvAppendLink	LNK			93	3											
DvAppendLinkSatelit	LSAT			93	4											
DvAppendReference	REF			93	5											
DvAppendSatelite	SAT			93	6											
DvDefaultSchema	rdv			93	7											
DvPreviewSchema	pdv			93	8											
DvAppendPointInTim	PIT			94	1											
DvSnapshotFromDat	SnapshotFromDate	DataType="DateTime2" Sc	0001-01-01 00:00:00.00	94	2											

BimlFlex metadata database

The BimlFlex database contains the metadata required to build out data warehousing processes through BimlFlex.

> **Important**
>
> The BimlFlex database contains crucial data for the data warehouse automation process. It is important to back up, and maintain a strict disaster recovery policy for, this database.

The database is used by BimlFlex and BimlStudio to model and build the data warehouse and support the automation. Modellers, analysts and data warehouse team members use the BimlFlex modelling front end to interact with the metadata. BimlStudio access the metadata and provides the development and build experience for creating data warehouse artefacts.

Customers and Versions

All metadata is stored on a per Customer, per Version basis.

Entities

All entities are stored in the app tables within the database. These tables correspond to the data shown in the modelling front end.

Source to Target mappings

All source to target mappings are stored for columns in the database.

Archive

All changes are stored in the archive tables.

Snapshots

All metadata for a version can be stored through the snapshot feature and saved for later recall when needed.

BimlCatalog database

The BimlCatalog database contains the orchestration and run time information needed to properly load the data warehouse. It logs audit information and errors from processes and provides orchestration for batches in case failures occur mid load.

> **Important**
>
> The BimlCatalog database contains crucial data for the data warehouse auditing, logging and orchestration. It is important to back up, and maintain a strict disaster recovery policy for, this database.

The BimlCatalog is an open source project available at the BimlCatalog GitHub repository

Parameter values

Any parameter variable values are stored in the configuration tables.

Orchestration

All orchestration information is maintained within the BimlCatalog database tables.

BimlCatalog Approach

The BimlCatalog database is open and queryable and can be interacted with by the data warehousing team. It also provides an abstraction layer through Stored Procedures for interacting with the data.

Reporting Views

For reporting there are views provided that simplifies querying information about the loads

Dashboard

A Power BI Dashboard is available in the repository that displays an overview as well as more detailed information about the executions logged in the BimlCatalog database.

Upgrade and Testing Process

BimlFlex is built on BimlStudio and contains the following components that are of interest when upgrading to a later version:

- BimlStudio Application
- BimlFlex component of BimlStudio
- BimlFlex Excel Add-in
- BimlFlex Bundle file
- BimlFlex Metadata database
- BimlCatalog Orchestration database
- BimlFlex Custom SSIS Components
- The BimlStudio application is distributed as a single, separate installer
- The BimlFlex Excel Add-in is distributed as a single, separate installer
- The BimlFlex databases are embedded in the Bundle used by BimlStudio and can be deployed and updated from within BimlStudio
- The BimlFlex Custom SSIS Components are distributes as an archive file that contains the component dll and a .bat file installer

Upgrading BimlStudio and BimlFlex from the Varigence website

The current version of BimlStudio is available for download here.

Other components are available on request from the BimlFlex Enterprise support team at bimlflex-support@varigence.com. The download locations are also detailed in the licensing email.

Preparation for upgrading an existing project

Current projects with existing Extension Point script files and existing metadata needs planning and consideration before upgrading.

A new version of BimlFlex and its components can include both fixes and new functionality. This is detailed in the Release Notes that accompany the new version.

1. Do database backups

Do backups of the existing metadata and catalog databases so that they can be rolled back if needed

2. Build expanded code files

To verify that the upgrade implements new functionality as expected and that no unforeseen effects are affecting the new version, build expanded code files for Biml and SQL for all projects.

The expanded code is automatically enabled in the Bundle Options for a new project. The `Build Flat Biml File` and `Build DDL Script File` options control the creation. By creating the 2 files for each project before and after upgrading it is easy to validate all changes.

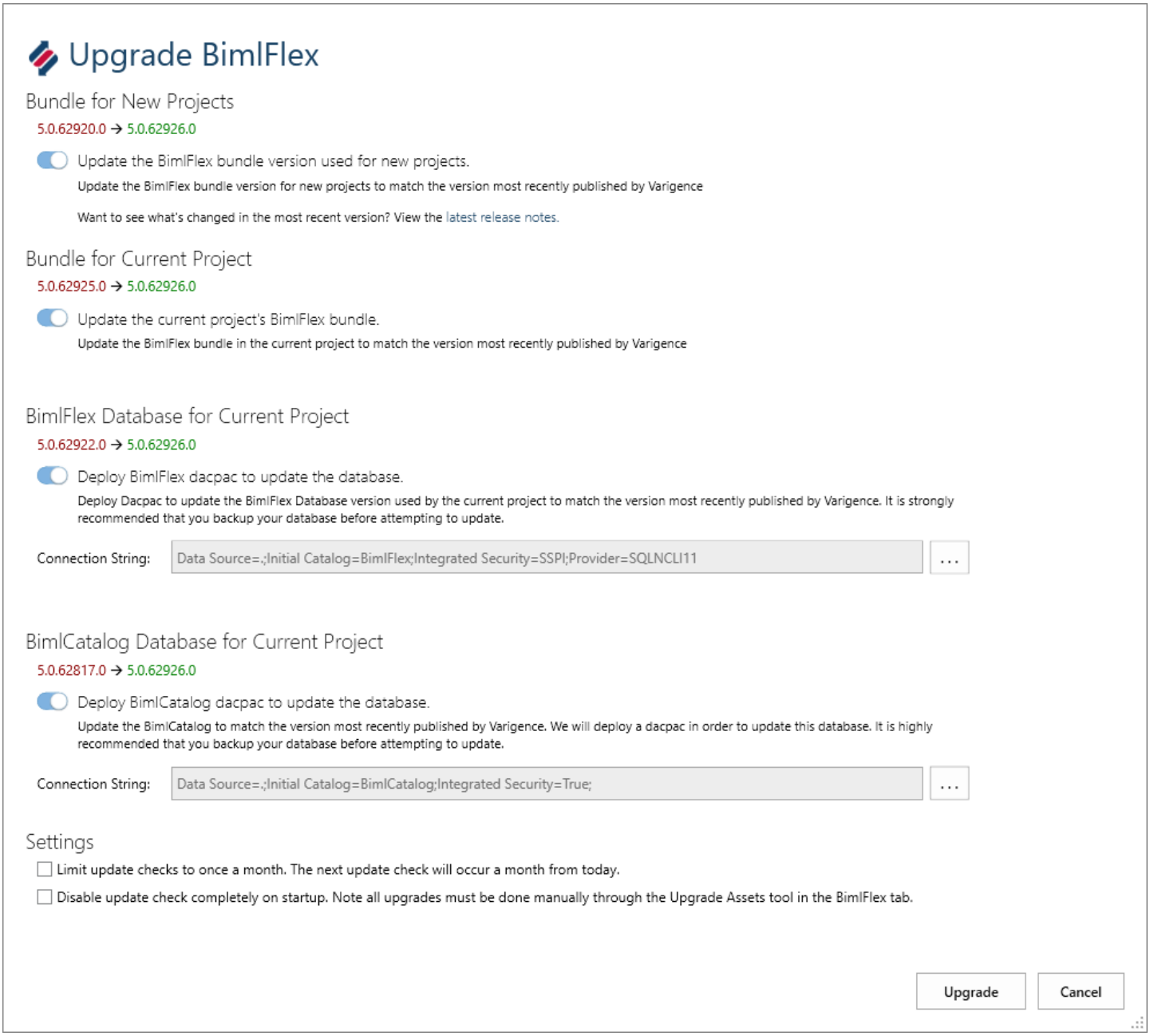
3. Backup project files

The BimlStudio project and the Custom Extension Point files and any other local file contents that have been created should be backed

up to a backup location so that a copy of the pre-upgrade project is available if needed.

4. Upgrade Bundle and Databases through BimlStudio

BimlStudio checks for updates to BimlFlex when opening a project. The Upgrade BimlFlex dialog is shown when a new version is available.



If there is a new version available online,BimlFlex will allow the new version to be downloaded and applied to both the project template location, the current project and the current databases. For new versions there is also a link to the release notes. These should be reviewed before starting an upgrade.

- Update the template Bundle file. BimlFlex uses a template file for creating new projects. BimlStudio can connect to and download the updated Bundle from the online location and make it available for new projects and as an upgrade source for other existing projects.
- Update the current project's Bundle. For a project with an older Bundle compared to either the online or the template Bundle. BimlStudio will update the project Bundle to the later version.
- Upgrade the current project's BimlFlex database. When opening a project BimlStudio will offer to upgrade that project's specific BimlFlex database. Use the Upgrade Assets dialog in the BimlFlex Ribbon tab to upgrade other database locations.

- Upgrade the current project's BimlCatalog database. When opening a project BimlStudio will offer to upgrade that project's specific BimlCatalog database. Use the Upgrade Assets dialog in the BimlFlex Ribbon tab to upgrade other database locations.

Reviewing the upgraded versions

Once the databases and Bundle file have been updated it is time to open the project in BimlStudio. Once the project has opened and downloaded, metadata applied, the BimlFlex solution is ready for testing.

1. review any errors or warnings There might have been updates to the logic applied by the Bundle. Review any error or warning messages in BimlStudio and review the release notes to see if any metadata or Extension Point behavior has changed.
2. recreate the Biml and SQL Scripts Rebuild all projects and compare the ExpandedCode files to their previous versions using a merge/file compare tool.
3. test build and run Test the build once the new project is verified as generating Biml with expected functionality. Run the new builds and validate both logs and data to verify behavior in the development and test environments before upgrading production runs.

Resolving issues

If there are unexpected behavior or issues with the upgrade, follow the guidelines in the BimlFlex Support Process to log a support ticket with the BimlFlex Support team at Varigence.

Support Process

Varigence provides support with issues encountered through using BimlFlex to build out the Data Warehouse artifacts.

For Varigence to be able to provide the best possible support for issues encountered while using BimlFlex the following process is recommended.

Creating a support case

BimlFlex support is provided through a dedicated email address: bimlflex-support@varigence.com

The BimlFlex Support team will communicate as appropriate to resolve issues. Reply to the ticket email to update the ticket.

Required information and data in a support case

For Varigence to be able to analyze and provide information for a case the included information needs to be detailed enough so that the context is clear and the steps to reproduce the scenario are available.

To be able to reproduce the issue it is also important that the metadata used for the model is provided. The metadata can be extracted from the database using the BimlFlex Support Utility Application that provides a few functions for support assistance. Details on how to access the utility are included below.

For any issues relating to Extension Points or external scripting, it is also required that the project folder with all Biml-files are zipped up and included.

Once a case is opened you will get regular updates and communications from the BimlFlex support team that will ensure your issue is timely resolved.

Accessing the Support Utility Application

More information on how to use the Utility is included in the Support Utility Application documentation.

This Utility Application includes the latest versions of the Bundle and the databases.

The Utility can be used to:

- Deploy the most recent versions of the BimlFlex and BimlCatalog databases
- Deploy the latest version of the BimlFlex bundle file
- Capture the metadata for a project to file for sending to Varigence for support issues.

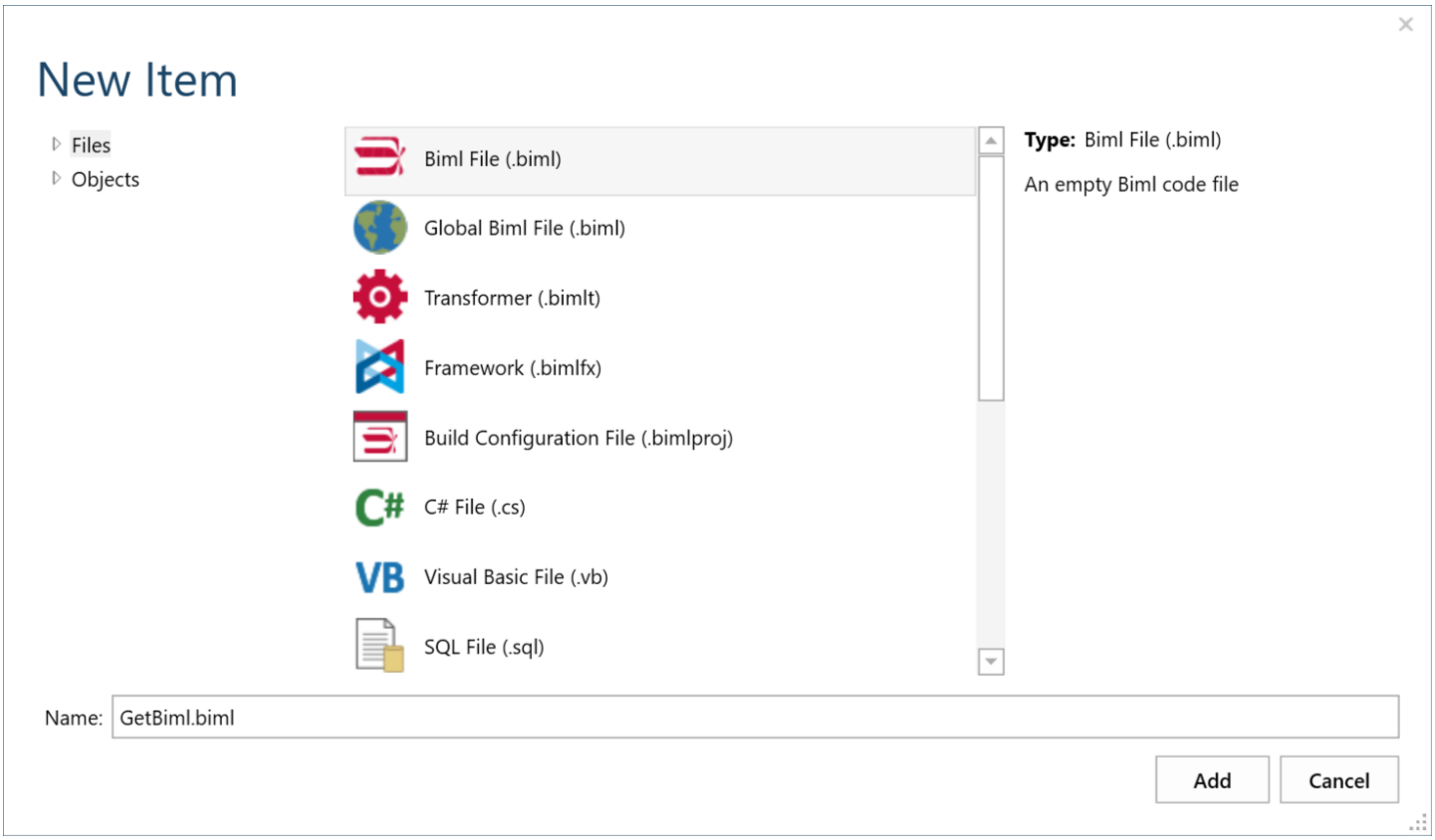
By obtaining the metadata and sending it as well as the full Biml output to BimlFlex support any issues can be analyzed and diagnosed directly.

Gathering Metadata

Use the BimlFlex Support Utility Application to connect to the BimlFlex database and save the metadata into a json text file. This data can be provided to the Support team for analysis.

Gathering project Biml

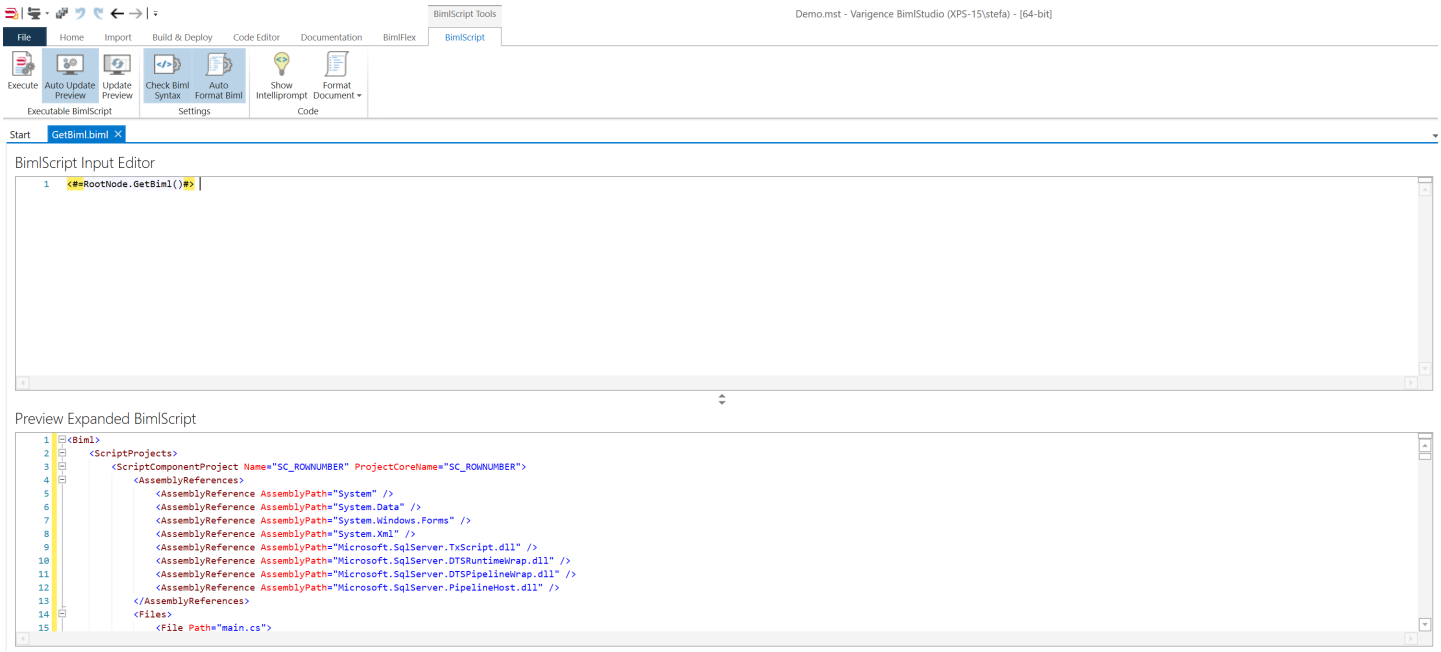
By adding a Biml Generator file to the project, the Biml from the project can be saved to a text file. This file can be provided to the Support team for analysis. This is achieved by manually creating a new Biml File in the project and add the following code to the document.



The new Biml File will be added and opened in BimlStudio.

```
<#=RootNode.GetBiml()#>
```

Adding the code and saving the file will automatically execute and provide the project Biml in the preview window.



This generated Biml is a text representation of the entire project and everything that will be generated.

Connections

TODO: Content coming soon

Batches

TODO: Content coming soon

Projects

TODO: Content coming soon

Objects

TODO: Content coming soon

Objects

TODO: Content coming soon

Attributes

The BimlFlex attributes metadata is used to control and override certain aspects on the framework.

This can be used to create Point In Time and Bridge tables, define settings overrides or specify additional metadata for entities.

TODO: Content coming soon

Metadata Columns

Project

Batch

Connection

Object

ColumnName

AttributeKey

AttributeValue

AttributeProperty

Description

IsDeleted

Predefined Attributes

AttributeKey

KEY	DESCRIPTION
IsDrivingKey	
CreateBridge	
CreatePIT	
DistributeRoundRobin	
DistributeReplicate	
RowStoreIndex	
RowCountSum	
ProtectionLevel	
SqlStartDelimiter	

KEY	DESCRIPTION
SqlEndDelimiter	
SqlIgnoreSchema	
SqlHashPattern	
SqlStringExtractPattern	
SqlStringLoadPattern	
SqlDateExtractPattern	
SqlDateLoadPattern	
SqlStringDataType	
SqlAnsiStringDataType	
SettingValue	

AttributeValue

KEY	DESCRIPTION
EncryptSensitiveWithUserKey	

AttributeProperty

KEY	DESCRIPTION
IsPrimaryHub,AddKey	
IsPrimaryHub	
AddKey	

Freehand Attributes

TODO: Coming Soon

Parameters

Parameters for load queries are added either as metadata in the Parameters sheet or via Extension Points. For high watermark delta loads and similar simple parameters adding the Parameter to the metadata will generate and include all required logic to the load process.

Extension Points can be added for more complex parameters that require custom logic.

Parameters added to a project, batch package or regular package can be used as any other SSIS parameter. BimlFlex applies standard practices for using parameters but supports any custom use of added parameters.

Basic Parameters

The most common scenario is to add parameters to a source query or add parameters that hold details about the particular BI solution. The approach used for these scenarios are different.

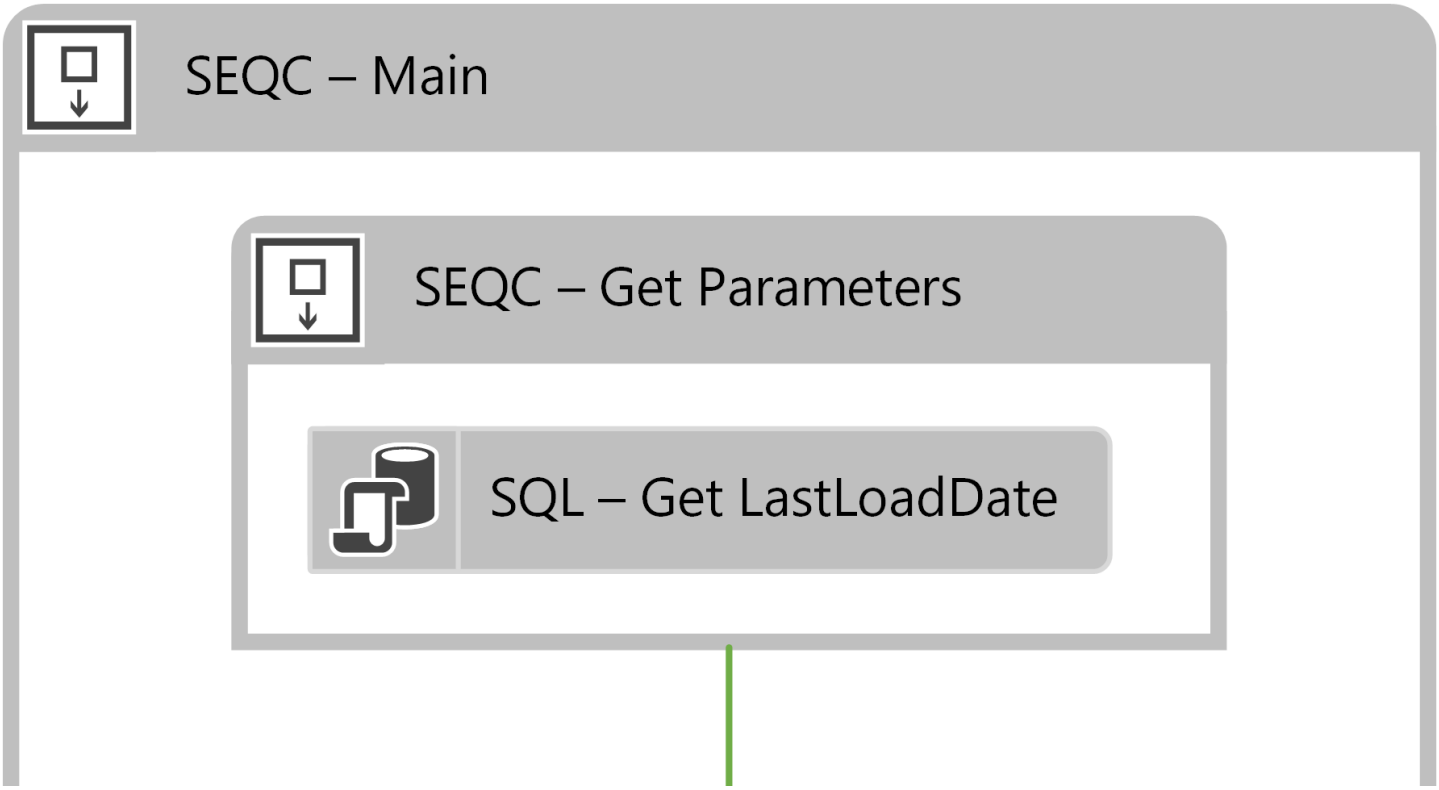
When the parameter metadata is entered into BimlFlex Excel Add-in, BimlFlex uses the `[BimlCatalog]` database to manage parameterized source queries. Parameters are persisted in `[BimlCatalog].[ssis].[ConfigVariable]` and logged to `[BimlCatalog].[ssis].[AuditLog]`.

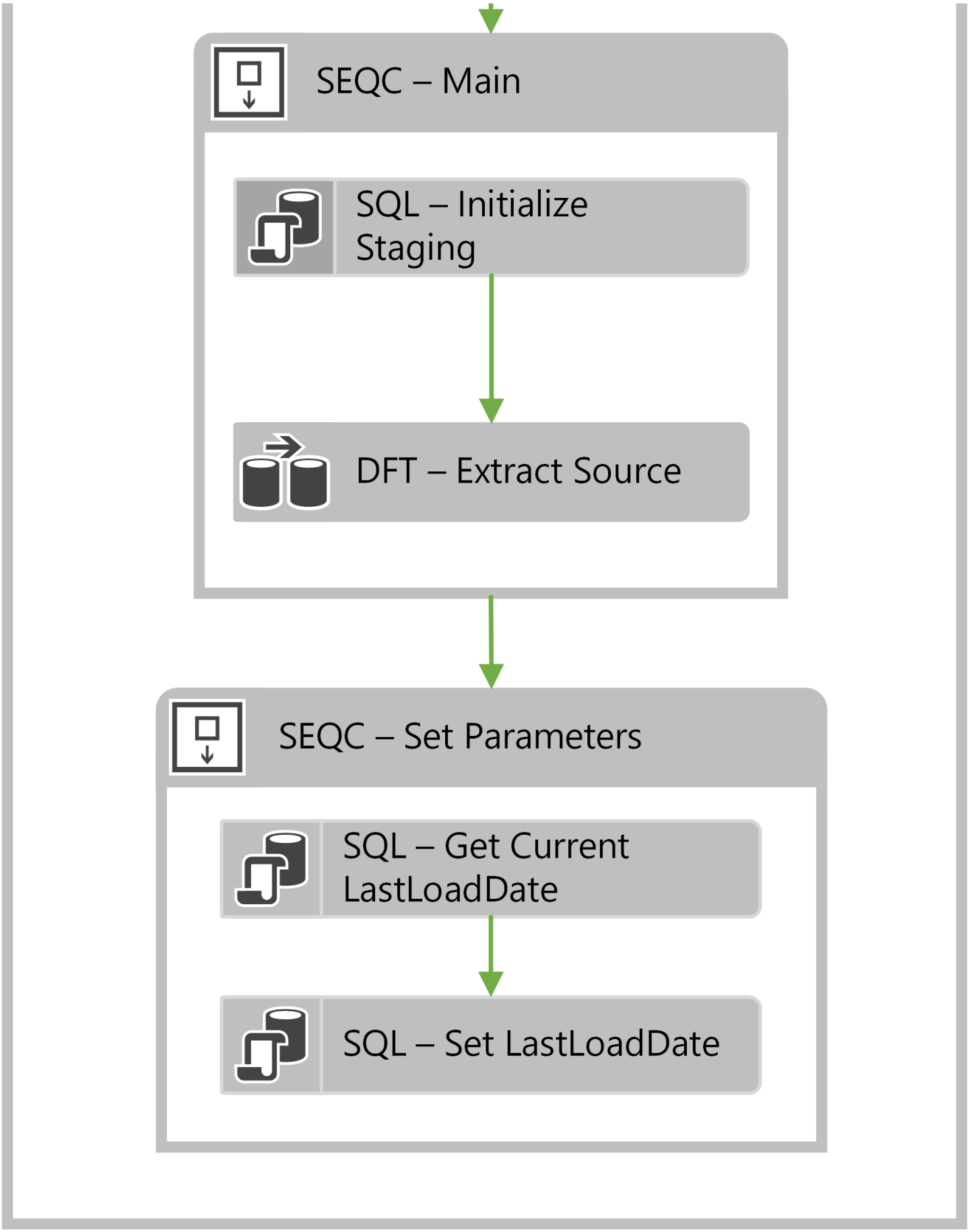
Here is an example of a parameter being stored in the `BimlCatalog` database in the `[ssis].[ConfigVariable]` table.

SYSTEMNAME	OBJECTNAME	VARIABLENAME	VARIABLEVALUE	EXECUTIONID
AW_SRC	AW_SRC.Sales.SalesPerson.ModifiedDate	LastLoadDate	20/09/2015	109

Parameters in BimlFlex Excel

When a parameter is added to an object in BimlFlex Excel Add-in, the metadata will be managed the BimlFlex framework and used to generate all the required components to track and update the value defined. Below demonstrates how a data parameter has been incorporated into a source to target loading package. Note that this load starts and ends with sequence containers that are specifically for managing this new parameter.





Using the above process, a package that tracks the maximum ID as a parameter can be rerun any number of times. Each time BimlFlex will store the maximum value and update it as it increases over time.

The process starts before the main container. The first control flow task looks up any relevant parameters. LookupParameter is used to check the catalog table `[BimlCatalog].[ssis].[ConfigVariable]` for the parameter that matches the current object being loaded.

The parameter value will be injected in to the source query in the normal data flow as a `WHERE` clause using the specified logic from the metadata.

After the load the Set Parameters tasks will get the new value for the parameter and update the `[BimlCatalog].[ssis].[ConfigVariable]` table.

Parameters in Extension Points

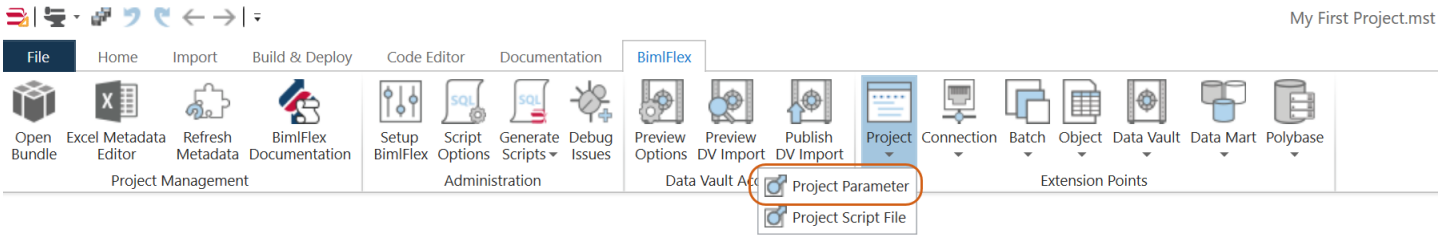
Extension Point-based parameters are available for more complex scenarios where the logic needed for the flow and parameter isn't easily injected through the normal parameter process.

It also supports specifying Project level Parameters that are commonly available in all packages in the project.

Extension Point Project Parameters

Add Extension Points in BimlStudio.

More details on Extension Points are in the Extension Points documentation



The newly created file contains some sample scripts:

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="ProjectParameter" target="ProjectName" #>
<!-- You can find more details on the Varigence website.
https://www.varigence.com/Documentation/Language/Element/AstParameterNode -->
<!-- The below example configures parameters that will be used in combination with Connection Expressions-->
<Parameter Name="ServerName" DataType="String" IsRequired="true">localhost</Parameter>
<Parameter Name="UserName" DataType="String" IsRequired="true">varigence</Parameter>
<Parameter Name="UserPassword" DataType="String" IsRequired="true">P@ssw0rd!</Parameter>
```

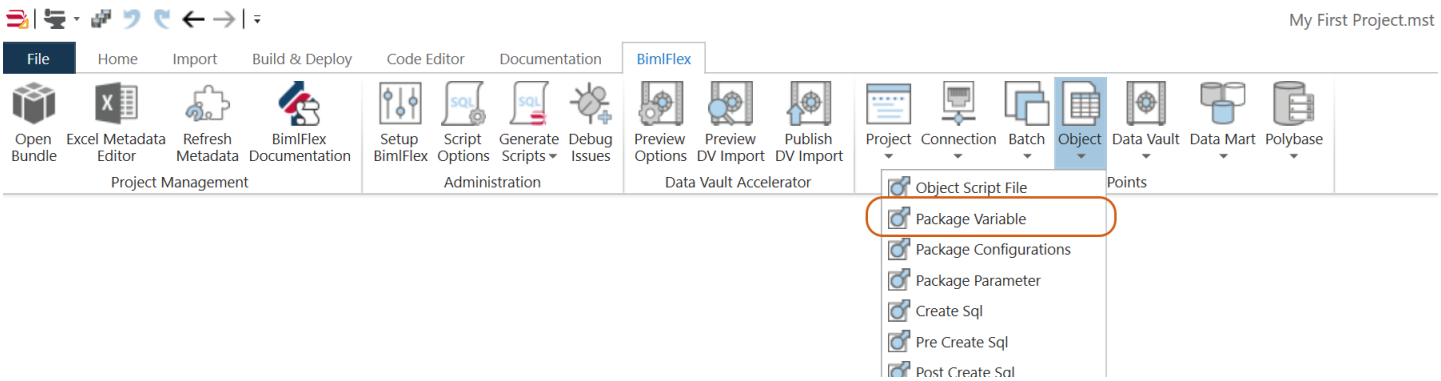
The directives are required for the Extension Point. An additional attribute for the target of the Extension Point needs to be specified. For a project parameter it is the name of the project it targets.

Once the Project Parameter is defined through the Extension Point it can be reused throughout the project in either metadata or additional Extension Points as needed.

Extension Point Package Parameters

Package Parameters only affect the individual package it targets. Package parameters can be used when a single package requires bespoke logic that doesnt fit the existing parameter logic. These parameters can be used for any logic and might not need to be persisted in the BimlCatalog database.

Add Extension Points in BimlStudio



The newly created file contains some sample scripts:

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="PackageVariable" target="ObjectName"#>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper" #>
<!-- You can find more details on the Varigence website.
https://www.varigence.com/Documentation/Language/Element/AstVariableNode -->
<!-- Variables can also be added to all the packages for the Batch using the PackageVariable combined with
CustomOutput.ObjectInherit = true; -->
<#* CustomOutput.ObjectInherit = true; *#>
<Variable Name="TenantCode" DataType="String">UNK</Variable>
<Variable Name="CurrentModifiedDate" DataType="String" Namespace="User">1900-01-01</Variable>
```

The directives are required for the Extension Point. An additional attribute for the target of the Extension Point needs to be specified. For a package parameter it is the name of the package it targets.

Once the Package Parameter is defined through the Extension Point it can be reused throughout the Package in either metadata or additional Extension Points as needed.

Metadata and framework configurations in BimlFlex

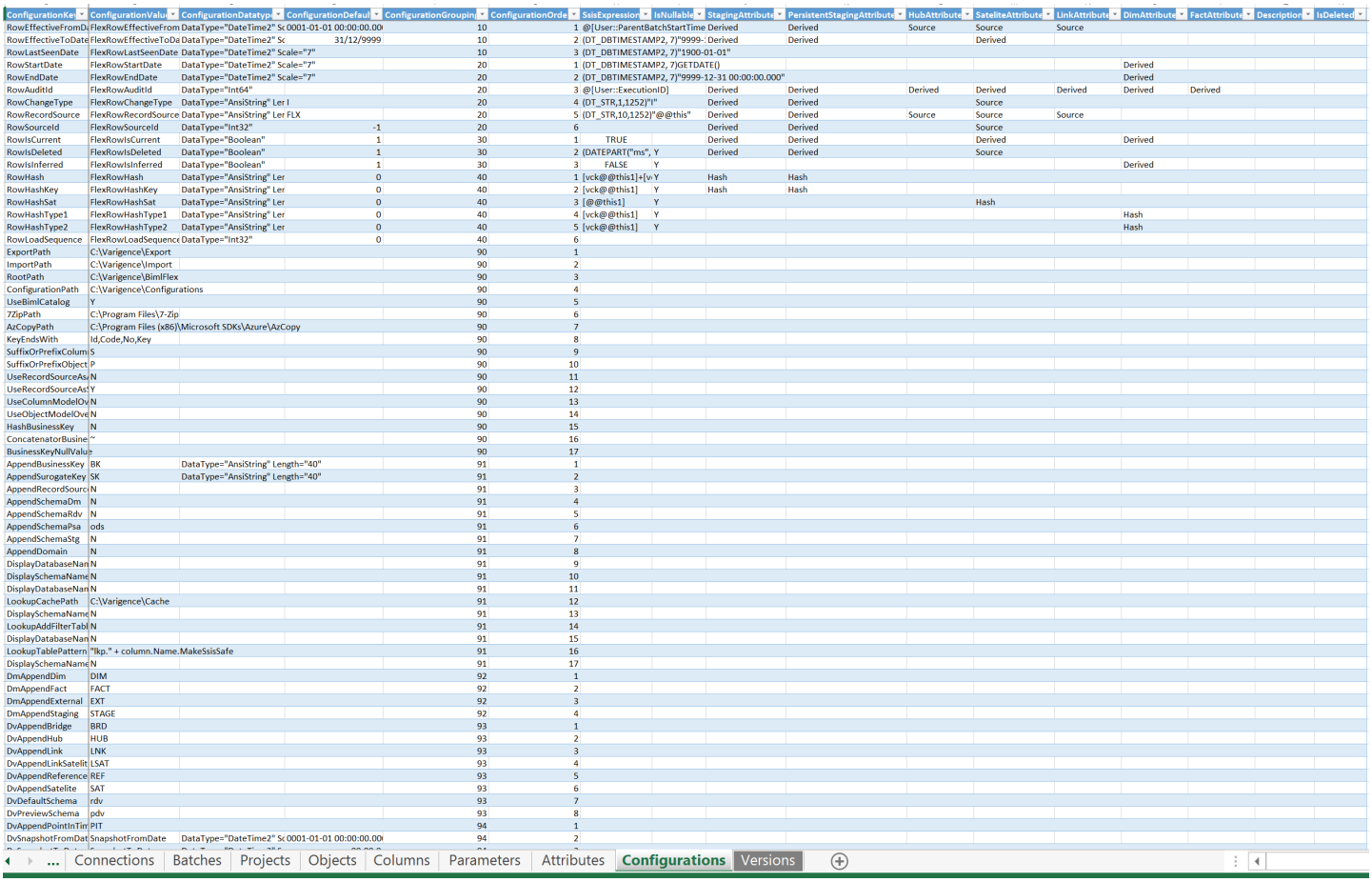
This document outlines the metadata and framework configurations available in BimlFlex.

These configurations drive the behavior of the BimlFlex product.

By changing them the produced artifacts can adapt to support requirements for file locations, naming conventions, data conventions etc.

The Configuration defaults are the Varigence recommended values and there is no need to change or configure unless there is a requirement to change specific behaviors. Align these settings with the organizations best practices and environmental requirements.

The Configurations are available in the Configurations sheet in the BimlFlex Excel Add-in.



Metadata column overview

KEY	VALUE
Configuration Key	the Configuration Key, the internal key BimlFlex refers to, cannot be changed
Configuration Value	the Configured Value, can be updated to support a different design pattern or behaviour
Configuration Datatype	the data type the configuration Value uses. Needs to be a valid data type definition
Configuration Default	the Configuration Key's Default Value

KEY	VALUE
Configuration Grouping	BimlFlex Internal Grouping of configurations
Configuration Order	BimlFlex Internal Ordering of configurations
SSIS Expression	the SSIS Expression used to derive the value. Needs to be a valid SSIS Expression. Uses the shorthand @@this to define the current entity
Is Nullable	Defines If the attribute is nullable Valid Enumeration {Empty, Y, N}
Staging Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Persistent Staging Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Hub Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Satellite Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Link Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Dim Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Fact Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Description	An optional description for custom attributes or definitions. The default configurations are described in this document
Is Deleted	Flag to set if a custom attribute has been entered but is now no longer needed and should be considered deleted. Valid Enumeration {Empty, Y, N}

Standard configurations

RowEffectiveFromDate

Description

The RowEffectiveFromDate defines the start of time for timelines in the data warehouse. The `DateTime2(7)` datatype supports a wider range of dates than traditional DateTime meaning starting on 1 Jan 0001 will support most use cases. If a specific start DateTime for timelines is needed, such as to support an existing design template, this can be updated to support it. Unless there is a good reason to change it, it is recommended to keep the default.

The SSIS Expression is used in the Staging and Persistent Staging loads to derive the RowEffectiveFromDate from the `@[User::ParentBatchStartTime]` variable. This is inserted into the Data Flow to give each row its effectiveness date. The default configuration is to use the start date time of the parent Batch for this column.

An example of implementation usage for the configuration default is in the Default constraint of this table creation script for the Address staging table from the AdventureWorks LT Source:

```
IF NOT EXISTS (SELECT * FROM sys.default_constraints WHERE [parent_object_id] = OBJECT_ID(N'[AWLT].
[Address]') AND [name] = 'DF_Address_FlexRowEffectiveFromDate')

ALTER TABLE [AWLT].[Address] ADD CONSTRAINT [DF_Address_FlexRowEffectiveFromDate] DEFAULT ('0001-01-01
00:00:00.000') FOR [FlexRowEffectiveFromDate]
```

Example

See defaults

Valid Value

Depends on configuration attribute

Default Metadata information

KEY	VALUE
Configuration Key	RowEffectiveFromDate
Configuration Value	FlexRowEffectiveFromDate
Configuration Datatype	DataType="DateTime2" Scale="7"
Configuration Default	0001-01-01 00:00:00.000
Ssis Expression	@[User::ParentBatchStartTime]
Staging Attribute	Derived
Persistent Staging Attribute	Derived
Hub Attribute	Source
Satellite Attribute	Source
Link Attribute	Source

RowEffectiveToDate

Description

The **RowEffectiveToDate** defines the end of time for timelines in the data warehouse. The DateTime2(7) datatype supports a wider range of dates than traditional DateTime meaning ending on 31 Dec 9999 will support most use cases. If a specific start DateTime for timelines is needed, such as to support an existing design template, this can be updated to support it. Unless there is a very good reason to change it, it is recommended to keep the default.

The SSIS Expression is used in the Staging and Persistent Staging loads to derive the RowEffectiveToDate from the `(DT_DBTIMESTAMP2, 7)"9999-12-31 00:00:00.000"` expression. This is inserted into the Data Flow to give each row its end date. The default configuration is to use the end of timeline definition.

An example of implementation usage for the configuration default is in the Default constraint of this table creation script for the Address staging table from the AdventureWorks LT Source:

```
IF NOT EXISTS (SELECT * FROM sys.default_constraints WHERE [parent_object_id] = OBJECT_ID(N'[AWLT].
[Address]') AND [name] = 'DF_Address_FlexRowEffectiveToDate')

ALTER TABLE [AWLT].[Address] ADD CONSTRAINT [DF_Address_FlexRowEffectiveToDate] DEFAULT ('9999-12-31') FOR
[FlexRowEffectiveToDate]
```

Example

See defaults

Valid Value

Depends on configuration attribute

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowEffectiveToDate
Configuration Datatype	DataType="DateTime2" Scale="7"
Configuration Default	9999-12-31
Ssis Expression	`(DT_DBTIMESTAMP2, 7)"9999-12-31"`
Staging Attribute	Derived
Persistent Staging Attribute	Derived
Satellite Attribute	Derived

RowLastSeenDate

Description

The **RowLastSeenDate** defines the default SSIS Expression used to derive the RowLastSeen attribute

Example

```
(DT_DBTIMESTAMP2, 7)"1900-01-01"
```

Valid Value

A valid SSIS expression and data type.

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowLastSeenDate
Configuration Datatype	DataType="DateTime2" Scale="7"
Ssis Expression	`(DT_DBTIMESTAMP2, 7)"1900-01-01"`

RowStartDate

Description

The **RowStartDate** defines the start of time definition for a row in the data warehouse. This attribute is used to define the timeline in use, from start of time to end of time.

Example

```
(DT_DBTIMESTAMP2, 7)GETDATE()
```

Valid Value

A valid SQL, SSIS Expression and data type that defines a date time

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowStartDate
Configuration Datatype	DataType="DateTime2" Scale="7"
Configuration Default	1900-01-01
Ssis Expression	`(DT_DBTIMESTAMP2, 7)GETDATE()`
Dim Attribute	Derived

RowEndDate

Description

The **RowEndDate** defines the end of time definition for a row in the data warehouse. This attribute is used to define the timeline in use, from start of time to end of time

Example

```
(DT_DBTIMESTAMP2, 7)"9999-12-31 00:00:00.000"
```

Valid Value

A valid SQL, SSIS Expression and data type that defines a date time

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowEndDate
Configuration Datatype	DataType="DateTime2" Scale="7"
Ssis Expression	`(DT_DBTIMESTAMP2, 7)"9999-12-31 00:00:00.000"`
Satellite Attribute	Derived

RowAuditId

Description

The **RowAuditId** defines the derivation pattern for the audit value for a row. The default derives the audit id from the ExecutionID user variable. This value will be added to all rows as the audit id, default column name: `[FlexRowAuditId]`

Example

```
@[User::ExecutionID]
```

Valid Value

A valid SSIS Expression and data type, attributes as per their enumerations

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowAuditId
Configuration Datatype	DataType="Int64"
Ssis Expression	`@[User::ExecutionID]`
Staging Attribute	Derived
Persistent Staging Attribute	Derived
Hub Attribute	Derived
Satellite Attribute	Derived
Link Attribute	Derived
Dim Attribute	Derived
Fact Attribute	Derived

RowChangeType

Description

The **RowChangeType** defines the string representation of the change type when inserting new rows into the data warehouse.

Example

```
I
```

Valid Value

A Valid SSIS Expression

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowChangeType
Configuration Datatype	DataType="AnsiString" Length="1"
Configuration Default	I
Ssis Expression	`(DT_STR,10,1252)"I"`
Staging Attribute	Derived
Persistent Staging Attribute	Derived
Satellite Attribute	Source

RowRecordSource

Description

The RowRecordSource defines the record source for the data. This is a required attribute for Data Vault sources and normally defined in the connections definition for external sources loaded into the Data Vault

Example

```
(DT_STR,10,1252)"@@this"
```

Valid Value

A valid SSIS Expression

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowRecordSource
Configuration Datatype	DataType="AnsiString" Length="10"
Configuration Default	FLX
Ssis Expression	`(DT_STR,10,1252)"@@this"`
Staging Attribute	Derived
Persistent Staging Attribute	Derived
Hub Attribute	Source
Satellite Attribute	Source
Link Attribute	Source

RowSourceId

Description

The **RowSourceId** defines the sequence number of the data row within the set. This is used to identify all rows in order within a batch.

Example

```
-1
```

Valid Value

A valid SSIS Expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowSourceId
Configuration Datatype	DataType="Int32"
Configuration Default	-1
Ssis Expression	true
Staging Attribute	Derived
Persistent Staging Attribute	Derived
Satellite Attribute	Source

RowIsCurrent

Description

The **RowIsCurrent** defines the current row flag for timelined data, such as for satellites. The RowIsCurrent is the definition for how the current row is defined in the data.

Example

```
true
```

Valid Value

A valid SSIS Expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowIsCurrent
Configuration Datatype	DataType="Boolean"

KEY	VALUE
Configuration Default	1
Ssis Expression	true
Staging Attribute	Derived
Persistent Staging Attribute	Derived
Satellite Attribute	Derived
Dim Attribute	Derived

RowIsDeleted

Description

The RowIsDeleted defines the derivation pattern to if a row is deleted or not

Example

```
false
```

Valid Value

A valid SSIS expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowIsDeleted
Configuration Datatype	DataType="Boolean"
Configuration Default	`(DATEPART("ms", GETDATE())%2)==1?TRUE:FALSE`
Ssis Expression	false
IsNullable	Y
Staging Attribute	Derived
Persistent Staging Attribute	Derived
Satellite Attribute	Source

RowIsInferred

Description

The RowIsInferred defines if the row is inferred

Example

```
false
```

Valid Value

SSIS Expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowIsInferred
Configuration Datatype	DataType="Boolean"
Configuration Default	1
Ssis Expression	false
IsNullable	Y
Dim Attribute	Derived

RowHash

Description

The RowHash defines the expression used to derive a full row hash. The default uses the combination of a forward hash and a backwards hash to minimise the risk for hash collisions. The default hashing mechanism uses the Varigence Custom SSIS Components.

Example

```
[vck@@this1]+[vck@@this2]
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Valid Value

A Valid SSIS Expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowHash
Configuration Datatype	DataType="AnsiString" Length="80"
Configuration Default	0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Ssis Expression	[vck@@this1]+[vck@@this2]
IsNullable	Y

KEY	VALUE
Staging Attribute	Hash
Persistent Staging Attribute	Hash

RowHashKey

Description

The **RowHashKey** defines the expression used to derive a key hash. The default hashing mechanism uses the Varigence Custom SSIS Components.

Example

```
[vck@@this1]
```

Valid Value

A valid SSIS Expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowHashKey
Configuration Datatype	DataType="AnsiString" Length="40"
Configuration Default	0000000000000000000000000000000000000000
Ssis Expression	[vck@@this1]
IsNullable	Y
Staging Attribute	Hash
Persistent Staging Attribute	Hash

RowHashSat

Description

The **RowHashSat** defines the expression used to derive a satellite attribute hash. The default hashing mechanism uses the Varigence Custom SSIS Components

Example

```
[@@this1]
```

Valid Value

A valid SSIS Expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowHashSat
Configuration Datatype	DataType="AnsiString" Length="40"
Configuration Default	0000000000000000000000000000000000000000
Ssis Expression	[vck@@this1]
IsNullable	Y
Dim Attribute	Hash

RowHashType1

Description

The **RowHashType1** defines the expression used to derive a forward hash. The default hashing mechanism uses the Varigence Custom SSIS Components

Example

```
[vck@@this1]
```

Valid Value

A valid SSIS Expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowHashType1
Configuration Datatype	DataType="AnsiString" Length="40"
Configuration Default	0000000000000000000000000000000000000000
Ssis Expression	[vck@@this1]
IsNullable	Y
Dim Attribute	Hash

RowHashType2

Description

The **RowHashType2** defines the expression used to derive a backward hash. The default hashing mechanism uses the Varigence Custom SSIS Components

Example

```
[vck@@this1]
```

Valid Value

A valid SSIS Expression and data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowHashType2
Configuration Datatype	DataType="AnsiString" Length="40"
Configuration Default	0000000000000000000000000000000000000000
Ssis Expression	[vck@@this1]
IsNullable	Y
Dim Attribute	Hash

RowLoadSequence

Description

The RowLoadSequence defines the data type for the Load Sequence organiser.

Example

```
DataType="Int32"
```

Valid Value

A valid sortable integer data type

Default Metadata information

KEY	VALUE
Configuration Value	FlexRowLoadSequence
Configuration Datatype	DataType="Int32"

Metadata and framework settings in BimlFlex

This document outlines the metadata and framework settings available in BimlFlex.

These Settings drive the behavior of the BimlFlex product.

By changing them, the produced artifacts can adapt to support requirements for file locations, naming conventions, data conventions etc.

The Settings defaults are the Varigence recommended values and there is no need to change or configure unless there is a requirement to change specific behaviors. Align these settings with the organizations best practices and environmental requirements.

The Settings are available in the Settings sheet in the BimlFlex Excel Add-in.

Metadata column overview

KEY	VALUE
Configuration Key	the Configuration Key, the internal key BimlFlex refers to, cannot be changed
Configuration Value	the Configured Value, can be updated to support a different design pattern or behaviour
Configuration Datatype	the data type the configuration Value uses. Needs to be a valid data type definition
Configuration Default	the Configuration Key's Default Value
Configuration Grouping	BimlFlex Internal Grouping of configurations
Configuration Order	BimlFlex Internal Ordering of configurations
SSIS Expression	the SSIS Expression used to derive the value. Needs to be a valid SSIS Expression. Uses the shorthand @@this to define the current entity
Is Nullable	Defines If the attribute is nullable Valid Enumeration {Empty, Y, N}
Staging Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Persistent Staging Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Hub Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Satellite Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Link Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Dim Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}

KEY	VALUE
Fact Attribute	Valid Enumeration {Ignore, Derived, Source, Default, Target, Hash}
Description	An optional description for custom attributes or definitions. The default configurations are described in this document
Is Deleted	Flag to set if a custom attribute has been entered but is now no longer needed and should be considered deleted. Valid Enumeration {Empty, Y, N}

Standard Settings

RootPath

Description

The **RootPath** defines the default Root Path folder of the BimlFlex solution.

Example

```
C:\Varigence\BimlFlex
```

Valid Value

A valid path

Default Metadata information

KEY	VALUE
Configuration Value	C:\Varigence\BimlFlex

ImportPath

Description

The **ImportPath** defines the default folder path for file import.

Example

```
C:\Varigence\Import
```

Valid Value

A valid path

Default Metadata information

KEY	VALUE
Configuration Value	C:\Varigence\Import

ExportPath

Description

The **ExportPath** defines the default folder path for file exports.

Example

```
C:\Varigence\Export
```

Valid Value

A valid path

Default Metadata information

KEY	VALUE
Configuration Value	C:\Varigence\Export

UseBimlCatalog

Description

The **UseBimlCatalog** defines whether or not to use the BimlFlex Catalog database for SSIS package orchestration and logging.

Example

```
Y
```

Valid Value

Enumeration {Y,N}

Default Metadata information

KEY	VALUE
Configuration Value	Y

ConfigurationPath

Description

The **ConfigurationPath** key defines the default path for configurations

Example

```
C:\Varigence\Configurations
```

Valid Value

A valid path

Default Metadata information

KEY	VALUE
Configuration Value	C:\Varigence\Configurations

7ZipPath

Description

The **7ZipPath** key defines the file path/location of the 7-Zip application that is used for zipping/compression of files. The 7-zip executables are needed for zip-related operations. The 7-Zip application is open source and available to use without license cost.

More information and downloads: http://www.7-zip.org/download.html

Example

```
C:\Program Files\7-Zip
```

Valid Value

A valid path to the 7-Zip executable

Default Metadata information

KEY	VALUE
Configuration Value	C:\Program Files\7-Zip

AzCopyPath

Description

The **AzCopyPath** key defines the file path/location of the AzCopy application used to copy files to Azure storage.

Example

```
C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy
```

Valid Value

A valid path to the AzCopy executable

Default Metadata information

KEY	VALUE
Configuration Value	C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy

KeyEndsWith

Description

The **KeyEndsWith** key defines the text the metadata import uses to identify key columns that aren't technically identified as keys in the source. Add any source specific key identifiers to enable automatic identification of key columns.

Example

```
Id,Code,No,Key,Cd
```

Valid Value

Any valid comma separated list of SQL or SSIS Strings

Default Metadata information

KEY	VALUE
Configuration Value	Id,Code,No,Key

SuffixOrPrefixColumn

Description

The SuffixOrPrefixColumn key defines the behaviour when defining column names. Use Suffix or Prefix to define if the column identifiers are added after or before the column names in the solution.

Example

```
S for Suffix will generate Entity_SK
P for Prefix Will generate SK_Entity
```

Valid Value

Enumeration {P, S}

Default Metadata information

KEY	VALUE
Configuration Value	S

SuffixOrPrefixObject

Description

The SuffixOrPrefixObject key defines the behaviour when defining object names. Use Suffix or Prefix to define if the object identifiers are added after or before the object names in the solution.

Example

```
S for Suffix Will generate Entity_HUB
P for Prefix Will generate HUB_Entity
```

Valid Value

Enumeration `{P, S}`

Default Metadata information

KEY	VALUE
Configuration Value	P

UseRecordSourceAsAppend

Description

The **UseRecordSourceAsAppend** Key specifies if the record source should be appended to the object name

Example

```
N
```

Valid Value

Enumeration `{Y, N}`

Default Metadata information

KEY	VALUE
Configuration Value	N

UseRecordSourceAsSchema

Description

The **UseRecordSourceAsSchema** Key specifies if the record source should be used as the schema for objects. As an example, the default behaviour means a source table called Product from the record source AWLT will be created as AWLT.Product in the Staging Area

Example

```
Y
```

Valid Value

Enumeration `{Y, N}`

Default Metadata information

KEY	VALUE
Configuration Value	Y

UseColumnModelOverride

Description

The **UseColumnModelOverride** Key specifies if the model override for column should be used instead of the source names for columns in the Staging and Persisted Staging areas. The recommended, and default behaviour, is to use source names for Staging and only use override names in the Data Vault/later layers.

Example

```
N
```

Valid Value

Enumeration `{Y, N}`

Default Metadata information

KEY	VALUE
Configuration Value	N

UseObjectModelOverride

Description

The **UseObjectModelOverride** Key specifies if the model override for objects should be used instead of the source names for objects in the Staging and Persisted Staging areas. The recommended, and default behaviour, is to use source names for Staging and only use override names in the Data Vault/later layers.

Example

```
N
```

Valid Value

Enumeration `{Y, N}`

Default Metadata information

KEY	VALUE
Configuration Value	N

HashBusinessKey

Description

The **HashBusinessKey** Key specifies if the Business Key should be hashed. This is implemented by default for Data Vault regardless of setting but can be specified for other modelling approaches.

Example

```
N
```

Valid Value

Enumeration `{Y, N}`

Default Metadata information

KEY	VALUE
Configuration Value	N

ConcatenatorBusinessKey

Description

The **ConcatenatorBusinessKey** Key specifies the value to use as filler between business keys when concatenating them. Single business keys are created from multiple source columns through concatenation to maintain a single business key. When concatenating it is important to be able to distinguish between similarly formed inputs. E.g. concatenating ABC and DEF without the concatenator will

make it the same as AB + CDEF (`ABCDEF`). The concatenator will maintain the two as different and distinct entities (`ABC~DEF` vs. `AB~CDEF`). Using a concatenator is required to maintain data integrity but the value can be configured to support an existing process, design pattern or specific requirement.

Example

```
~
```

Valid Value

any valid string value usable for string parsing

Default Metadata information

KEY	VALUE
Configuration Value	~

BusinessKeyNullValue

Description

The **BusinessKeyNullValue** Key specifies the defined value to use for null values in the business key.

Example

```
~NULL~
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	NVL

AppendBusinessKey

Description

The **AppendBusinessKey** Key specifies the string to append to the Business Key Columns. Prefixing or suffixing is specified by the SuffixOrPrefixColumn configuration

Example

```
BK = BusinessKeyColumnName_BK
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE	DATATYPE
Configuration Value	BK	DataType="AnsiString" Length="40"

AppendSurrogateKey

Description

The **AppendSurrogateKey** Key specifies the string to append to the Surrogate Key Columns. Prefixing or suffixing is specified by the SuffixOrPrefixColumn configuration

Example

```
SK = KeyColumnName_SK
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE	DATATYPE
Configuration Value	SK	DataType="AnsiString" Length="40"

AppendRecordSource

Description

The **AppendRecordSource** Key specifies if the Record source should be appended to object names

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

AppendSchemaDm

Description

The **AppendSchemaDm** Key specifies if the Schema should be appended in the Data Mart layer

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

AppendSchemaRdv

Description

The **AppendSchemaRdv** Key specifies if the Schema should be appended in the Data Vault layer

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

AppendSchemaPsa

Description

The **AppendSchemaPsa** Key specifies if the Schema should be appended for the Persistent Staging layer when colocated in the Staging database

Example

```
ods
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	ods

AppendSchemaStg

Description

The AppendSchemaStg Key specifies if the source schema should be appended to the object name. this is useful for when a source has multiple schemas with the same object name repeated across these schemas. To be able to distinguish between them in the Staging Area the schema name needs to be added. The default process disregards the schema for simplicity in the naming. An example where this might be needed is when loading all tables from the WideWorldImporters demo database where the same table name is repeated across multiple schemas.

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

AppendDomain

Description

Should Domain be added

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

DisplayDatabaseNameStg

Description

Should the Database name be added to the Staging Layer

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

DisplaySchemaNameStg

Description

Should the source schema name be added to the Staging Layer

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

LookupCachePath

Description

The `LookupCachePath` Key specifies the path for cache files for the lookup process when it is using external persistence of cache data.

Example

```
C:\Varigence\Cache
```

Valid Value

Any valid and safe path

Default Metadata information

KEY	VALUE
Configuration Value	C:\Varigence\Cache

DisplayDatabaseNameRdv

Description

The `DisplayDatabaseNameRdv` Key specifies if the database name should be added to the Rdv layer

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

DisplaySchemaNameRdv

Description

The **DisplaySchemaNameRdv** Key specifies if the source schema name should be added to the Rdv Layer

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

LookupAddFilterTable

Description

The **LookupAddFilterTable** Key specifies if table filter should be added to the lookup

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

DisplayDatabaseNameDm

Description

The **DisplayDatabaseNameDm** Key specifies if the objects database name should be displayed in the Data Mart.

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

LookupTablePattern

Description

The **LookupTablePattern** Key specifies the lookup naming convention used for SSIS table lookup.

Example

```
lkp.ReferenceColumnName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	"lkp." + column.Name.MakeSsisSafe

DisplaySchemaNameDm

Description

The **DisplaySchemaNameDm** Key specifies if the objects schema name should be displayed in the Data Mart.

Example

```
N
```

Valid Value

Enumeration {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

DmAppendDim

Description

The **DvAppendDim** Key specifies the string to append to Dimension objects in the Data Mart. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
DIM = DIM_DimensionEntityName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	DIM

DmAppendFact

Description

The **DvAppendFact** Key specifies the string to append to Fact objects in the Data Mart. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
FACT = FACT_FactEntityName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	FACT

DmAppendExternal

Description

The **DvAppendExternal** Key specifies the string to append to External objects in the Data Mart. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
EXT = EXT_ExternalEntityName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	EXT

DmAppendStaging

Description

The **DvAppendStaging** Key specifies the string to append to Staging objects in the Data Mart. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
STAGE = STAGE_DimensionEntityName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	STAGE

DvAppendBridge

Description

The **DvAppendBridge** Key specifies the string to append to Bridge objects. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
BRD = BRD_BridgeEntityName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	BRD

DvAppendHub

Description

The **DvAppendHub** Key specifies the string to append to Hub objects. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
HUB = HUB_EntityName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	HUB

DvAppendLink

Description

The **DvAppendLink** Key specifies the string to append to Link objects. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
LNK = LNK_LinkName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	LNK

DvAppendLinkSatellite

Description

The **DvAppendLinkSatellite** Key specifies the string to append to Link Satellite objects. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
LSAT = LSAT_LinkName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	LSAT

DvAppendReference

Description

The **DvAppendReference** Key specifies the string to append to Reference objects. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
REF = REF_ReferenceEntityName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	REF

DvAppendSatellite

Description

The **DvAppendSatellite** Key specifies the string to append to Satellite objects. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
SAT = SAT_EntityName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	SAT

DvPreviewSchema

Description

The **DvPreviewSchema** Key specifies the default schema to use for Data Vault Accelerator generated preview objects

Example

```
pdv
```

Valid Value

Any valid and safe SQL schema name

Default Metadata information

KEY	VALUE
Configuration Value	pdv

DvDefaultSchema

Description

The **DvDefaultSchema** Key specifies the default schema to use for Data Vault objects

Example

```
rdv
```

Valid Value

Any valid and safe SQL schema name

Default Metadata information

KEY	VALUE
Configuration Value	rdv

DvAppendPointInTime

Description

The **DvAppendPointInTime** Key specifies the string to append to Point in time objects. Prefixing or suffixing is specified by the SuffixOrPrefixObject configuration

Example

```
PIT = PIT_EntityEventName
```

Valid Value

Any valid and safe SQL and SSIS String

Default Metadata information

KEY	VALUE
Configuration Value	PIT

DvSnapshotFromDate

Description

The **DvSnapshotFromDate** Key specifies the Data Vault Snapshot from/start date

Example

```
0001-01-01 00:00:00.000
```

Valid Value

Any valid and safe SQL and SSIS date datatype and date expression

Default Metadata information

KEY	VALUE	DATATYPE	DEFAULT
Configuration Value	SnapshotFromDate	DataType="DateTime2" Scale="7"	0001-01-01 00:00:00.000

DvSnapshotToDate

Description

The **DvSnapshotToDate** Key specifies the Data Vault Snapshot to/end date

Example

```
9999-12-31 00:00:00.000
```

Valid Value

Any valid and safe SQL and SSIS date datatype and date expression

Default Metadata information

KEY	VALUE
Configuration Value	SnapshotToDate
Configuration Datatype	DataType="DateTime2" Scale="7"
Configuration Default	9999-12-31 00:00:00.000

DvSnapshotIncremental

Description

The **DvSnapshotIncremental** Key specifies if the Data Vault Snapshot feature should provide incremental snapshots.

Example

```
Y
```

Valid Value

Enumerator {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	Y

DvSnapshotLastModifiedDate

Description

The **DvSnapshotLastModifiedDate** Key specifies the Data Vault Snapshot last modified date

Example

```
0001-01-01 00:00:00.000
```

Valid Value

Any valid and safe SQL and SSIS datatype and expression

Default Metadata information

KEY	VALUE
Configuration Value	LastModifiedDate
Configuration Datatype	DataType="DateTime2" Scale="7"
Configuration Default	0001-01-01 00:00:00.000

EnableRollbackStg

Description

The **EnableRollbackStg** Key specifies if the Staging Area should accommodate the orchestration rollback feature. This will roll back a failed previous load when identified by the orchestration engine. Note that the Staging Area is truncated on load using the normal load pattern making rollback here irrelevant.

Example

```
N
```

Valid Value

Enumerator {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

EnableRollbackPsa

Description

The **EnableRollbackPsa** Key specifies if the Persistent Staging Area should accommodate the orchestration rollback feature. This will roll back a failed previous load when identified by the orchestration engine.

Example

```
N
```

Valid Value

Enumerator {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

EnableRollbackRdv

Description

The **EnableRollbackRdv** Key specifies if the Raw Data Vault should accommodate the orchestration rollback feature. This will roll back a failed previous load when identified by the orchestration engine.

Example

```
N
```

Valid Value

Enumerator {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

EnableInitialRecordRdv

Description

The **EnableInitialRecordRdv** Key specifies if the Raw Data Vault should produce initial records for entities. This is useful when an unbroken timeline is needed to support equijoins, inner joins on Hubs/Links to Sats regardless of the effectiveness dates used. With this configuration set to No, a satellite load of a new business key will only add a single row to the Raw Data Vault table. The effectiveness will be from the batch load date time to end of time. With the configuration set to Yes the Satellite load process will add 2 rows, the additional one will be a zero or ghost row with an effectiveness from start of time to the batch load date time.

Example

```
N
```

Valid Value

Enumerator {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	N

EnableEndDateRdv

Description

The **EnableEndDateRdv** Key specifies if the Raw Data Vault should end date loaded information. This is useful to simplify reads out of the Data Vault. The end dating will change the end data of the previous row to the load date of the new row and the new row will have an end date equal to the end of time specification. The process will also maintain a isCurrent flag for rows meaning it is trivial to derive the current valid set of data. The end dating process adds time and effort to the load processing time and can therefore be configured if needed. Setting this to No can potentially increase performance when loading in to the Data Vault.

Example

```
N
```

Valid Value

Enumerator {Y, N}

Default Metadata information

KEY	VALUE
Configuration Value	Y

SsisMaxConcurrentExecutables

Description

The **SsisMaxConcurrentExecutables** Key specifies the number of SSIS control flow executables that can run in parallel. The default value of -1 translates to the number of logical processors plus 2 concurrent tasks. This value can be tweaked to optimise performance in certain scenarios.

Example

```
10
```

Valid Value

A valid Integer value

Default Metadata information

KEY	VALUE
Configuration Value	-1

SsisEngineThreads

Description

The **SsisEngineThreads** Key specifies the SSIS property with the same name. this defines the number of engine threads SSIS will use. This value can be tweaked to optimise performance in certain scenarios.

Example

```
10
```

Valid Value

A valid Integer value

Default Metadata information

KEY	VALUE
Configuration Value	10

SsisMaximumInsertCommitSize

Description

The **SsisMaximumInsertCommitSize** Key specifies the SSIS property with the same name. this defines the maximum insert commit size to use in a bulk operation. This value can be tweaked to optimise performance in certain scenarios.

Example

```
2147483647
```

Valid Value

A valid integer value

Default Metadata information

KEY	VALUE
Configuration Value	2147483647

SsisRowsPerBatch

Description

The **SsisRowsPerBatch** Key specifies the SSIS property with the same name. this defines the number of rows to use in a bulk operation. This value can be tweaked to optimise performance in certain scenarios.

Example

```
500000
```

Valid Value

A valid integer value

Default Metadata information

KEY	VALUE
Configuration Value	500000

SsisValidateExternalMetadata

Description

The SsisValidateExternalMetadata Key specifies if the SSIS component should validate the external metadata against the cached information. Setting this to false can be useful if the source metadata should be disregarded due to temporary changes or similar scenarios

Example

```
True
```

Valid Value

Boolean Enumerator {True False}

Default Metadata information

KEY	VALUE
Configuration Value	true

SsisDelayValidation

Description

The SsisDelayValidation Key specifies if the Ssis component should delay metadata validation. This is useful when the source or reference is not always available and there is a need to delay validation until later

Example

```
True
```

Valid Value

Boolean Enumerator {True, False}

Default Metadata information

KEY	VALUE
Configuration Value	true

SsisCheckConstraints

Description

The SsisCheckConstraints Key specifies if the destination transformation should check constraints when writing to source. By default, this is disabled to enable faster transfers. Within Staging, Persistent Staging and Data Vault layers' constraints should not be enforced as it makes the solution less flexible and disallows parallel and out of sequence loading.

Example

```
False
```

Valid Value

Boolean Enumerator {True, False}

Default Metadata information

KEY	VALUE
Configuration Value	false

SsisCommandTimeout

Description

The SsisCommandTimeout Key specifies the command timeout to use for SSIS tasks

Example

```
10
```

Valid Value

A valid Integer value

Default Metadata information

KEY	VALUE
Configuration Value	0

SsisDefaultBufferMaxRows

Description

Defines the Maximum number of rows used in a task buffer. The default used is 10,000 rows

Example

```
10000
```

More information

https://msdn.microsoft.com/en-us/library/ms141031.aspx

Valid Value

A valid Integer value

Default Metadata information

KEY	VALUE
Configuration Value	10000

SsisDefaultBufferSize

Description

Define the default size of the buffer that the task uses, by setting the DefaultBufferSize property. The default buffer size is 10 megabytes,

with a maximum buffer size of 2^31-1 bytes.

Example

```
10485760
```

More information

https://msdn.microsoft.com/en-us/library/ms141031.aspx

Valid Value

A valid positive Integer value up to 2^31-1 bytes.

Default Metadata information

KEY	VALUE
Configuration Value	10485760

SsisBufferTempStoragePath

Description

Defines the part where SSIS will store temporary buffer data if needed when processing a package. The default location (defined by the TMP/TEMP environment variables) is used if this key is empty.

Example

```
E:\FastDisk\Folder\
```

Valid Value

A valid path.

Default Metadata information

KEY	VALUE
Configuration Value	

SsisBLOBTempStoragePath

Description

Defines the part where SSIS will store temporary BLOB data if needed when processing a package. The default location (defined by the TMP/TEMP environment variables) is used if this key is empty.

Example

```
E:\FastDisk\Folder\
```

Valid Value

A valid path.

Default Metadata information

KEY	VALUE
Configuration Value	

AzureDestStorageAccountName

Description

The Destination Storage Account Name used when writing to an Azure storage container.

Valid Value

A valid Account Name. Account names are unique across Azure, and contains 3-24 lowercase characters and numbers

Default Metadata information

KEY	VALUE
Configuration Value	

AzureDestStorageAccountKey

Description

The Destination Account Key used when writing to an Azure storage container.

Valid Value

A valid Account Key.

Default Metadata information

KEY	VALUE
Configuration Value	

AzureDestContainerName

Description

The Destination Container Name used for writing to an Azure storage container.

Valid Value

A valid Azure Container Name. container names are 3-63 lowercase alphanumeric and dash

Default Metadata information

KEY	VALUE
Configuration Value	

AzureSourceStorageAccountName

Description

The Source Storage Account Name used for sourcing from an Azure storage container.

Valid Value

A valid Account Name. Account names are unique across Azure, and contains 3-24 lowercase characters and numbers

Default Metadata information

KEY	VALUE
Configuration Value	

AzureSourceStorageAccountKey

Description

The Source Container Account Key used for sourcing from an Azure storage container.

Valid Value

A valid Azure Storage Account Key.

Default Metadata information

KEY	VALUE
Configuration Value	

AzureSourceContainerName

Description

The Source Container Name used for sourcing from an Azure storage container.

Valid Value

A valid Azure Container Name. container names are 3-63 lowercase alphanumeric and dash

Default Metadata information

KEY	VALUE
Configuration Value	

Data Type Mappings

TODO: Content coming soon

Versions

TODO: Content coming soon

Importing Metadata

The primary source BimlFlex uses to generate Data Warehousing and Business Intelligence assets is metadata. Metadata is most commonly in the form of objects with corresponding attributes.

Although it is relatively straightforward to define individual metadata objects, it can take considerable time and effort to create, update and maintain the number of objects that would typically be involved in an enterprise data warehouse solution.

The BimlFlex Excel Add-in provides an import tool that connects to a database source and populates the BimlFlex database with metadata.

This tool can be used in any scenario where a schema is available, and there is a need to bring that metadata into a BimlFlex project. If a change happens in the source, the metadata can be imported again to update the model.

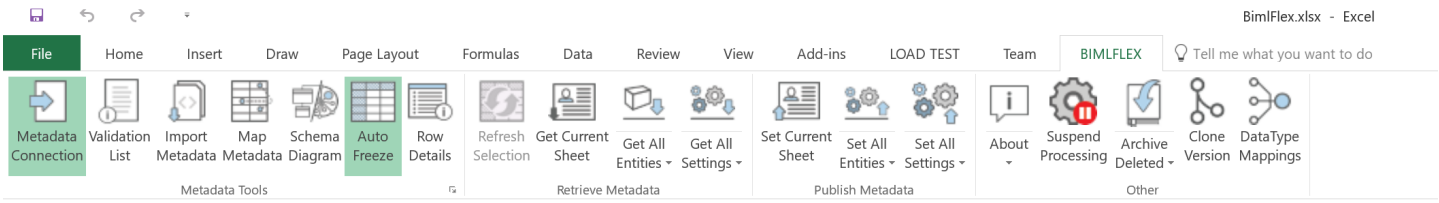
Watch Recordings

02. BimlFlex - Importing Metadata

In this session, we look at importing metadata using BimlFlex. The metadata is published and committed and persisted in the meta vault repository.

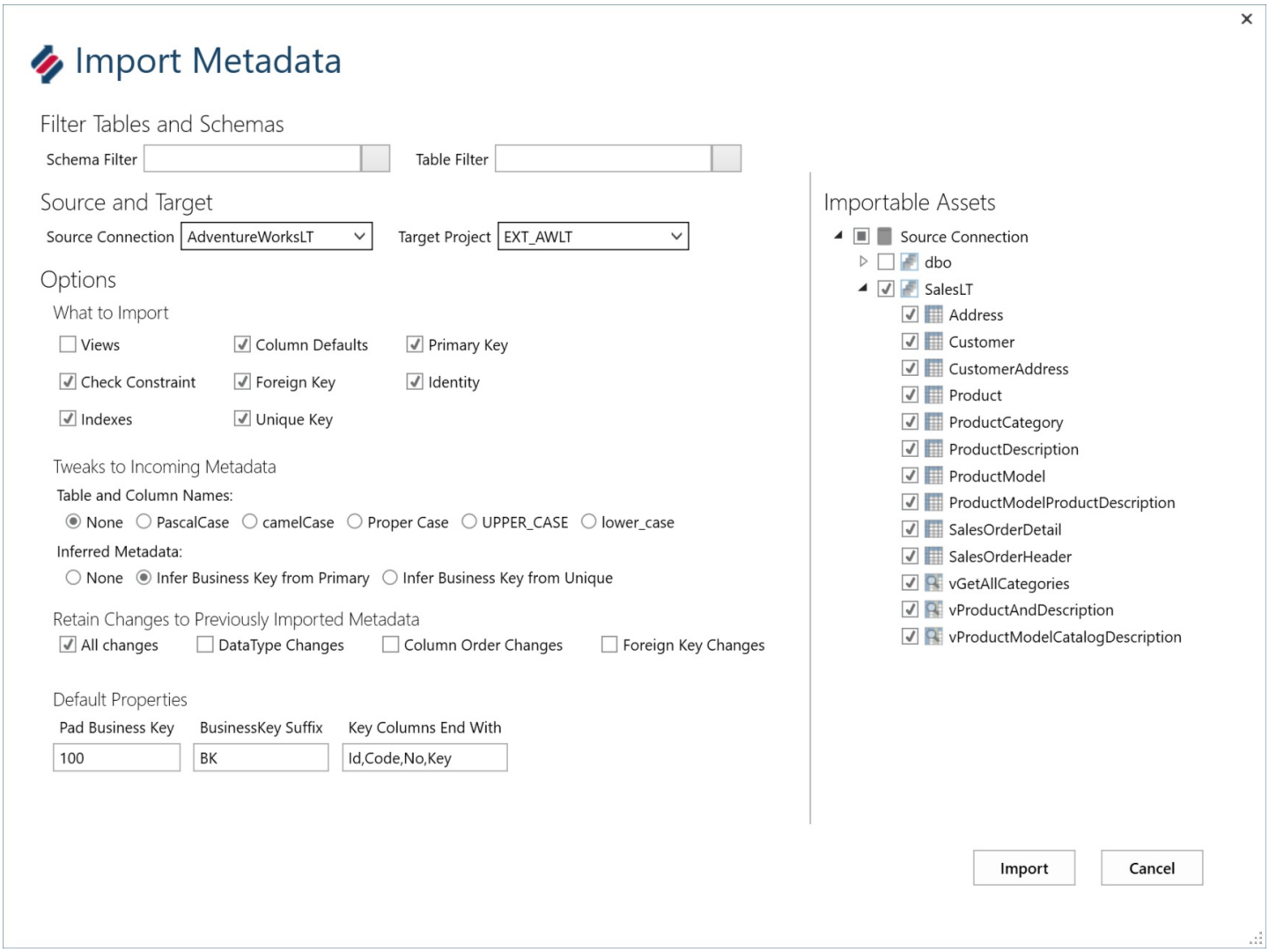
Locating the tool

In the BimlFlex Excel Add-in, BimlFlex tab, in the metadata tools group, is the button `Import Metadata`. This opens the Import Metadata dialog.



Import Metadata UI

There are a number of options for importing metadata. Align these options with the solution architecture and metadata conventions.



The example source is the AdventureWorksLT database, using the project `EXT_AWLT` as the target for imported metadata.

On the left-hand side are all the main options. Use these to control the behavior of the import, such as what object types to import, naming conventions to apply and what to retain if the metadata has been imported before.

Metadata Options Definitions

A brief overview of the UI.

Filter Tables and Schemas

The Schema Filter and Table Filter can be used to limit the Assets made available/visible for import. This is useful if a subset of available tables will be imported or if new tables are added to an existing repository. Filtering also improves performance when connecting to and importing large database schemas.

Source and Target

OPTION NAME	DEFINITION
Source Connection	Name of the connection that points to the data source the project is extracting from. The Connection is defined in the Connections tab in the Metadata.
Target Project	Name of the previously defined project that will use this metadata as its source in the source to target loading. The Project is defined in the Project tab in the Metadata.

What to Import

Allows inclusion or exclusion of certain object types in the imported metadata. The database objects available to be chosen include:

- Views
- Column Defaults
- Primary Keys
- Check Constraints
- Foreign Keys
- Identities
- Indexes
- Unique Keys

Tweaks to Incoming Metadata

Options that can override the format of incoming metadata. Align these options with organizational standards and best practices.

Table and column names

Options to apply naming conventions to imported metadata.

- None - This will retain the source system default naming conventions.
- PascalCase - E.g. "CustomerAddress"
- camelCase - E.g. "customerAddress"
- Proper Case - E.g. "Customer Address"
- UPPER_CASE - E.g. "CUSTOMER_ADDRESS"
- lower_case - E.g. "customer_address"

Inferred Metadata

Infers Business Keys from column constraints

- None - Nothing
- Infer Business Key from Primary Key
- Infer Business Key from Unique Key

Retain Changes to Previously Imported Metadata

Controls if existing metadata should be retained during the import of metadata.

- All Changes, Don't override anything.
- Data Type Changes
- Column Order Changes
- Foreign Key Changes

Default Properties

This controls default behaviour of inferred Business Keys.

- Pad Business Keys (Amount) – defines business key width
- BusinessKey Suffix - Optionally add a suffix onto column name to indicate what is a business key
- Key Columns End With – When inferring keys, what substring to look for at the end of a column name - Id, Code, No, Key or any custom string added

Importable Assets

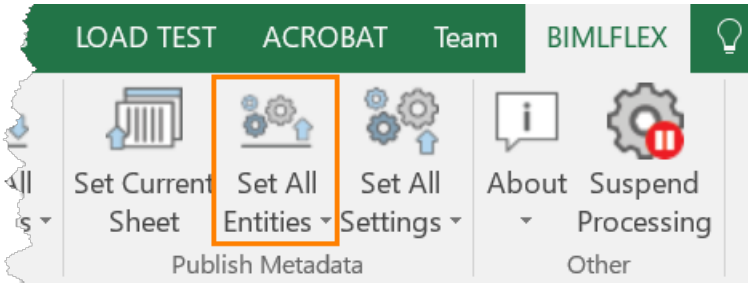
The Importable Assets list is read from the Source Connection. Choose items that should be included in the metadata import

The metadata will then be imported into the currently open metadata spreadsheet.

	B	C	D	E	F	G	H	I	J	K
1	Connection	ObjectName	ColumnName	DataType	Length	Precision	Scale	Ordinal	ChangeType	IsPrimaryKey
2	AdventureWorl	SalesLT.Custome	Customer_BK	String	100			1	KEY	
3	AdventureWorl	SalesLT.Custome	CustomerID	Int32				2	KEY	Y
4	AdventureWorl	SalesLT.Custome	NameStyle	Boolean				3	CHG	
5	AdventureWorl	SalesLT.Custome	Title	String	8			4	CHG	
6	AdventureWorl	SalesLT.Custome	FirstName	String	50			5	CHG	
7	AdventureWorl	SalesLT.Custome	MiddleName	String	50			6	CHG	
8	AdventureWorl	SalesLT.Custome	LastName	String	50			7	CHG	
9	AdventureWorl	SalesLT.Custome	Suffix	String	10			8	CHG	
10	AdventureWorl	SalesLT.Custome	CompanyName	String	128			9	CHG	
11	AdventureWorl	SalesLT.Custome	SalesPerson	String	256			10	CHG	
12	AdventureWorl	SalesLT.Custome	EmailAddress	String	50			11	CHG	
13	AdventureWorl	SalesLT.Custome	Phone	String	25			12	CHG	
14	AdventureWorl	SalesLT.Custome	PasswordHash	AnsiString	128			13	CHG	
15	AdventureWorl	SalesLT.Custome	PasswordSalt	AnsiString	10			14	CHG	
16	AdventureWorl	SalesLT.Custome	rowguid	Guid				15	CHG	
17	AdventureWorl	SalesLT.Custome	ModifiedDate	DateTime				16	CHG	
18	AdventureWorl	SalesLT.Custome	CustomerAddress	String	100			1	KEY	

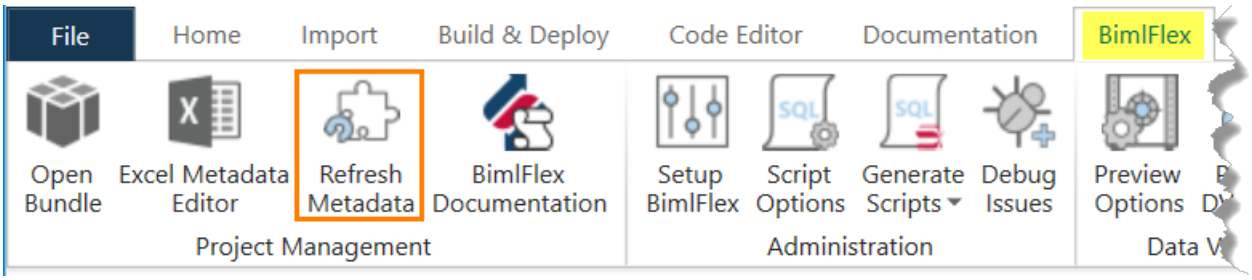
Note that this has not yet been committed to the metadata repository, these are unsaved rows of metadata.

After review, commit the imported metadata to the BimlFlex metadata repository by clicking `Set All Entities` from the publish metadata section.



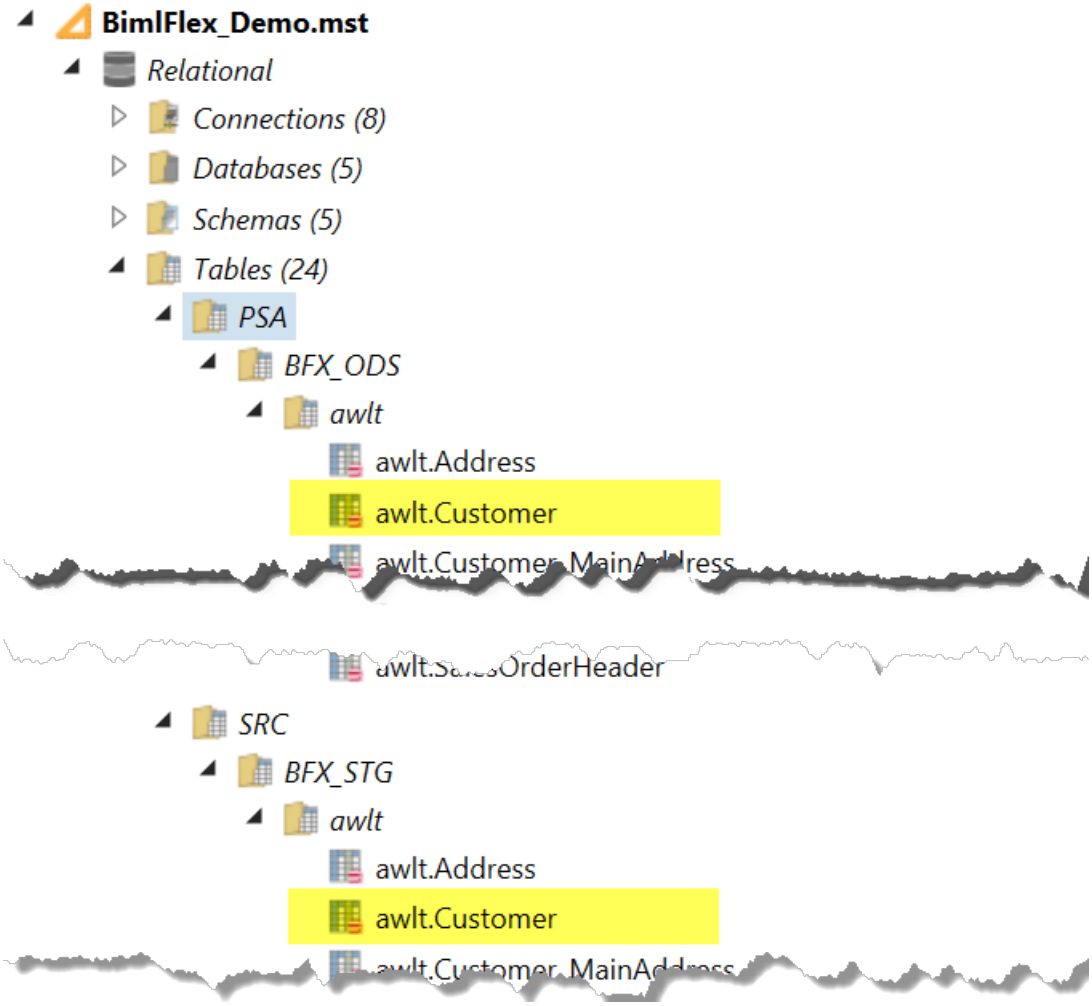
Verify Imported Metadata in BimlStudio

Open the BimlFlex project in BimlStudio or press the `Refresh Metadata` button if already opened.

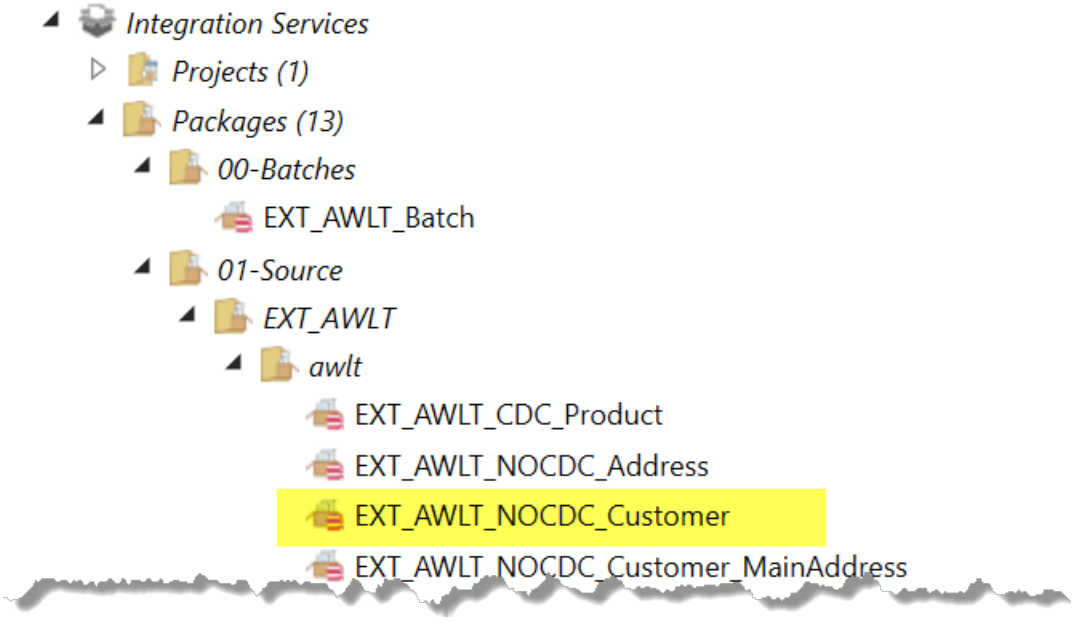


Verify that the newly imported objects is visible under the `Relational -> Tables -> SRC` folder and the `Integration Services -> 01-Source` folder.

Tables



SSIS Packages



Source to Staging Templates

BimlFlex offers users the ability to automate the development of best practice ETL out of the box using metadata modeling. As part of this, there are various options for users when designing a staging layer for their solution.

Watch Recordings

03.BimlFlex - Source To Staging

In this session, we explore the process of configuring Source to Staging process.

05.BimlFlex - Object Inheritance

In this session we look at Object Inheritance to reuse metadata.

13.BimlFlex - Import Files

In this session we look at how to configure flat files.

Source to Staging

Using metadata, the source schemas are mapped to a staging environment. This provides all the required metadata for the source to staging/persistent staging. When run through Ssis, data is transferred and any transformations that are defined using SsisExpressions and/or SqlExpressions are applied.

The source to staging creation can be toggled on or off based on what metadata has been defined.

Databasea sources and Flat File sources utilize different templates. The template used is controlled by the IntegrationTemplate property of the project.

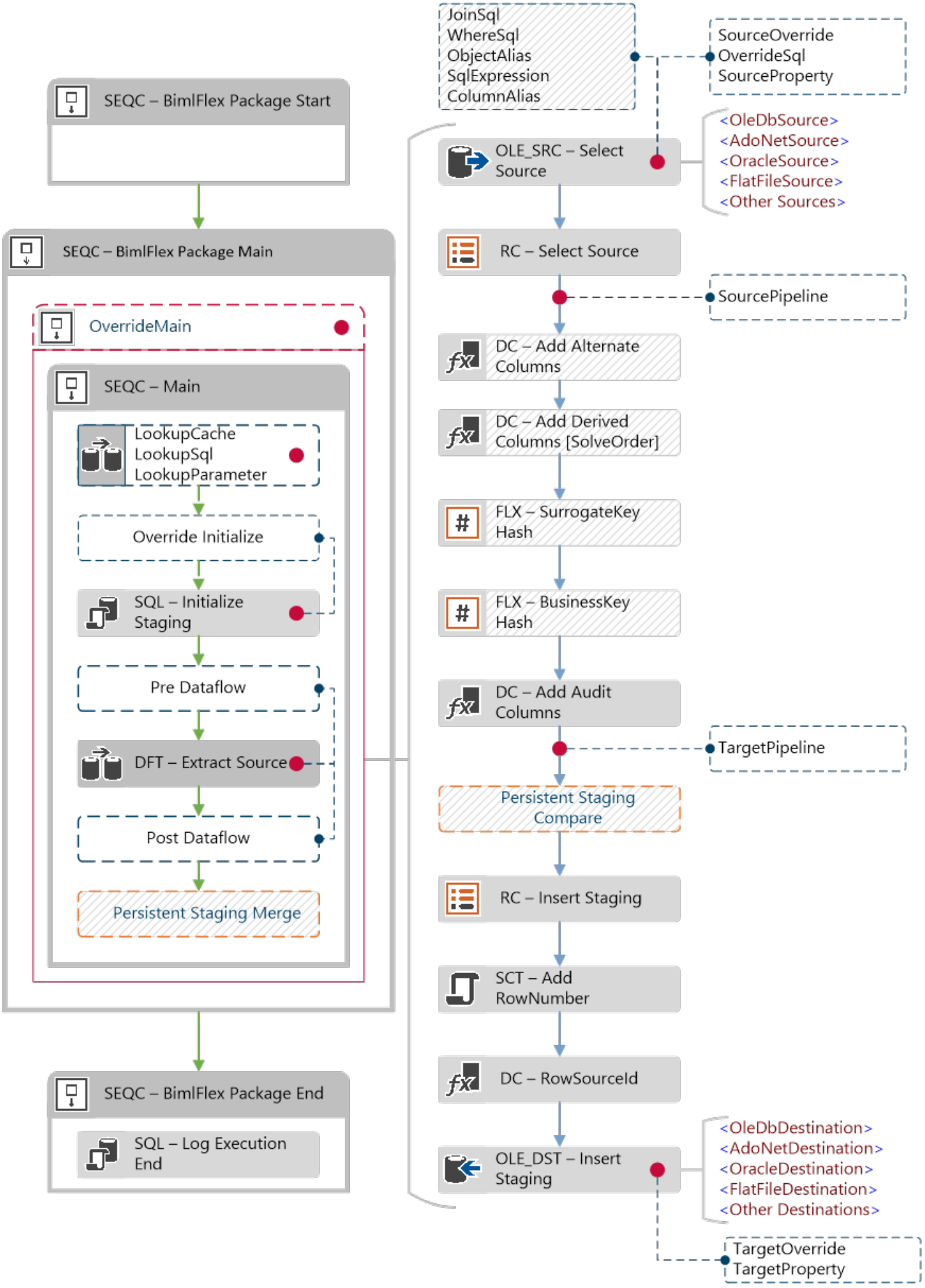
Integration Templates

Below is the currently supported Integration Templates

PROJECT INTEGRATION TEMPLATES	STAGING AVAILABLE
Source -> Target	Yes
Source -> File Extract	Not Applicable
Source -> Zip File Extract	Not Applicable

ETL Pattern Overview

The standard framework being applied. This can be overridden by using metadata settings or Extension Points, signified by the red dots and dashed lined rectangles.



[SEQC – BimlFlex Package Start] is added to every package and have an OnPreExecute EventHandler that decides if the package should execute and/or perform any Rollback operations prior to executing.

[SEQC – Main] is where the core source to staging load is implemented.

[SQL – Initialize Staging] Truncates the target staging table.

[DFT – Extract Source] Data flow task that implements the source to staging data load process.

[OLE_SRC – Select Source] Source component for the extraction. This contains the source SQL `SELECT` statement.

[RC – Select Source] logs the number of input rows into `[BimlCatalog].[ssis].[RowCount]`

[DC – Audit Columns] adds all the ETL audit columns to the pipeline. Depending on configuration these could be:

- `RowEffectiveFromDate`
- `RowEffectiveToDate`
- `RowLastSeenDate`
- `RowStartDate`
- `RowEndDate`
- `RowAuditId`
- `RowChangeType`
- `RowRecordSource`
- `RowIsCurrent`
- `RowIsDeleted`

[RC – Insert Staging] logs the number of insert rows into `[BimlCatalog].[ssis].[RowCount]`

[SCT – Add RowNumber] adds a sequence number to the pipeline. (as not all destinations support identity columns).

[DC – RowSourceId] derives the sequence number into the data type and column name specified for RowSourceId in the Configuration metadata.

[OLE_DST – Insert Staging] inserts all new columns into the staging table that have been extracted from the source component along with any derived columns.

[SEQC – BimlFlex Package End] is added to every package and logs package success completion.

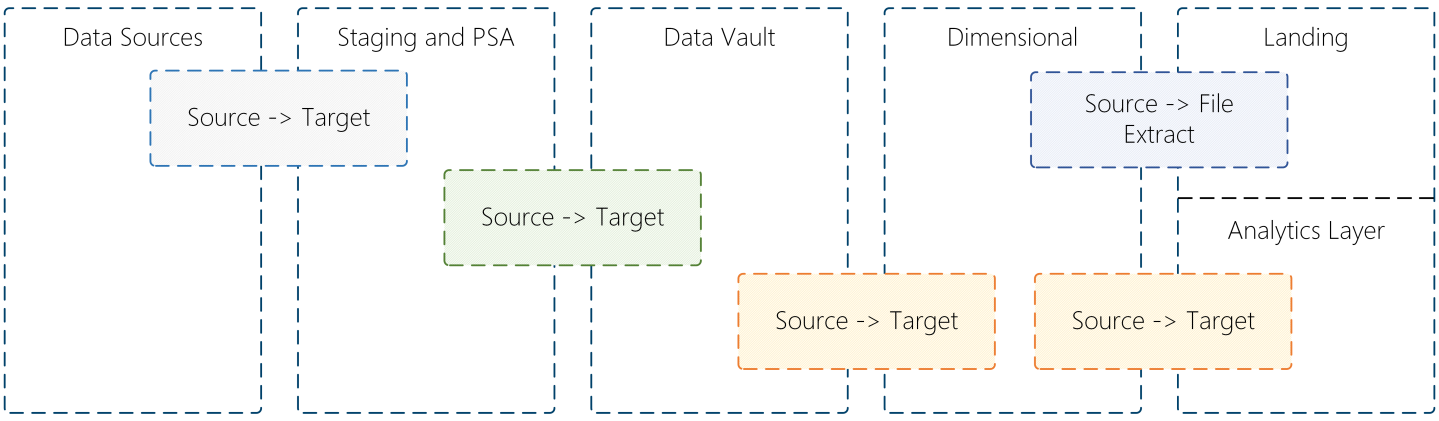
Adding Staging to a Solution

Project Implementation

BimlFlex supports source to target project that extracts data from a source system, staging deltas and persisting changes into Persistent Staging.

Once data is staged it can be loaded further to the target destination. There are several ways a project can be configured.

The diagram below shows sample project loads loads to suit different scenarios.



Base project design and architecture in BimlFlex based on the layers and modeling techniques used in the organization.

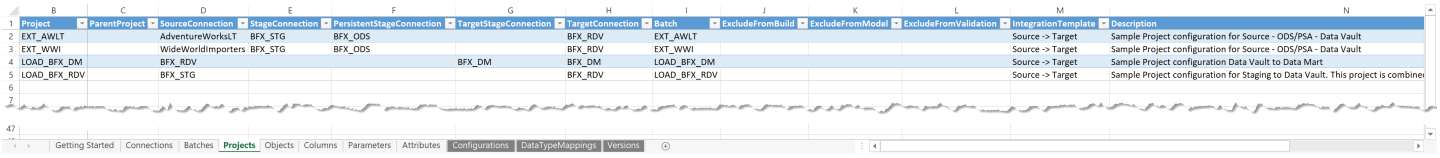
Example: properties included in a project with staging.

ATTRIBUTE NAME	EXAMPLE PROJECT ATTRIBUTES
Project	EXT_AW
Source Connection	AdventureWorksLT
Stage Connection	AW_STG
Persistent Stage Connection (Optional)	AW_ODS
Target Connection (final destination of the data: DW, DV etc)	AW_DV
Batch Name	EXT_AW
Parent Batch (Optional)	
Integration Template. (Source Target)	Source Target

This Project configuration describes a project that loads data that from source system AdventureWorksLT and prepares for the final Data Vault (`BFX_RDV`) destination.

Adding a valid staging connection in the Stage Connection property will include all required tables and ETL to the solution in BimlStudio.

By defining the final destination as Data Vault, the process optimizer includes hashing of keys into Staging that are required for Data Vault. Note that the sample projects below has separate projects for Source To Staging to Persistent Staging and Staging to Data Vault.



Project	ParentProject	SourceConnection	StageConnection	PersistentStageConnection	TargetStageConnection	TargetConnection	Batch	ExcludeFromBuild	ExcludeFromModel	ExcludeFromValidation	IntegrationTemplate	Description
EXT_AWLT		AdventureWorksLT	BFX_STG	BFX_ODS		BFX_RDV	EXT_AWLT				Source -> Target	Sample Project configuration for Source - ODS/PSA - Data Vault
EXT_WWI		WideWorldImporters	BFX_STG	BFX_ODS		BFX_RDV	EXT_WWI				Source -> Target	Sample Project configuration for Source - ODS/PSA - Data Vault
LOAD_BFX_DM		BFX_RDV			BFX_DM	BFX_DM	LOAD_BFX_DM				Source -> Target	Sample Project configuration Data Vault to Data Mart
LOAD_BFX_RDV		BFX_STG				BFX_RDV	LOAD_BFX_RDV				Source -> Target	Sample Project configuration for Staging to Data Vault. This project is combine

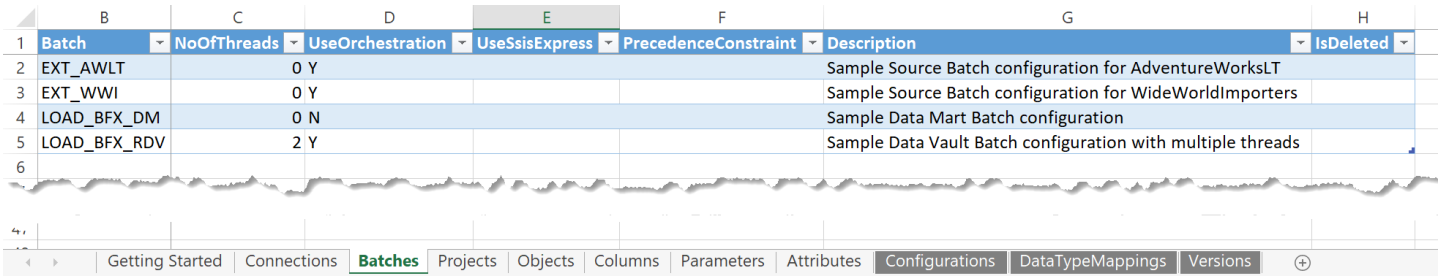
Getting Started | Connections | Batches | Projects | Objects | Columns | Parameters | Attributes | Configurations | DataTypeMappings | Versions

Defining a new Source to Staging project:

1. Insert a valid batch name to use as its parent package
2. Insert valid source and target destination connections
3. Insert a valid staging connection, and if applicable add a valid Persistent Staging connection.
4. Select the Source to Target integration template. Source to Staging loads uses the Source to Target template.
5. Commit the changes to the Metadata database through the "Set Current Sheet" button on the Projects tab.

Batch Implementation

Batches represent parent packages that control what set of child packages will be executed together.



	B	C	D	E	F	G	H
1	Batch	NoOfThreads	UseOrchestration	UseSsisExpress	PrecedenceConstraint	Description	IsDeleted
2	EXT_AWLT	0	Y			Sample Source Batch configuration for AdventureWorksLT	
3	EXT_WWI	0	Y			Sample Source Batch configuration for WideWorldImporters	
4	LOAD_BFX_DM	0	N			Sample Data Mart Batch configuration	
5	LOAD_BFX_RDV	2	Y			Sample Data Vault Batch configuration with multiple threads	
6							

Defining a Batch:

1. Use a descriptive name
2. `NoOfThreads`, Number Of Threads, control the parallelism if the execution within the Batch
3. `UseOrchestration`, Use Orchestration, controls if the Batch will use the Orchestration function within the Batch
4. `UseSsisExpress`, Use SSIS Express, controls if BimlFlex should generate SSIS Express packages. SSIS Express packages can run without a licensed Sql Server but has limited functionality
5. `PrecendenceConstraint`, Precedence Constraint, controls if the execute package tasks in the batch should implement a specific Precedence Constraint

Customizing Source Queries

BimlFlex supports tweaks to automated source queries through metadata as well as custom source queries through Extension Points. Joins and where filters are easily added through metadata.

JoinSql

Add `JOIN` statements in the JoinSql column in the Objects sheet. Example, to join the Customer and Sales tables:

```
INNER JOIN [sales].[Customer] cst ON sls.[CustomerID] = cst.[CustomerID]
```

The join condition uses aliases for tables. The alias for a table is specified in the ObjectAlias metadata column

WhereSql

Add `WHERE` statements in the WhereSql column in the Objects sheet. Example, filter AddressType on a string literal:

```
WHERE ca.[AddressType] = 'Main Office'
```

GroupBySql

Add `GROUP BY` statements in the GroupBySql column in the Objects sheet.

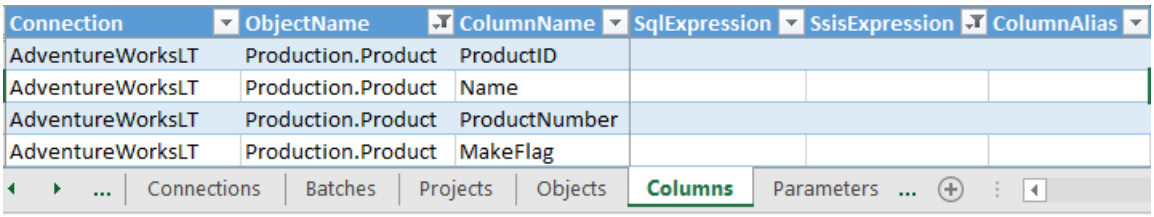
```
GROUP BY [CustomerID]
```

ObjectAlias

The ObjectAlias metadata column used on the Objects tab. A customer table can be aliased `cst` to use in custom Sql statements.

Column Overrides

BimlFlex supports a number of overrides on individual columns.



Connection	ObjectName	ColumnName	SqlExpression	SsisExpression	ColumnAlias
AdventureWorksLT	Production.Product	ProductID			
AdventureWorksLT	Production.Product	Name			
AdventureWorksLT	Production.Product	ProductNumber			
AdventureWorksLT	Production.Product	MakeFlag			

SqlExpression

`SqlExpression` is used to apply a transformation to a column in the query. This can apply any valid SQL expression to the column query extracting the data.

Example:

```
CONVERT(VARCHAR(60), [City])
```

SsisExpression

SsisExpression is used to apply a Derived Column transformation during extraction. This can apply any valid SSIS expression to the column in the data flow. Example:

```
(DT_STR, 60, 1252)ALT_CITY
```

Column alias

Column alias defines the SQL alias for a source query.

Example, the source query needs to apply a rename of a column. BimlFlex will create the query using the original column name and add `AS [Alias]` to the query.

Example code:

Adding the Alias

```
AlternateName
```

to the column

```
Name
```

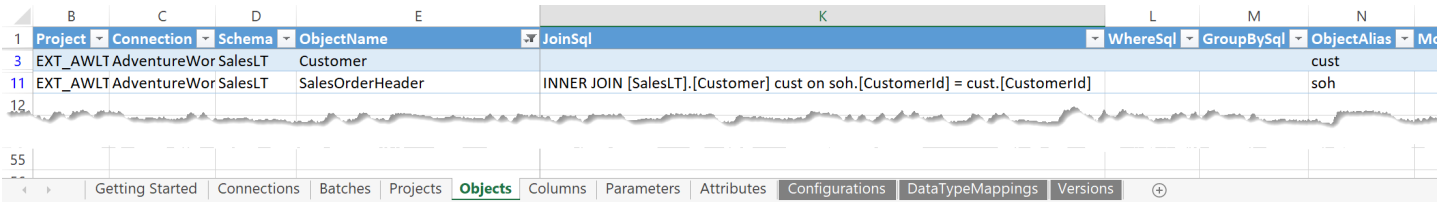
will generate the following SELECT snippet for the column

```
[Name] AS [AlternateName]
```

Example: Joining Tables in BimlFlex

Example of a join between the `SalesOrderHeader` and `Customer` tables to add columns from the `Customer` table to the `SalesOrderHeader` Staging and Persistent Staging through an `INNER JOIN`. This will add the join and any specified additional columns into the source `SELECT` statement.

Objects in metadata

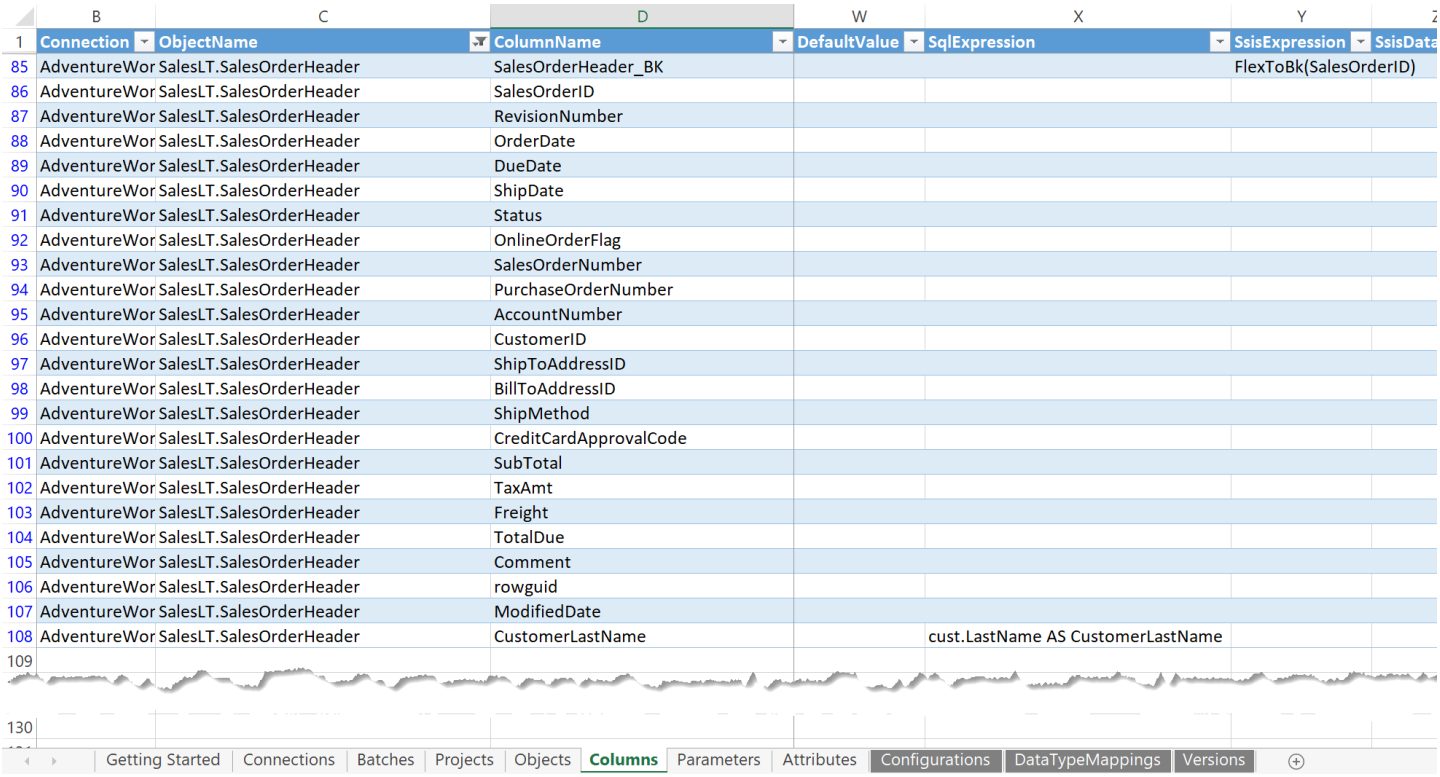


	B	C	D	E	K	L	M	N
1	Project	Connection	Schema	ObjectName	JoinSql	WhereSql	GroupBySql	ObjectAlias
3	EXT_AWLT	AdventureWor	SalesLT	Customer				cust
11	EXT_AWLT	AdventureWor	SalesLT	SalesOrderHeader	INNER JOIN [SalesLT].[Customer] cust on soh.[CustomerId] = cust.[CustomerId]			soh
12								
55								

Getting Started | Connections | Batches | Projects | **Objects** | Columns | Parameters | Attributes | Configurations | DataTypeMappings | Versions

Example: `ObjectAlias` and `JoinSql` added to metadata

Columns in metadata



	B	C	D	W	X	Y
1	Connection	ObjectName	ColumnName	DefaultValue	SqlExpression	SsisExpression
85	AdventureWor	SalesLT.SalesOrderHeader	SalesOrderHeader_BK			FlexToBk(SalesOrderID)
86	AdventureWor	SalesLT.SalesOrderHeader	SalesOrderID			
87	AdventureWor	SalesLT.SalesOrderHeader	RevisionNumber			
88	AdventureWor	SalesLT.SalesOrderHeader	OrderDate			
89	AdventureWor	SalesLT.SalesOrderHeader	DueDate			
90	AdventureWor	SalesLT.SalesOrderHeader	ShipDate			
91	AdventureWor	SalesLT.SalesOrderHeader	Status			
92	AdventureWor	SalesLT.SalesOrderHeader	OnlineOrderFlag			
93	AdventureWor	SalesLT.SalesOrderHeader	SalesOrderNumber			
94	AdventureWor	SalesLT.SalesOrderHeader	PurchaseOrderNumber			
95	AdventureWor	SalesLT.SalesOrderHeader	AccountNumber			
96	AdventureWor	SalesLT.SalesOrderHeader	CustomerID			
97	AdventureWor	SalesLT.SalesOrderHeader	ShipToAddressID			
98	AdventureWor	SalesLT.SalesOrderHeader	BillToAddressID			
99	AdventureWor	SalesLT.SalesOrderHeader	ShipMethod			
100	AdventureWor	SalesLT.SalesOrderHeader	CreditCardApprovalCode			
101	AdventureWor	SalesLT.SalesOrderHeader	SubTotal			
102	AdventureWor	SalesLT.SalesOrderHeader	TaxAmt			
103	AdventureWor	SalesLT.SalesOrderHeader	Freight			
104	AdventureWor	SalesLT.SalesOrderHeader	TotalDue			
105	AdventureWor	SalesLT.SalesOrderHeader	Comment			
106	AdventureWor	SalesLT.SalesOrderHeader	rowguid			
107	AdventureWor	SalesLT.SalesOrderHeader	ModifiedDate			
108	AdventureWor	SalesLT.SalesOrderHeader	CustomerLastName		cust.LastName AS CustomerLastName	
109						
130						

Getting Started | Connections | Batches | Projects | Objects | **Columns** | Parameters | Attributes | Configurations | DataTypeMappings | Versions

Example: added `CustomerLastName` column joined from the `Customer` table.

This manually added column uses the following SqlExpression to inject the column from the joined table using the alias defined for that table as well as a custom name in the `SalesOrderHeader` Staging table.

SqlExpression:

```
cust.LastName AS CustomerLastName
```

Example, generated source query:

```
SELECT
     soh.[SalesOrderID]
     -- several excluded columns
     ,soh.[ModifiedDate]
     ,cust.LastName AS [CustomerLastName]
FROM [SalesLT].[SalesOrderHeader] soh
     INNER JOIN [SalesLT].[Customer] cust on soh.[CustomerId] = cust.[CustomerId]
```

Steps to create a source query join using two tables that are included in metadata are as follows:

1. Determine which table (A) will be used to join against an already existing table (B)
2. Ensure that both tables A and B both have an object alias set in the object's metadata
3. Add a join statement into the JoinSql column of table A ensuring that the correct object alias is being used for both
4. If additional columns are added to the load add them to the columns sheet

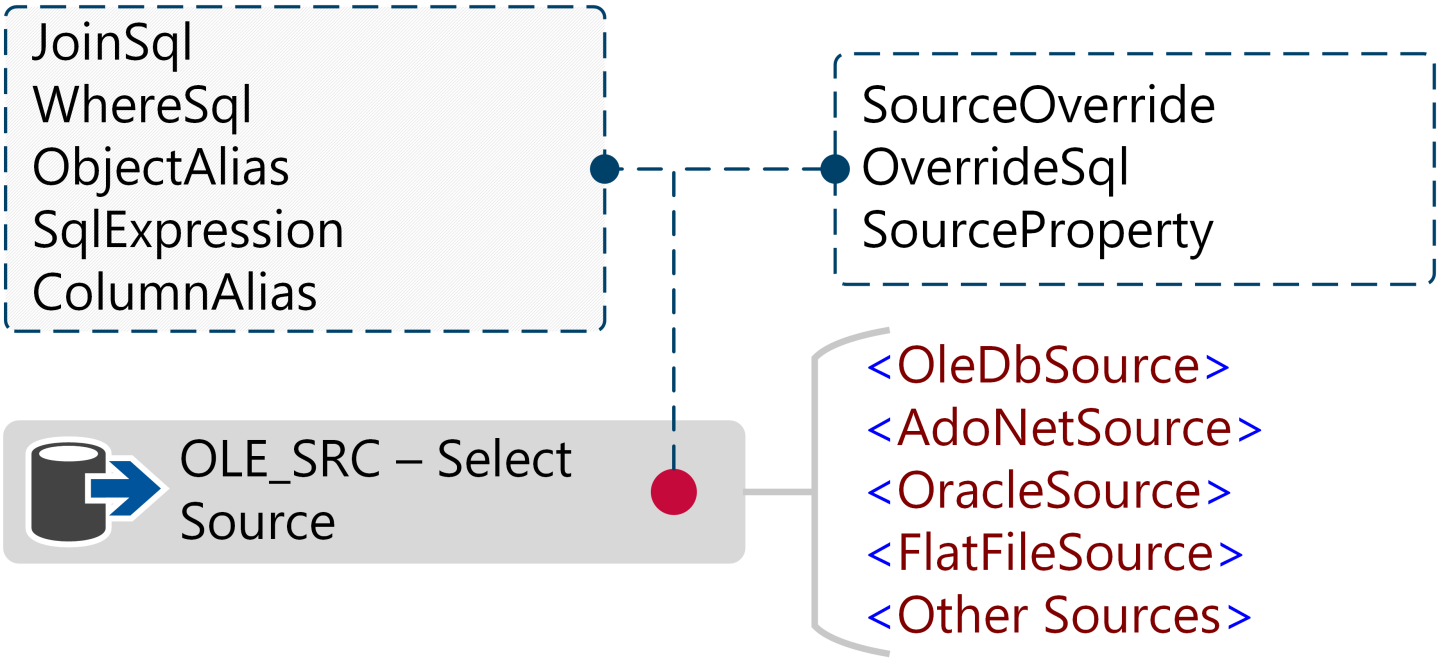
5. Set the changes on both Object and column metadata sheets
6. Check the output in BimlStudio is as expected
7. Build and test

Custom Source Queries with Extension Points

Another option for customizing source queries is by adding an Extension Point. Extension Points are a comprehensive feature of BimlFlex and has its own documentation here. A subset focused on Source to Staging is included here.

Extension points are custom logic and tasks that can be injected directly into the BimlFlex framework in order to extend standard BimlFlex functionality.

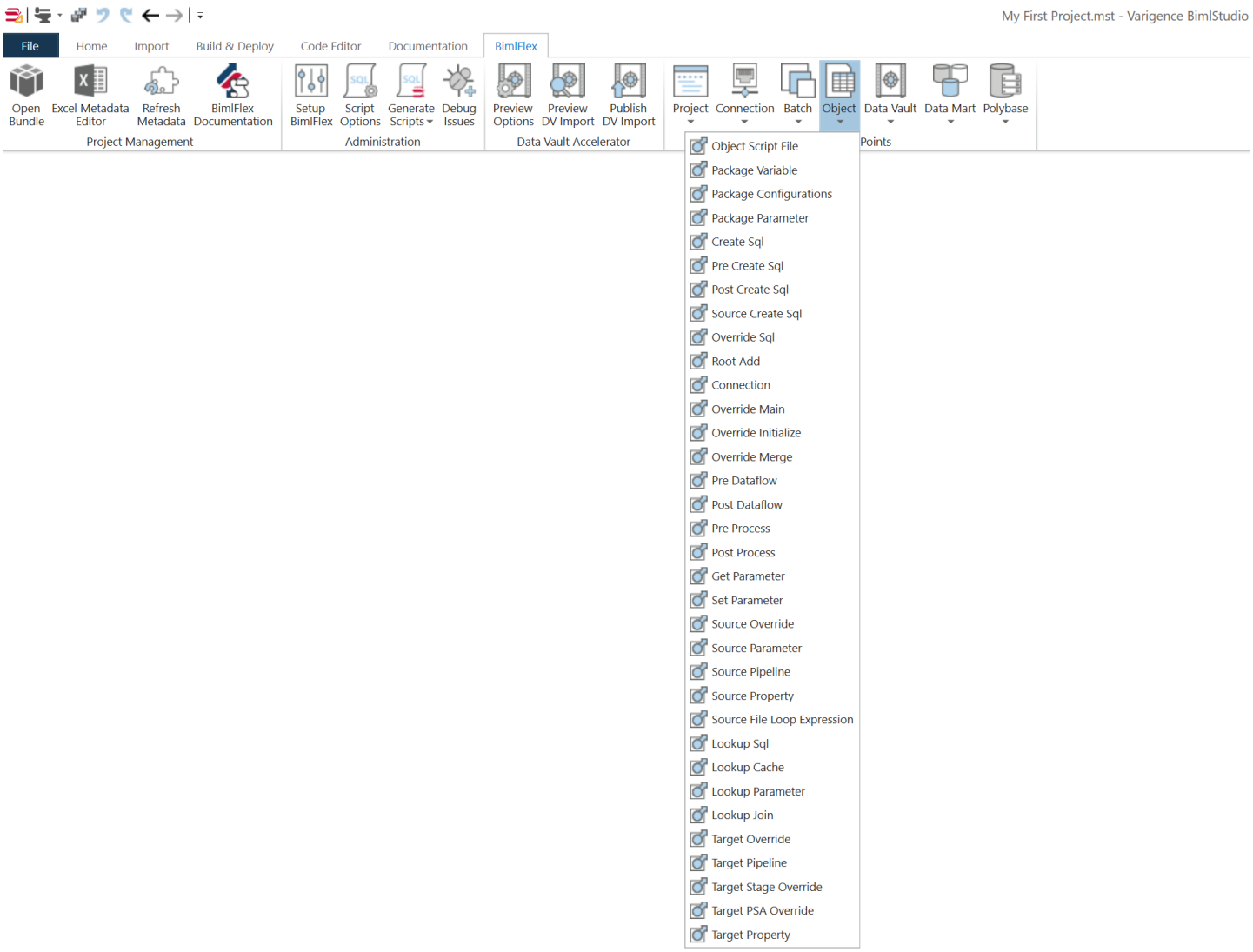
The diagram below shows at which point the extension points are added to the source component ETL in BimlFlex. The left-hand side box contains the adjustments a user can make to the source query in metadata and the right-hand box contains the extension points that can be added that control the source query.



Create a Source Override Extension Point

Example, create a source override extension point.

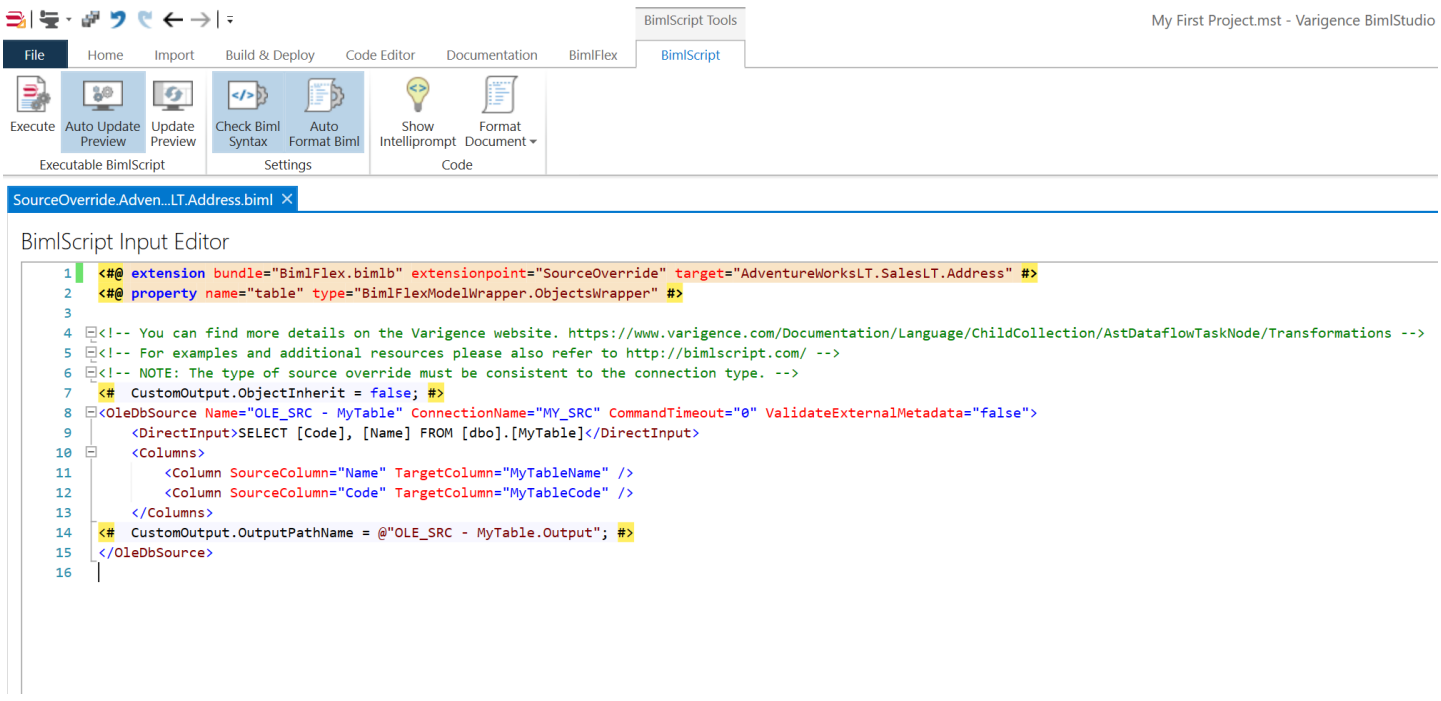
In BimlStudio, click the Source Override in the BimlFlex Tab, Extension Points group, Object drop-down to create the Extension Point Biml file.



In the new Biml file that opens, add the required target attribute for the source this will override, such as `AdventureWorksLT.SalesLT.Address` for the source connection, schema and table.

Rename the added Extension Point file so that it is easy to identify it later.

Add the custom Sql query and the resulting columns to the code.



Load Threading

BimlFlex supports optimising the ETL processes by selectively adding multi-threading to packages.

If threading is applied to the ETL pattern, this will divide the amount of rows being processed across the available cores provided by the server. This can results in performance improvements.

To adjust, edit NumberOfThreads on the Connection and Object tabs. Determine the number of threads to allocate using the BalanceDistributor component and add the value as appropriate. The BalanceDistributor is the component in SSIS that allows threading to occur. Setting the NumberOfThreads to two will alter the **[DFT – Extract Source]** data flow. 0 represents the default of 1 thread.

Load Hashing

Hashing of incoming rows also allows for potential performance increases. Creating row hashes minimizes the number of comparisons that are required in the delta process.

[FLX – Full Row Hash]

Hashes all the columns in the row (using a double SHA-1 hash to eliminate hash collisions).

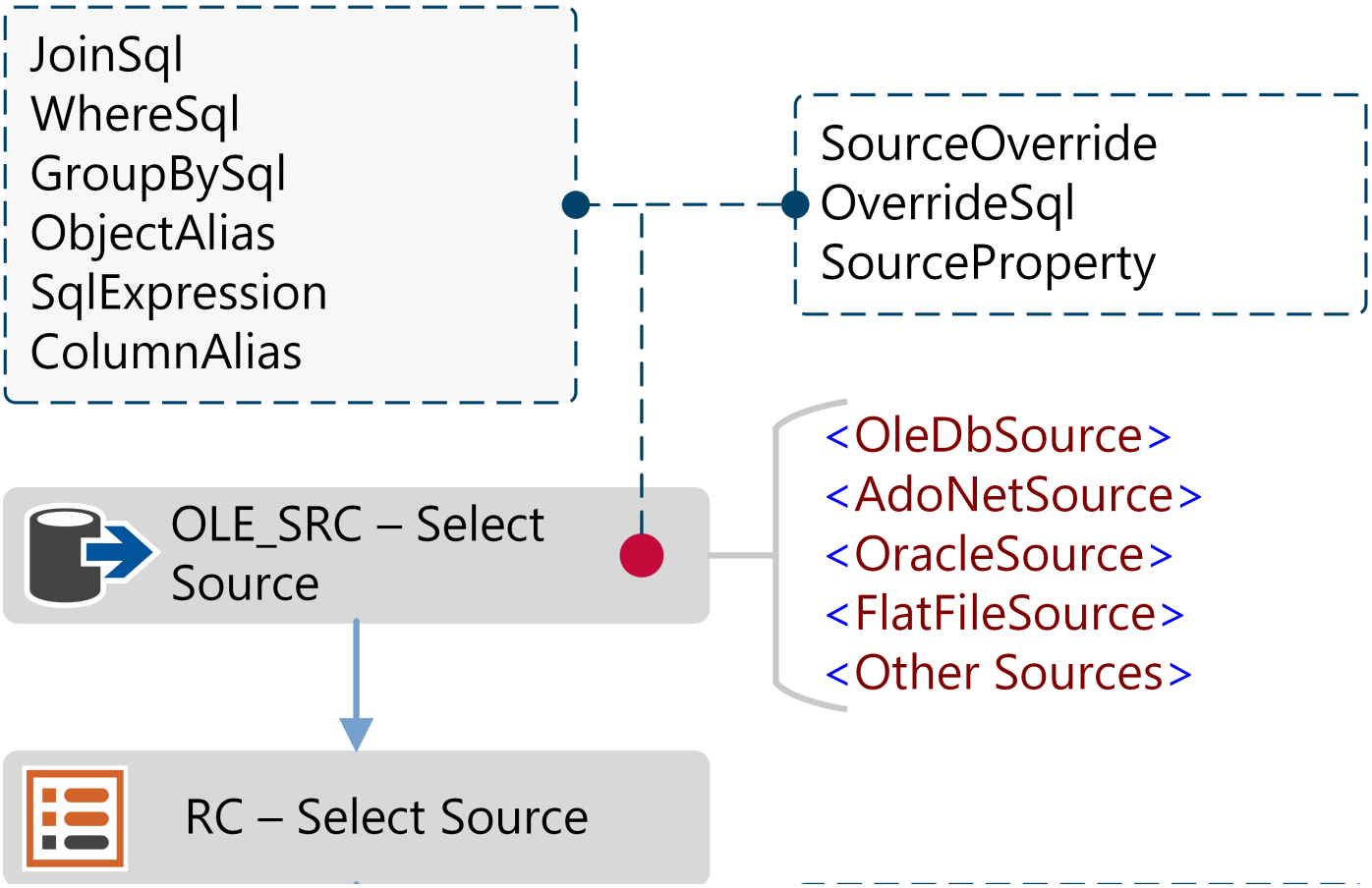
BimlFlex uses this for delta detection in the staging layer (comparing loads to Persistent Staging) for sources that do not have built-in CDC.

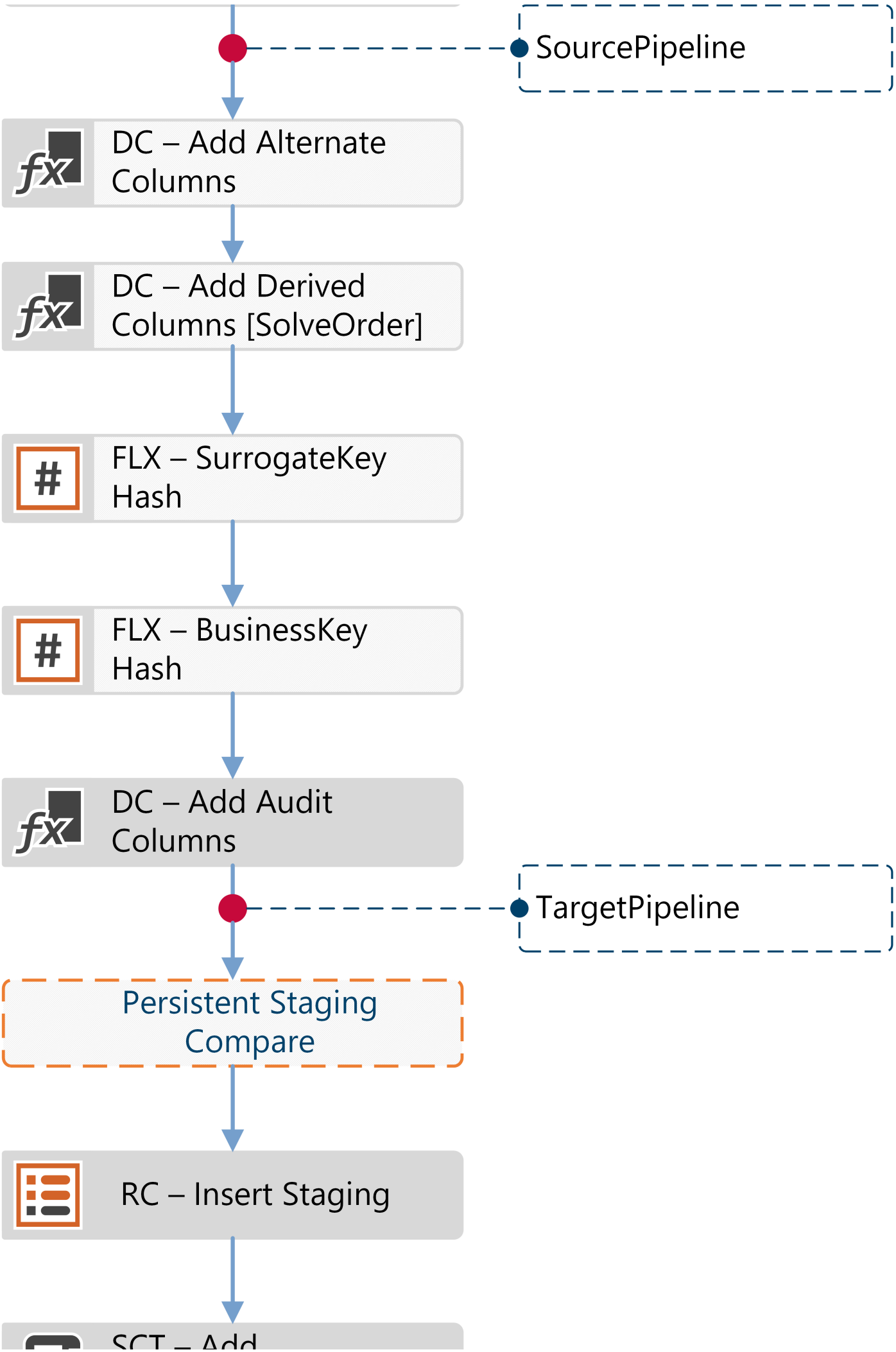
[FLX – BusinessKey Hash]

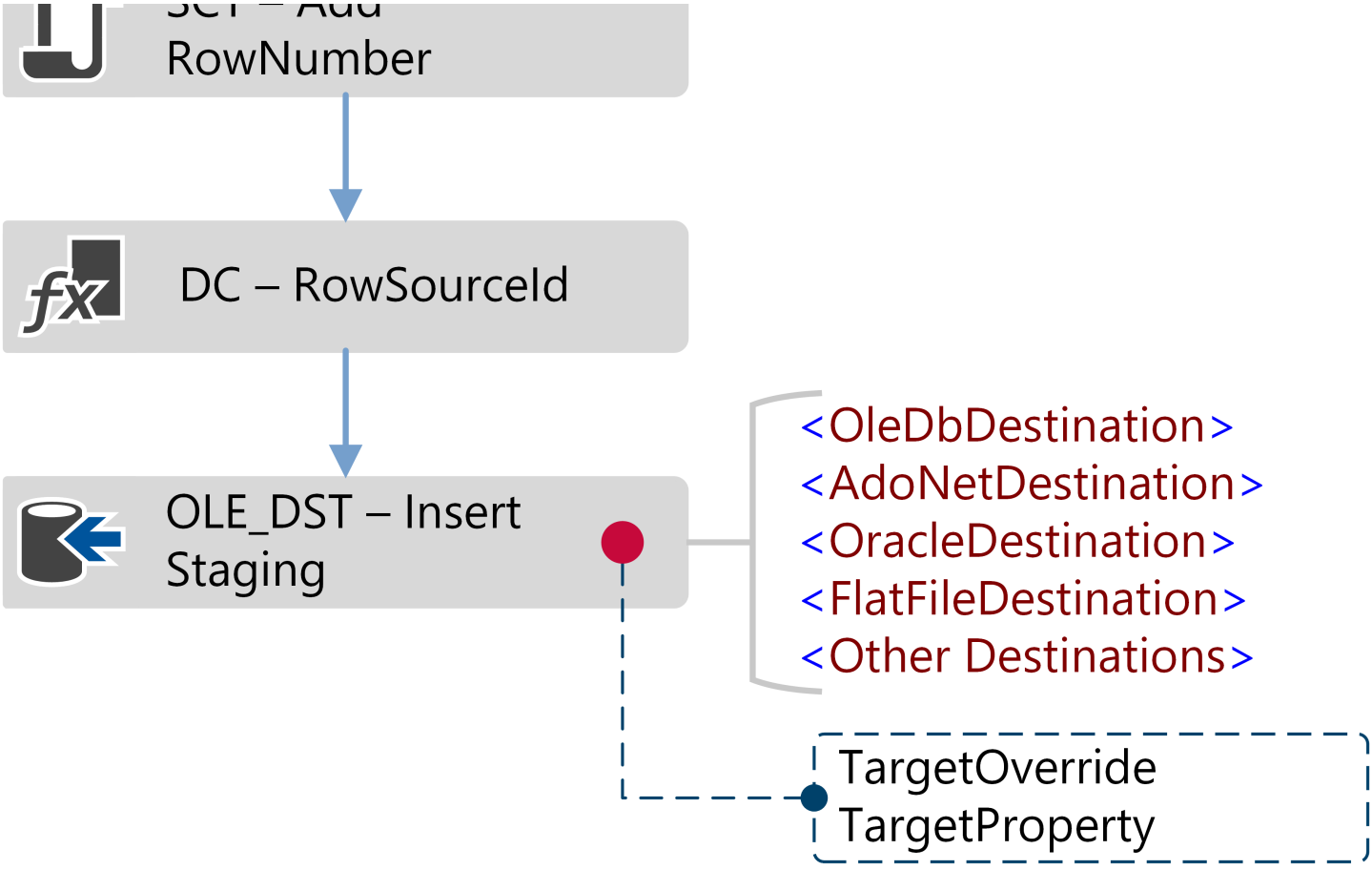
Hashing the Business Key for use in Data Vault and MPP systems like Microsoft Azure SQL Data Warehouse.

Configuring hashing is done through the configurations page of the BimlFlex Excel Add-in. Note that some destinations override the default behavior.

Data Vault targets will always hash Business Keys as that is a requirement for Data Vault.

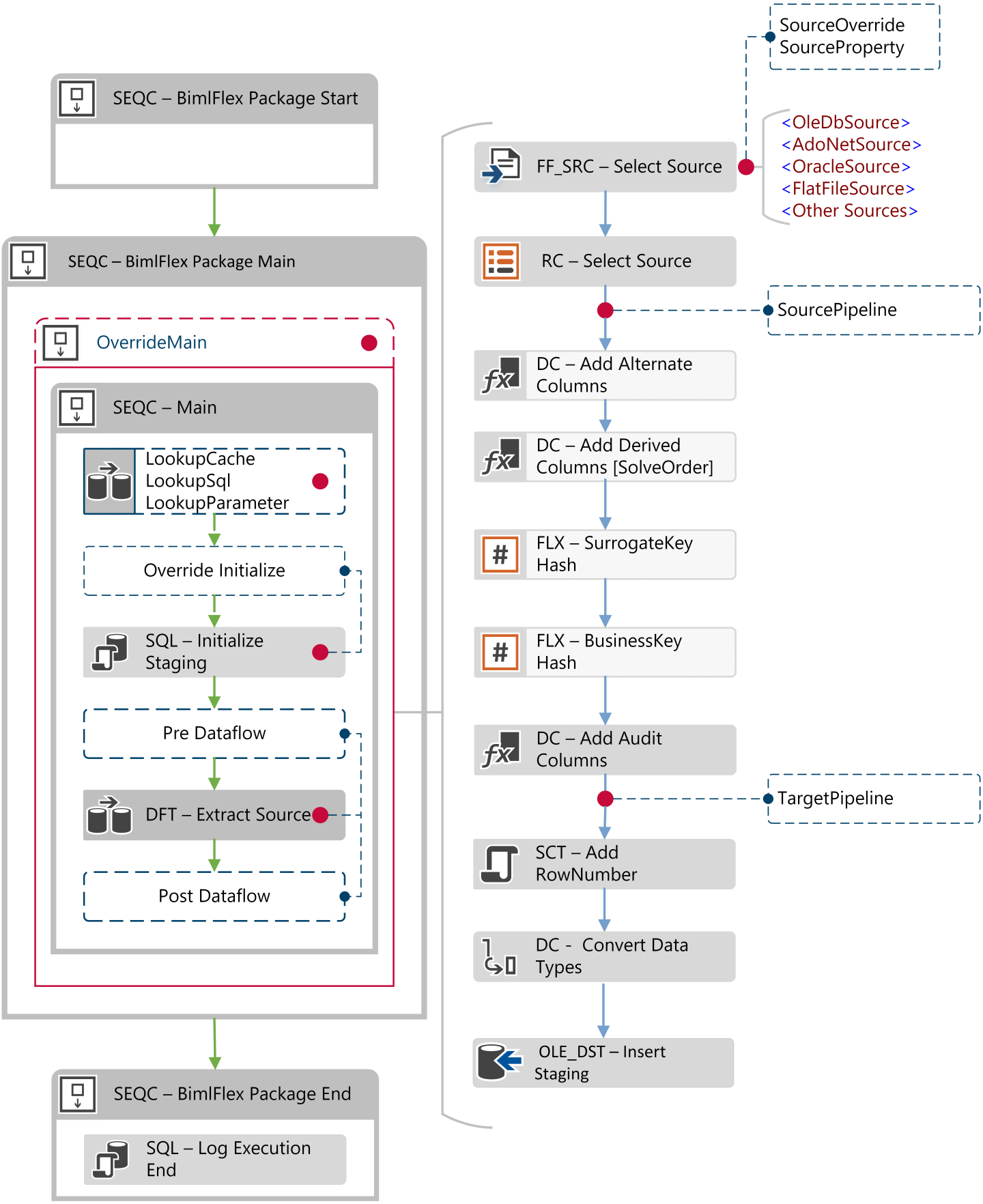



SourcePipeline
DC – Add Alternate Columns
DC – Add Derived Columns [SolveOrder]
FLX – SurrogateKey Hash
FLX – BusinessKey Hash
DC – Add Audit Columns
TargetPipeline
Persistent Staging Compare
RC – Insert Staging



File to Staging

For file based sources there are some additional configurations required. As the file doesn't provide the required metadata, the Objects and Columns sheets needs to be populated manually.



Adding File to Staging Metadata

Each source File requires a flat file connection on the Connections tab in BimlFlex Excel. This will define the attributes for the file.

Example attributes for a flat file connection:

ATTRIBUTE NAME	EXAMPLE PROJECT ATTRIBUTES
Name	CustomerFile
ConnectionString	C:\ImportFiles
Catalog	CustomerFiles
ConnectionType	FILE
SystemType	"File Delimited", "File Ragged Right", "Excel"
IntegrationStage	Source
RecordSource	TF
FilePath	C:\ImportFiles|
FilePattern	Customers*.*

Flat file loads also need a Batch entity. Add a new Batch with appropriate name and configuration.

Each file will have its own Batch. It is possible to group multiple Batches using the Parent Batch entry so that several files are loaded in a single Batch.

Flat file loads also need a Project entity. Add a new Project with appropriate name and configuration.

Flat files also need a file specification defined in the Objects sheet. File Type, first row column names, delimiters and codepage/encoding needs to correspond to the format of the files.

Each column in the source needs a column definition in the Columns sheet. The only requirement in this step is that the table used as the target for the customer rows has the correct number of columns with matching data types.

Refresh the solution metadata in BimlStudio, build and test the solution.

Persistent Staging

Adding Persistent Staging

Persistent staging can be seamlessly added to a project. Similar to how BimlFlex handles staging in a project, BimlFlex will automatically generate all the required table schemas and ETL needed to include persistent staging.

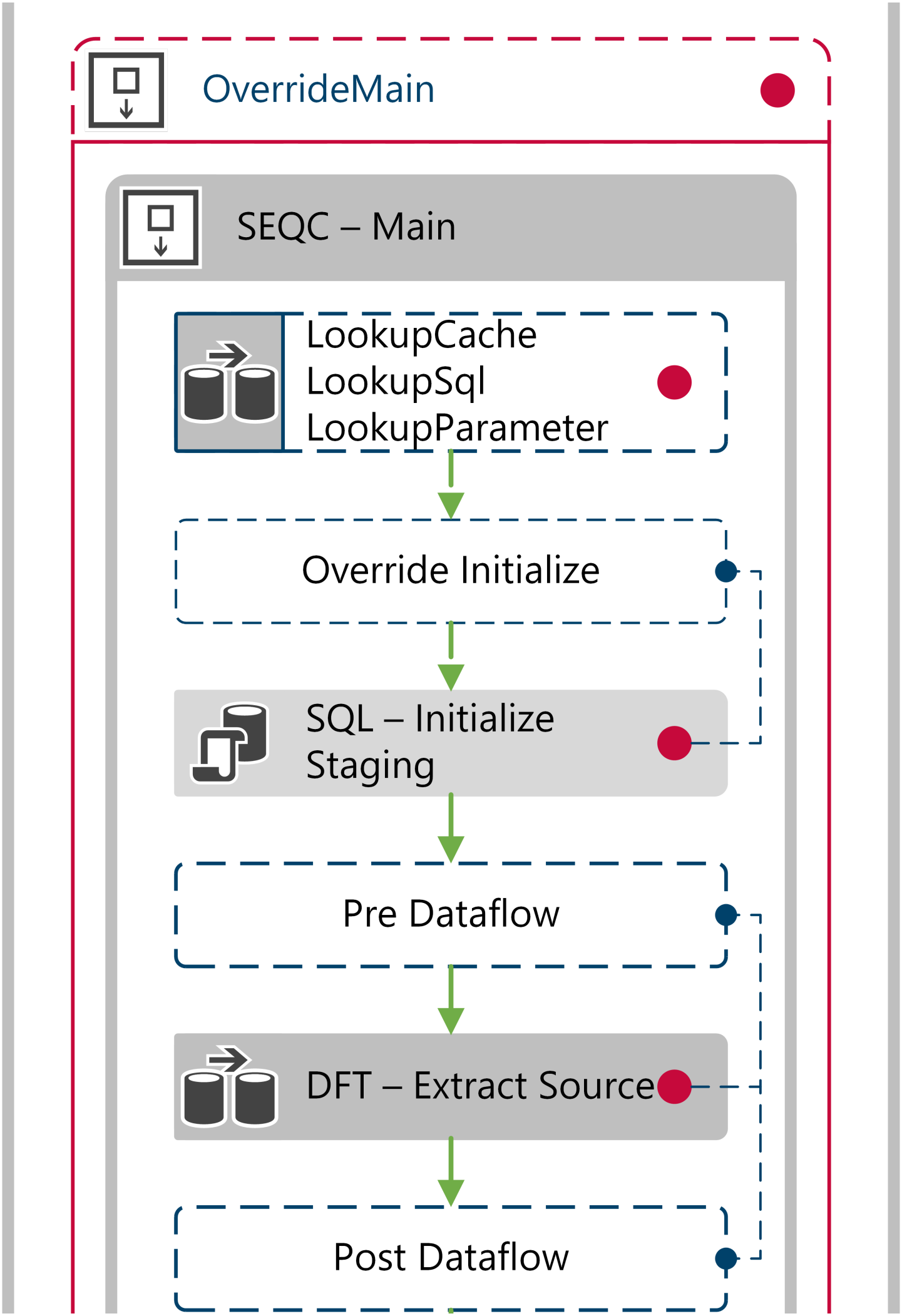
Delta/change detection for the Persistent layer is done on the defined primary keys of the source.

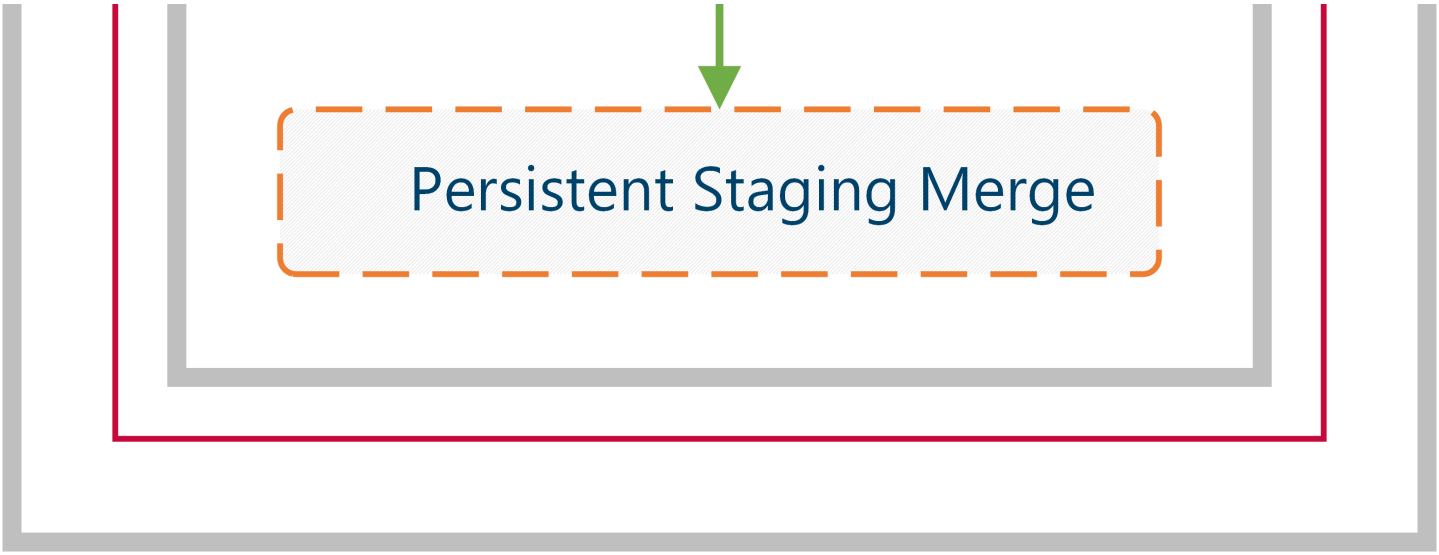
All the required information is already in place and can be inferred from the staging and source table metadata.

Creation and inclusion of Persistent Staging loads is controlled by the PersistHistory Flag for the connection.

By configuring Y for the source the ETL process and table definitions are included for the source.

SEQC – BimlFlex Package Main


OverrideMain
SEQC – Main
LookupCache
LookupSql
LookupParameter
Override Initialize
SQL – Initialize Staging
Pre Dataflow
DFT – Extract Source
Post Dataflow



The above template includes the following

[LKP – PSA/ODS]

Lookup rows from Persistent Staging are used to perform CDC/Delta Detection. If the source system changes, users can override this by defining an ExtensionPoint for LookupJoin or LookupSql. The following query is generated by the template.

```
SELECT
     SRC.[RowHash]
    ,SRC.[RowRecordSource]
FROM
     [aw].[Address] SRC
<# if (table.GetPersistHistory()) { #>
WHERE
     SRC.[RowIsCurrent] = 1
<# } #>
```

[SQL – Insert PSA/ODS]

When `@IsInitialLoad` is set to `TRUE`, perform a direct insert into the PSA database bypassing the end dating and MERGE process.

[SQL – Merge PSA/ODS]

When `@IsInitialLoad` is set to `FALSE`, `MERGE` the changes into the PSA database.

[SQL – Update End Dates]

Updates relevant End Dates and Current Flags.

Persistent Staging to Staging

BimlFlex supports reloading the data warehouse using persisted staging as the source. The standard Build process automatically creates a reinitialisation project that reloads the Staging tables from Persistent Staging.

For the sample AdventureWworksLT source, the project `INIT_FROM_PSA_EXT_AWLT_Project` is generated.

BimlFlex Data Vault Accelerator

On Data Vault modelling

This guide provides information on the Accelerator but assumes a sound understanding of the Data Vault modelling approach.

Watch Recordings

04. BimlFlex - Data Vault Accelerator

In this session, we discuss the various modelling and configuration options available with BimlFlex.

Introduction to the Accelerator

The Accelerator provides a quick-start opportunity with a best effort, technical modelling of Data Vault constructs out of the source metadata. It is configurable and provides a preview that can be rerun as many times as necessary so that the initial Data Vault modelling can be completed faster than through manual metadata modelling.

It is important to remember that the Data Vault modelling approach is based around **Core Business Concepts** (CBC) that are built upon **Enterprise Wide Business Keys** (EWBK) to allow for intra-systems integrations. The Accelerator does it's best to derive this from the source metadata but there is a need for a modeler with enterprise knowledge to translate the data and events into a source system agnostic model. The Accelerator makes it easy to get started and iterate through variations so that the analysts and subject matter experts can validate and tune the model to best match business processes.

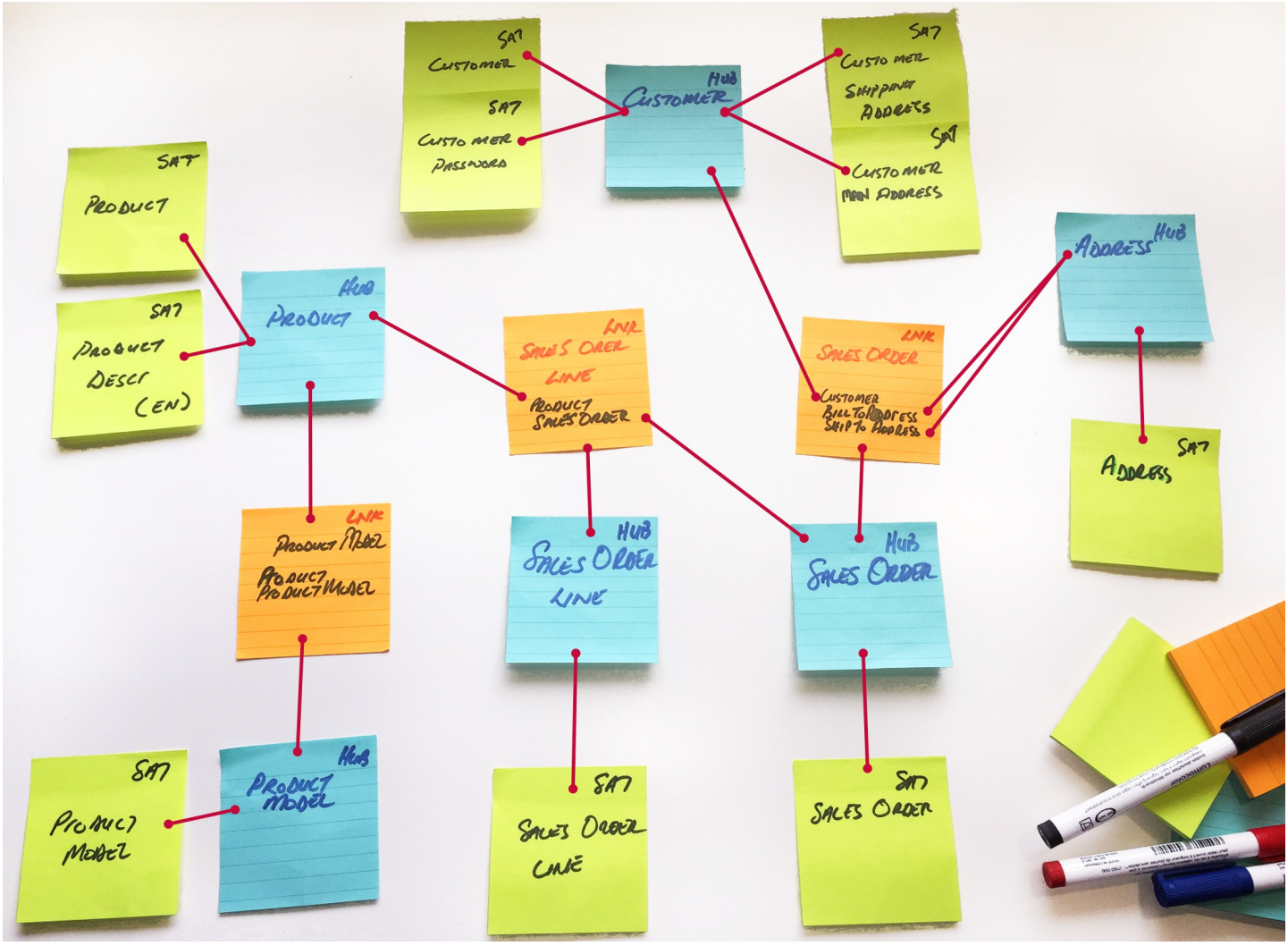
Starting Point

The starting point for the examples in this document is when all source metadata has been imported for the AdventureWorks LT database, the Source to Staging and Persistent Staging has been completed and it is time to start integrating the staged data into the Raw Data Vault.

Follow the guide for Source To Staging and import all SalesLT tables from the AdventureWorksLT source.

The Source and Target model

Before starting the integration and acceleration of the technical artefacts, it is important to have an understanding of the expected target model and how the data from the source model can be loaded into this target. The source model is defined by the source. In most cases, the source model is not directly transposable to the target model so some analysis and modelling are required. The technical implementation in the source is then tweaked to match the expected Data Vault model.



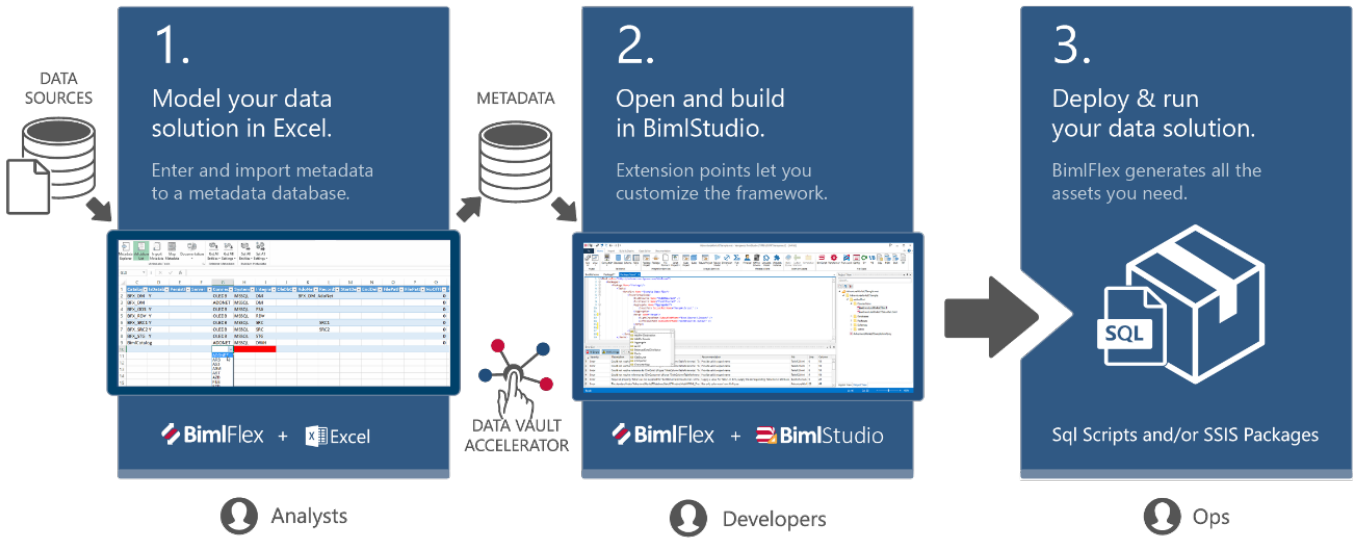
Once the target model is drafted, compare the accelerated metadata with the expected outcome and tweak accordingly.

Prerequisites

This guide builds on the other guides in the series and assumes that the environment has been set up and configured, source metadata has been imported and that the source to staging process has been modelled and completed.

The BimlFlex Workflow

The Accelerator integrates into and is part of the normal BimlFlex workflow



The workflow uses BimlFlex Excel and BimlStudio with BimlFlex to model and manage all metadata. The data can be pushed and updated from both Excel and BimlStudio.

Accelerate the Model

Based on the source metadata and the source database, the source model can be used to relate the data in the staging area. Based on information gained from the business process a user can derive a target data model that:

- Is source agnostic
- Is aligned with the business process
- Is based on Core Business Concepts
- Describes events and transactions through the relations between the Core Business Concepts.

The accelerator will use the metadata in the source for the project to derive the potential Hubs, Links and Satellites. The derived Data Vault model is available to preview and refine through editing the metadata and updating the preview.

The workflow where metadata is updated and tweaked and reprocessed through the Accelerator into the Data Vault preview allows a rapid design cycle of the Raw Data Vault.

Should the Accelerator not be needed it is possible to manually define the Data Vault artefacts and generate out the full model through the normal BimlFlex process. This is described in more detail in the guide for Raw and Business Data Vault.

Once the Accelerator preview matches the model expectations the metadata can be published to the repository and become part of the normal metadata set.

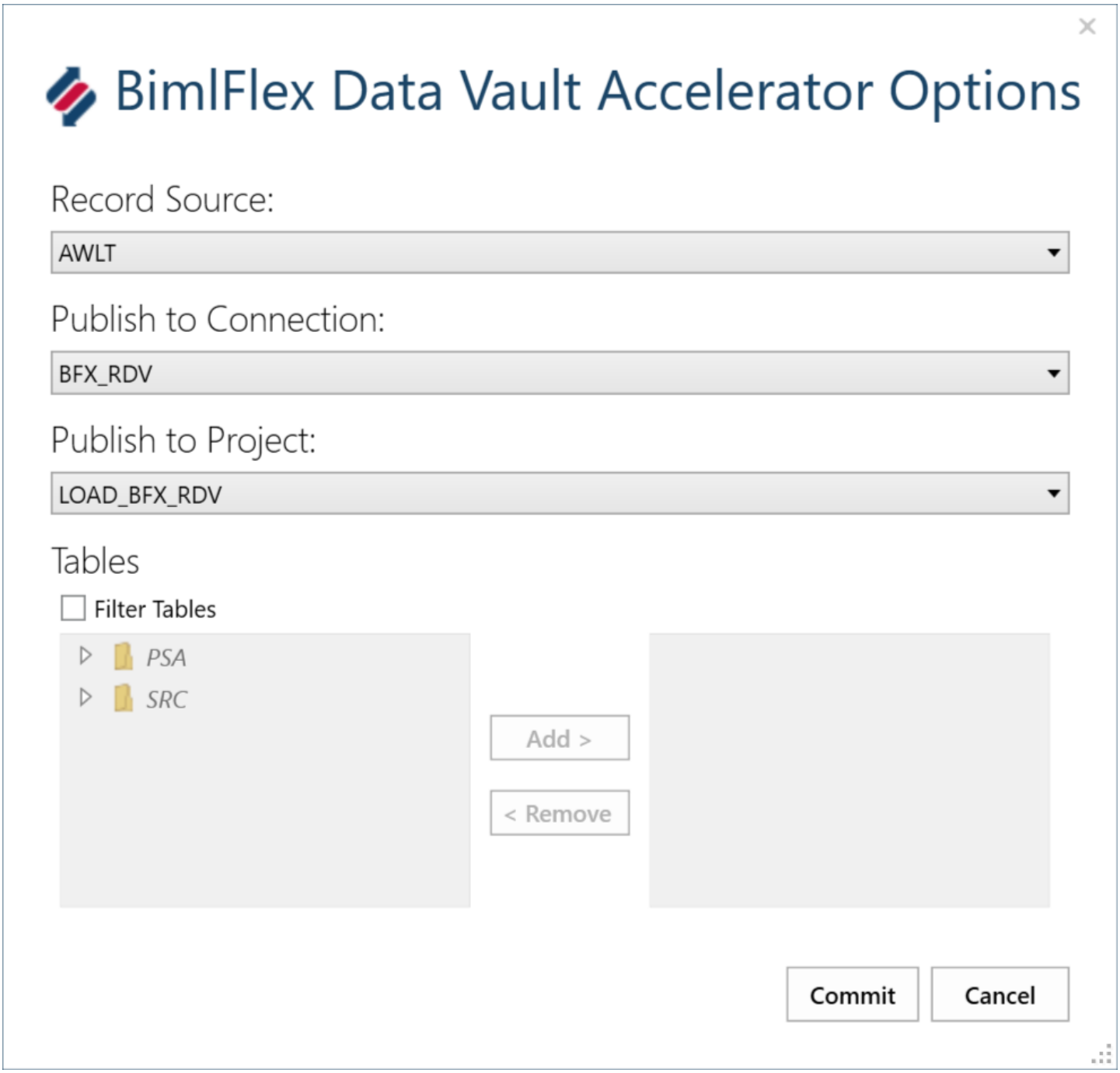
Sample workflow

The following narrative will guide us through the Accelerator process end to end. It is part of the overall BimlFlex workflow described earlier.

1. run the metadata import for the source system if not done already. This import identifies source tables as a base for either Hubs, Links or Satellites.
2. publish the metadata to the repository
3. refresh the metadata in BimlStudio
4. configure the Accelerator Preview Options to specify the record source, project and connection to use
5. preview the Accelerator through the table list in the solution explorer.
6. preview the Schema through the documentation option
7. Review and tweak the metadata
8. Rerun the preview
9. Publish the preview to the metadata repository
10. Build the database artefacts and SSIS artefacts from the newly published Data Vault metadata

The Accelerator User Interface

The Accelerator is integrated into the BimlFlex Ribbon on the BimlFlex tab in BimlStudio. Once the required base information is available for the Accelerator it can be configured and previewed directly in BimlStudio.

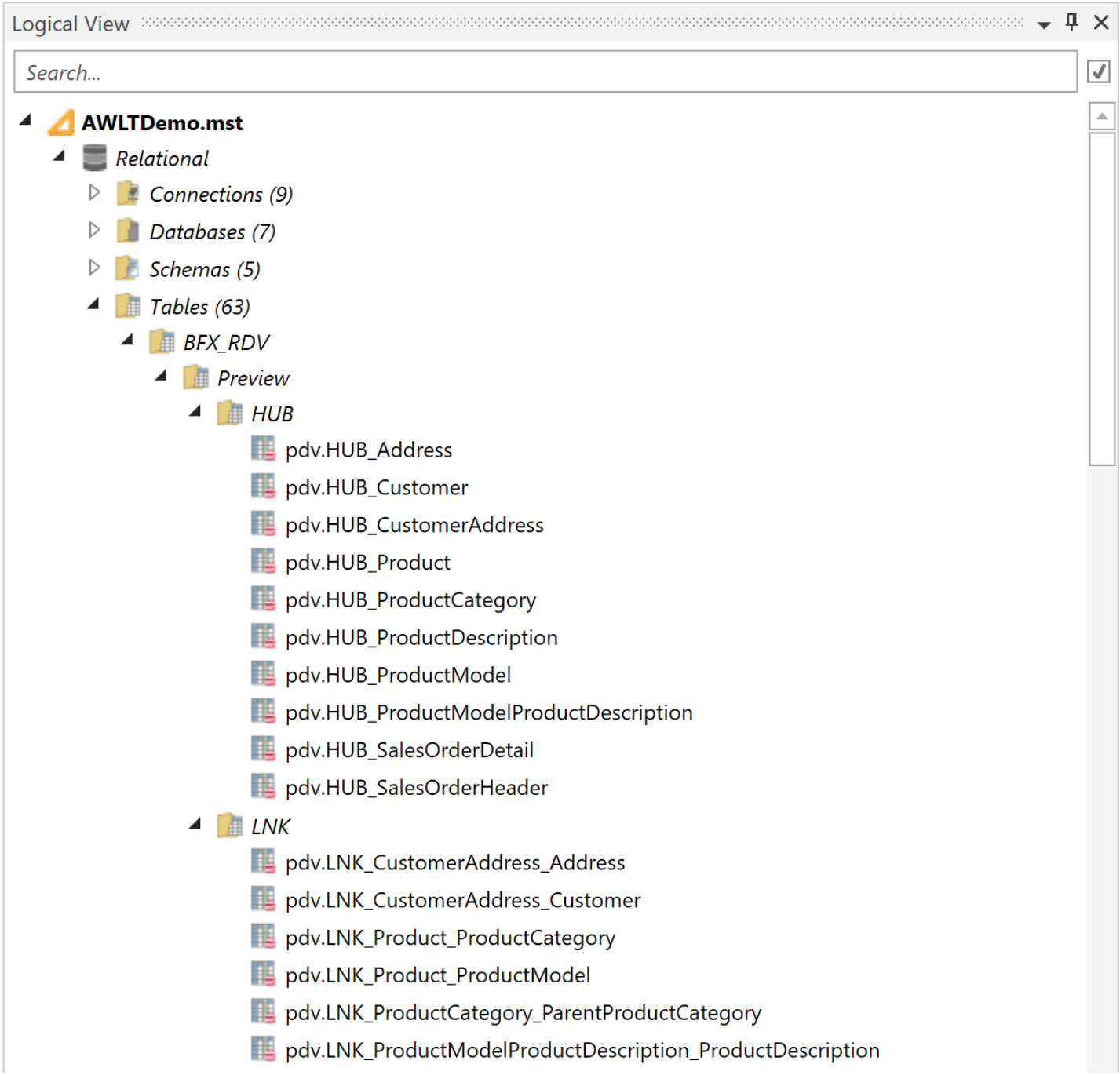


Clicking the Preview Options opens the BimlFlex Data Vault Accelerator Options dialog. By specifying the:

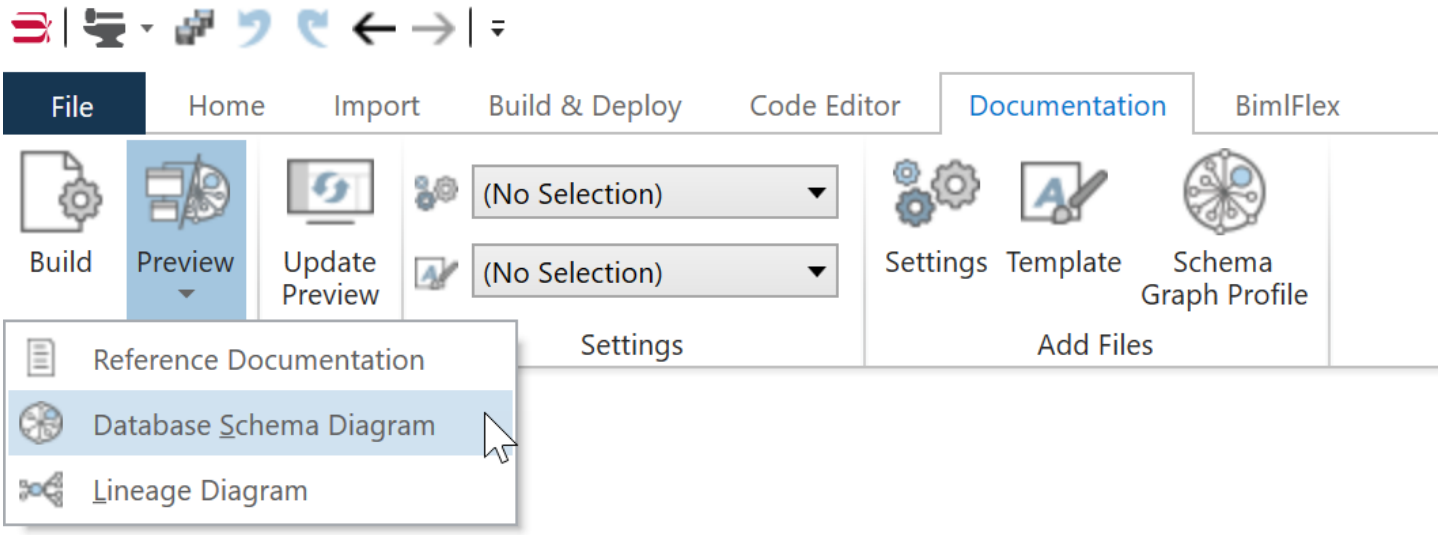
- Record Source (required for all Data Vault sources, specified in the Connections Tab in the Excel Metadata editor)
- Destination connection to publish the data to (any connection in the Connections Tab with a Raw Data Vault or Business Data Vault Integration Stage)
- Destination Project (a valid destination project for the chosen connection)

In the preview, it is also possible to filter the tables used for the preview from the source. This is useful where the full set of source tables have been included but the Data Vault is built piece by piece. Starting to source and persist changes from the source without having to consider the Data Vault process means the Staging part of the solution can be completed sooner.

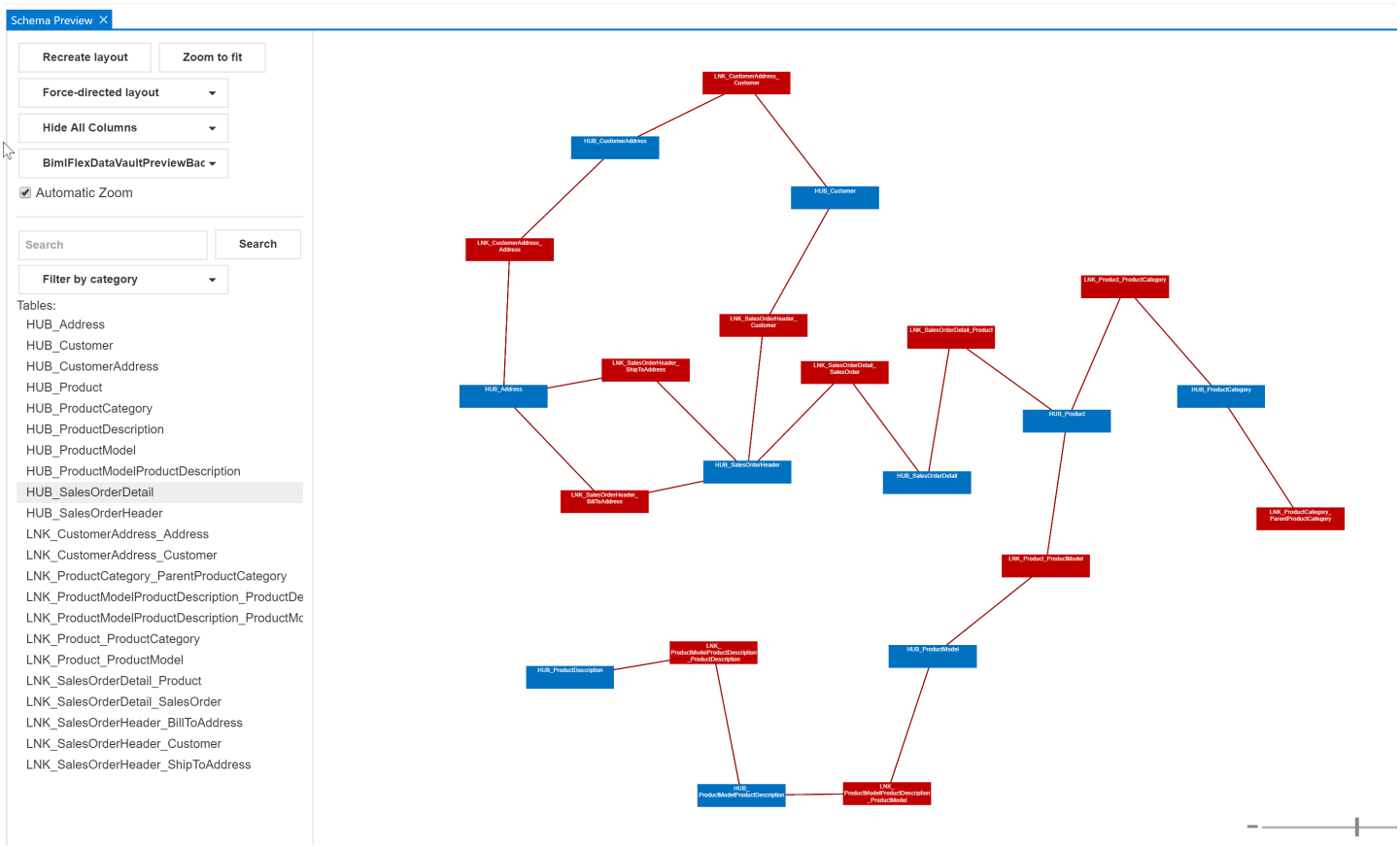
Constraining the Data Vault Acceleration to a subset allows for a more agile delivery where valuable parts of the solution can be put to good use as soon as they are done.



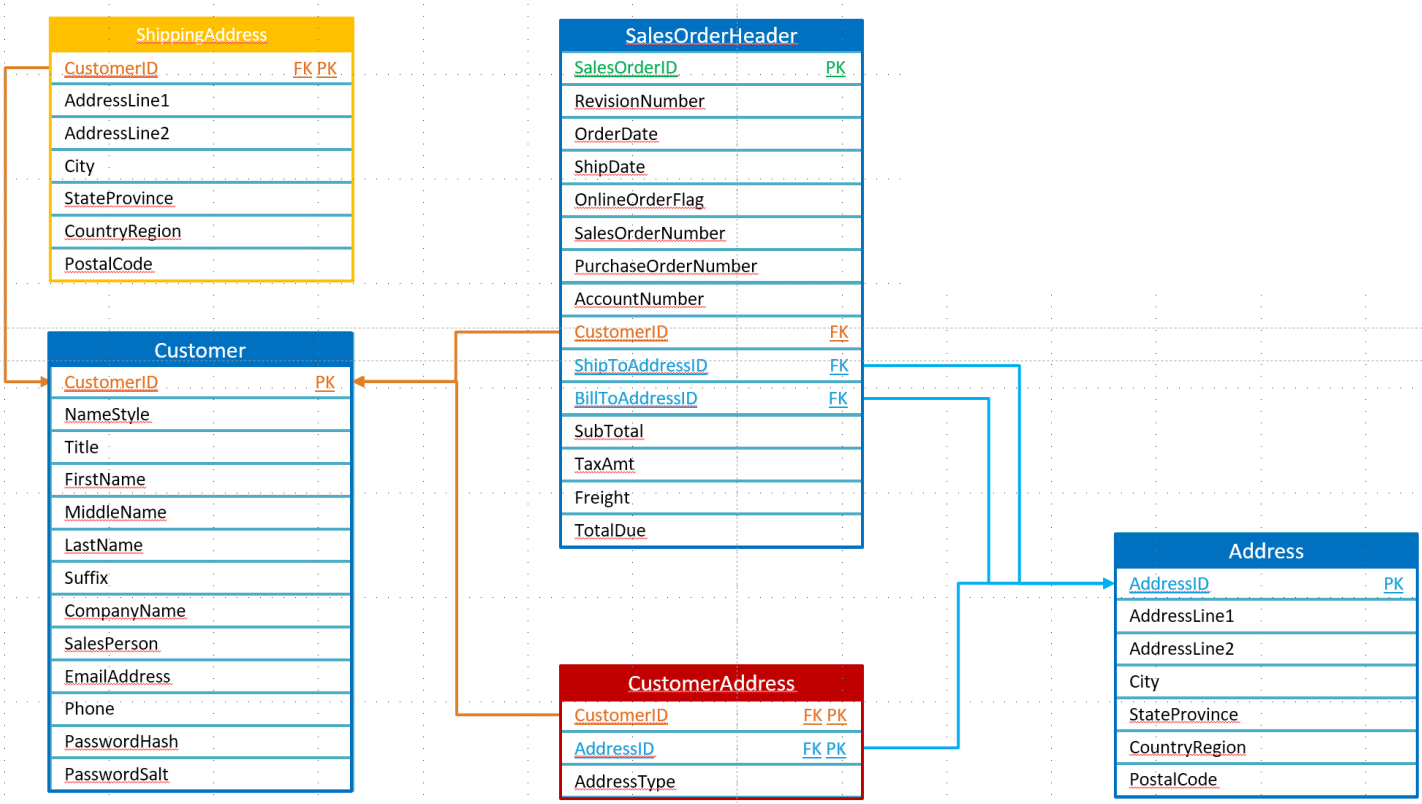
Once the Preview Options are defined and committed it is possible to Preview the Data Vault model. The preview will generate the destination table artefacts in the solution explorer to the right in BimlStudio.



Once the preview objects are available it is possible to review the information in the list of tables. It is also possible to generate a Database Schema Diagram that details the tables and their relationships in an easy to review diagram.



The default schema will include all tables from all stages/layers. By filtering the diagram to only include the Hubs and Links by choosing the BimlFlexDataVaultPreviewBackbone in the filtering pane a user can review the CBC's and UOW's. Anything that needs tweaking, can be updated in the metadata which will, in turn, produce a new schema from the updated information.



Iterating through the metadata and updating the model to better support the target Data Vault model, each time a user should set the updated metadata to be persisted to the Metadata Database. This will create all Data Vault tables and columns as well as the Source To Target mappings needed to populate the Data Vault from the chosen source.

If there is a need to update the model and rerun the preview, it can be done at any time. Once the metadata from the preview has been published, all metadata will be available in the Excel metadata editor and available for direct manipulation there. The pattern of tweaking the Metadata to fit the business process and target model will be repeated however many times needed before publishing.

Note that the accelerator will not resurface any entities already accelerated and marked as excluded or deleted. To see these entities, include them in the project by removing the excluded/deleted flag.

Updating the metadata to meet requirements

In the Excel-based Metadata Editor, there are numerous options for manipulating the source metadata so that the Accelerator will produce the desired Data Vault model. Some of the common requirements include:

- Choosing the Business Key used for the Hubs. By analysing business processes and the source data it is possible to find EWBK's that aren't the technical source keys
- Pulling disparate information stored in complex relationships in the source into a Satellite connected to the relevant Hub. For information, such as addresses there is normally no need to maintain complex relationships from the source. An address is just an attribute of the entity with a location
- Adjusting the grain in UOW's so that the correct Hubs are included in Links
- Separating out data into different Satellites based on rate of change, storage requirements or similar
- Reviewing Driving Key relationships for Links where there is no one FK relationship in the source

The Analyse Rules for BimlFlex Accelerator

There are a set of rules applied through the Accelerator to make it perform what it does.

The first rule applied is using the source metadata to derive the Table type. By using relationships defined in the source and imported through the source metadata import a user can run the following table logic.

Mark potential satellite tables

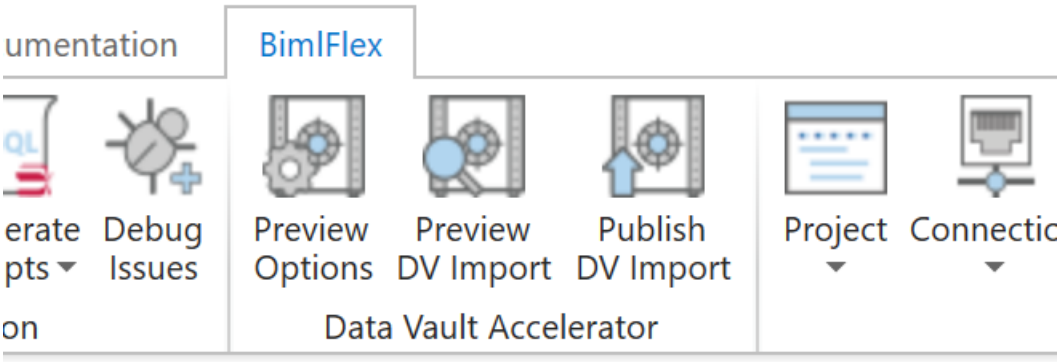
Every table with **no incoming foreign keys** and only **one outgoing foreign key** with all the referencing columns also **primary key** columns and no other columns are part of the primary key is a candidate to become a Satellite source table.

Mark as links

Every table with **no incoming foreign keys** and more than **one outgoing foreign key** with all the referencing columns also primary key columns is a candidate to become a Link source table.

Mark as hubs

Tables that don't fit any of the categories above are going to be Hub source tables.



If the source doesn't have relationships defined, such as flat files, these needs to be added to the metadata for the Accelerator to derive the table types.

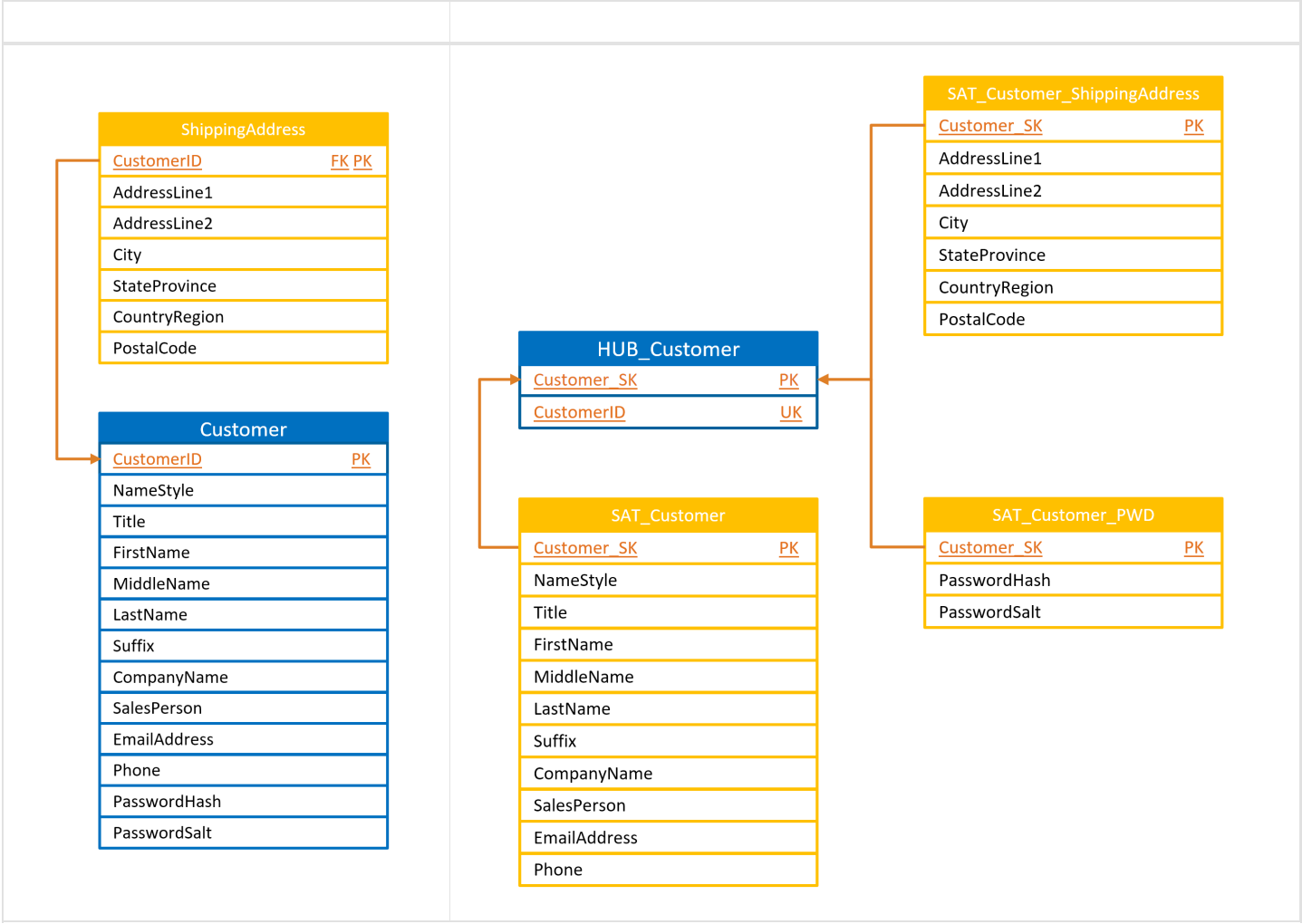
Acceleration

All acceleration uses default naming conventions so basic names for Hubs and Satellites will be derived automatically according to the Configurations

Accelerate Satellites

The first Acceleration step is to accelerate tables defined as Satellites. It will look for the defined business key and use it as the Surrogate Key for the connection to the Hub. Once the Key to use is defined and added as the hashed surrogate key the rest of the columns in the Satellite table are added as Satellite attributes.

It is possible to use the ModelGrouping metadata feature to split the source table data and group information into separate Satellites. This is useful for when part of the data in the source table changes more frequently. The ModelGrouping name will be added to the end of the Satellite name and the data from the source will go into this separate Satellite as well as the main Satellite.



The sample above shows how the `Customer` and `ShippingAddress` source tables are Accelerated into a `HUB_Customer` and three separate Satellites, one of them by adding `PWD` to the `ModelGrouping` for the password columns.

Accelerate Links

The Link tables in Data Vault are many to many relationship tables. Instead of having one-way foreign keys in the entities the relationship between entities is defined in a separate table.

The tables identified as Links by the Accelerator will be similar. They already have the relationships between the source tables defined and technical keys to the surrounding tables. The Foreign Keys will be translated into Surrogate Keys linking to the respective Hubs.

All created Links will have an effectiveness Link Satellite added.

Any attributes left in the source Link Table will be added to the default Link Satellite. They can be separated out into their own Satellites if needed using the ModelGrouping override.

It's worth noting that if there are source system applied rules for relationships (such as a product can only be in one category) they will need to be specified as Driving Keys in the Accelerated Link unless the Accelerator can derive the behaviour from relationship constraints.

For Hub tables that have Foreign Key relationships defined the Accelerator will generate Links for each Foreign Key between the defined Business Keys. These Links will need to be reviewed as the UOW they describe might not align with the business process.

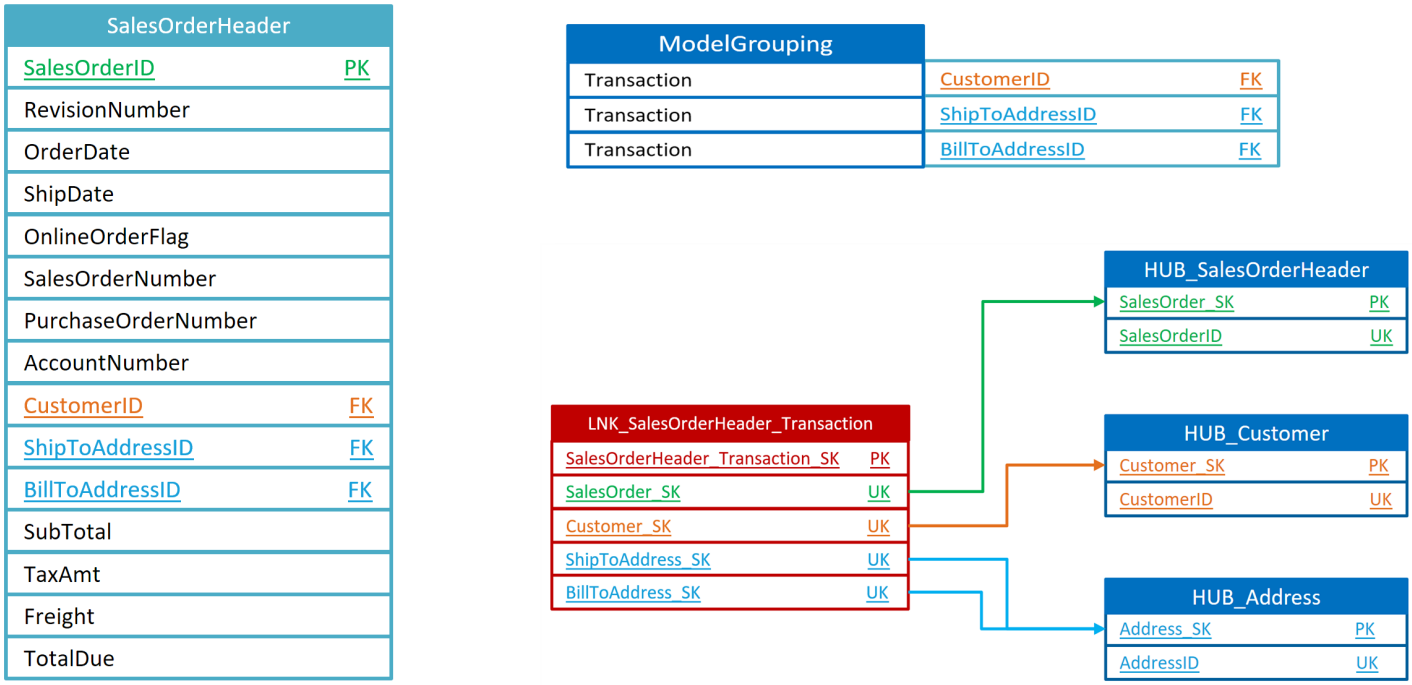
Accelerate Hubs

Any remaining tables and Business Keys will be used to create Hubs. The defined Business Key for the table will be used for the Hub Attribute and this attribute will be hashed into the Hub Surrogate Key.

Reviewing Unit of Work

The UOW defines the granularity of the relationship between the entities. It is vital that the granularity is correct and that it properly describes the business process a user is interested in.

Using the ModelGrouping attribute, a user can define the granularity from the source table to group multiple Foreign Key relations into a single Link

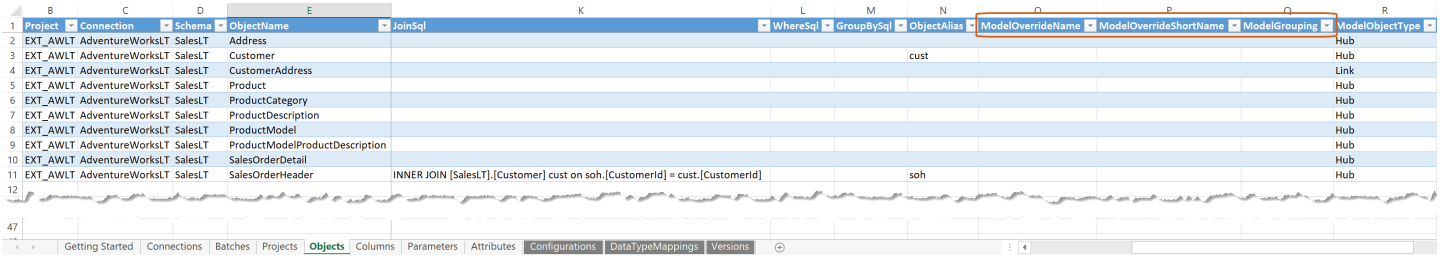


Adjusting Model Overrides

The columns in BimlFlex Excel that start with Model are used only by the Data Vault Accelerator. By updating the metadata with model information, a user can drive the behaviour of the Accelerator to generate the Data Vault model needed.

There are overrides for Objects as well as Columns

Object Overrides



Project	Connection	Schema	ObjectName	JoinSql	WhereSql	GroupBySql	ObjectAlias	ModelOverrideName	ModelOverrideShortName	ModelGrouping	ModelObjectType
EXT_AWLT	AdventureWorksLT	SalesLT	Address								Hub
EXT_AWLT	AdventureWorksLT	SalesLT	Customer				cust				Hub
EXT_AWLT	AdventureWorksLT	SalesLT	CustomerAddress								Link
EXT_AWLT	AdventureWorksLT	SalesLT	Product								Hub
EXT_AWLT	AdventureWorksLT	SalesLT	ProductCategory								Hub
EXT_AWLT	AdventureWorksLT	SalesLT	ProductDescription								Hub
EXT_AWLT	AdventureWorksLT	SalesLT	ProductModel								Hub
EXT_AWLT	AdventureWorksLT	SalesLT	ProductModelProductDescription								Hub
EXT_AWLT	AdventureWorksLT	SalesLT	SalesOrderDetail								Hub
EXT_AWLT	AdventureWorksLT	SalesLT	SalesOrderHeader	INNER JOIN [SalesLT].[Customer] cust on soh.[CustomerId] = cust.[CustomerId]			soh				Hub

Object ModelOverrideName

This column contains the actual table name to be used as after the solution is modelled.

If a source system has entity names like `GL002` and the actual business name is `GeneralLedger` the Accelerator will convert this to `[HUB_GeneralLedger]` and `[SAT_GeneralLedger]`

ModelOverrideShortName

This column contains the short name to be used for Links and Link Satellites after the solution is modelled.

A Link between `GeneralLedger` and `ChartOfAccounts` will use the default name `[LNK_GeneralLedger_ChartOfAccounts]`.

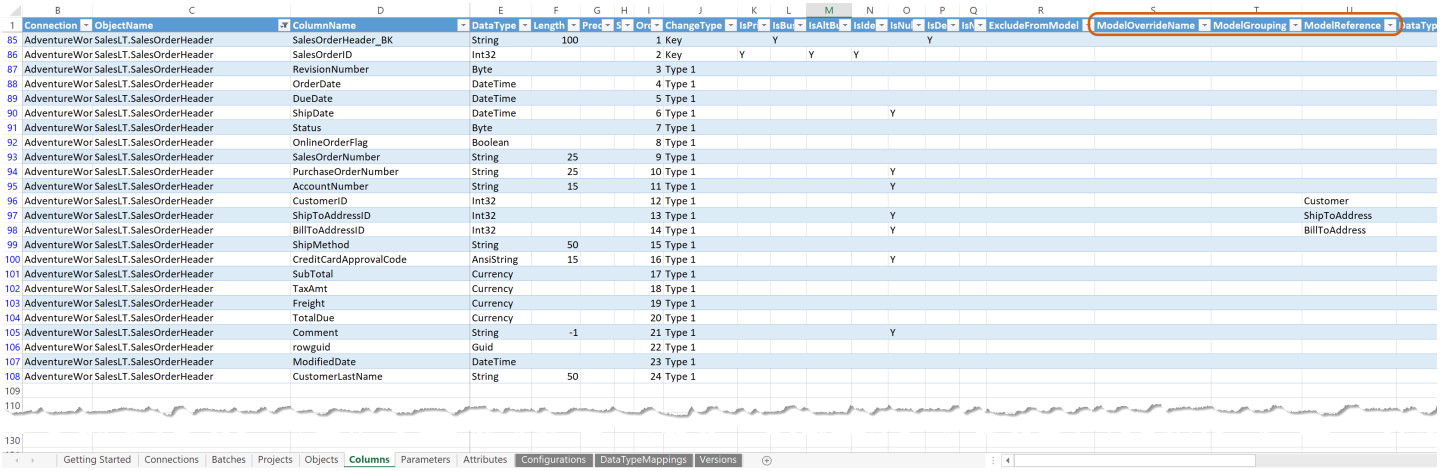
Defining the value `GL_COA` here will generate the name `[LNK_GL_COA]` providing easy naming flexibility through metadata when modelling more complex relationships.

ModelObjectType

Interpreted when importing metadata from source and can be overridden after additional analysis. The rules defined above uses the keys and relationships to derive Hub, Link or Satellite as the type.

Please note that this attribute is only relevant when using BimlFlex to generate a Raw Data Vault model.

Column Overrides



	Connection	ObjectName	ColumnName	DataType	Length	ChangeType	ModelOverrideName	ModelGrouping	ModelReference
85	AdventureWor	SalesLT.SalesOrderHeader	SalesOrderHeader_BK	String	100	1 Key			
86	AdventureWor	SalesLT.SalesOrderHeader	SalesOrderID	Int32		2 Key			
87	AdventureWor	SalesLT.SalesOrderHeader	RevisionNumber	Byte		3 Type 1			
88	AdventureWor	SalesLT.SalesOrderHeader	OrderDate	DateTime		4 Type 1			
89	AdventureWor	SalesLT.SalesOrderHeader	DueDate	DateTime		5 Type 1			
90	AdventureWor	SalesLT.SalesOrderHeader	ShipDate	DateTime		6 Type 1			
91	AdventureWor	SalesLT.SalesOrderHeader	Status	Byte		7 Type 1			
92	AdventureWor	SalesLT.SalesOrderHeader	OnlineOrderFlag	Boolean		8 Type 1			
93	AdventureWor	SalesLT.SalesOrderHeader	SalesOrderNumber	String	25	9 Type 1			
94	AdventureWor	SalesLT.SalesOrderHeader	PurchaseOrderNumber	String	25	10 Type 1			
95	AdventureWor	SalesLT.SalesOrderHeader	AccountNumber	String	15	11 Type 1			
96	AdventureWor	SalesLT.SalesOrderHeader	CustomerID	Int32		12 Type 1			Customer
97	AdventureWor	SalesLT.SalesOrderHeader	ShipToAddressID	Int32		13 Type 1			ShipToAddress
98	AdventureWor	SalesLT.SalesOrderHeader	BillToAddressID	Int32		14 Type 1			BillToAddress
99	AdventureWor	SalesLT.SalesOrderHeader	ShipMethod	String	50	15 Type 1			
100	AdventureWor	SalesLT.SalesOrderHeader	CreditCardApprovalCode	AnsiString	15	16 Type 1			
101	AdventureWor	SalesLT.SalesOrderHeader	SubTotal	Currency		17 Type 1			
102	AdventureWor	SalesLT.SalesOrderHeader	TaxAmt	Currency		18 Type 1			
103	AdventureWor	SalesLT.SalesOrderHeader	Freight	Currency		19 Type 1			
104	AdventureWor	SalesLT.SalesOrderHeader	TotalDue	Currency		20 Type 1			
105	AdventureWor	SalesLT.SalesOrderHeader	Comment	String	-1	21 Type 1			
106	AdventureWor	SalesLT.SalesOrderHeader	rowguid	Guid		22 Type 1			
107	AdventureWor	SalesLT.SalesOrderHeader	ModifiedDate	DateTime		23 Type 1			
108	AdventureWor	SalesLT.SalesOrderHeader	CustomerLastName	String	50	24 Type 1			

Getting Started | Connections | Batches | Projects | Objects | Columns | Parameters | Attributes | Configurations | DataTypeMappings | Versions

Column ModelOverrideName

This is used to define the business attribute name.

While the same column names are used in source to staging to persistent staging, it is usually advisable to use actual business process aligned names in the Data Vault.

If a source system has column names like `GL002CD123` and the actual business name is `GeneralLedgerCode` the Accelerator will convert this to `[GeneralLedgerCode]`

ModelGrouping

This attribute is used to group columns and split them into separate Satellites and to group columns defining a Unit of Work for Links.

ModelReference

This attribute is used to define the Link and Link Satellite concatenated table name. It is required when multiple table references exist on the source table.

An example would be the `SalesOrderLine` source table where the `ShippingAddress` and `BillingAddress` both reference the `Address` table.

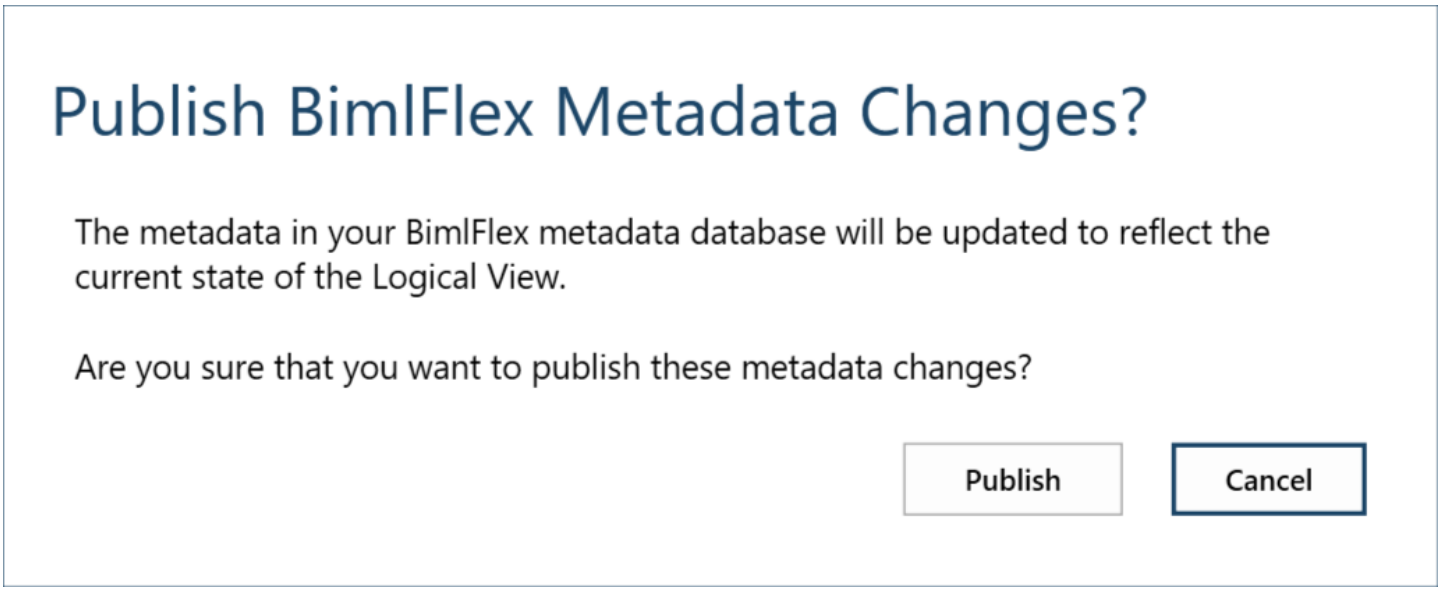
Two relationships will be created with the ModelReference forming part of the name. This attribute is autogenerated when importing metadata and can be overridden.

Final review and preview publish

Once the metadata has been modelled and the preview represents the target model a user will then publish the metadata to the metadata repository.

Publishing the final preview

By clicking the Publish DV Import the new data will be committed to the Metadata database and made available in BimlFlex Excel.



Once the Metadata is published it is possible to update the metadata, getting all entities, in BimlFlex Excel to review the new objects and columns now available for further modelling.

Once reviewed in BimlFlex Excel Add-in it is possible to generate all Tables and database structures as well as SSIS packages based on the Accelerated metadata from within BimlStudio.

Data Vault Templates

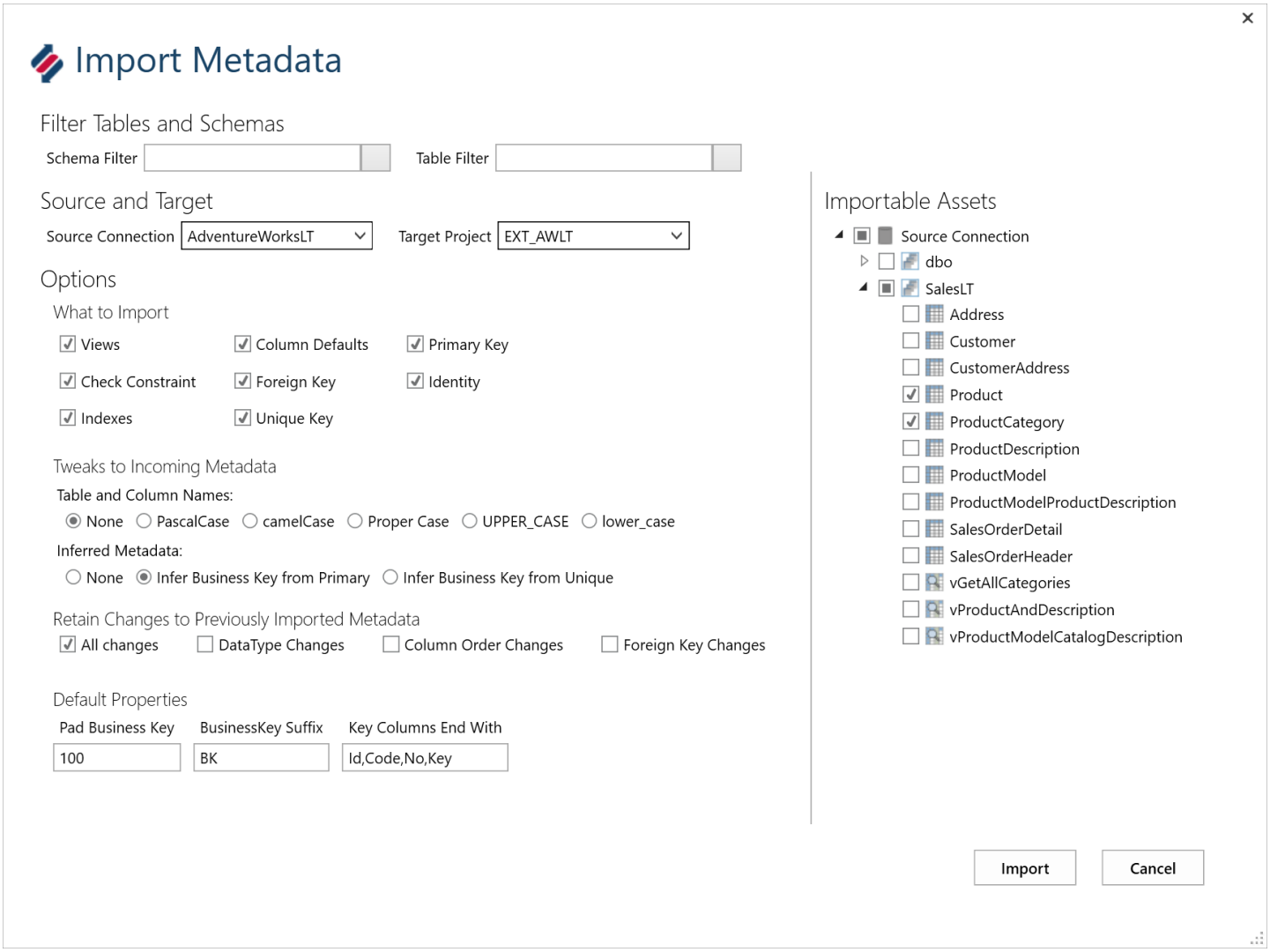
This document outlines the creation and management of Data Vault artefacts in BimlFlex.

The Data Vault can be accelerated using the BimlFlex Data Vault Accelerator that is described in its own document. This document focuses on the process of manually creating Data Vault artefacts by manipulating metadata.

Setup

This document assumes the BimlFlex product has been installed and configured and that the AdventureWorksLT sample metadata is ready to be imported to a new empty customer.

The walkthrough will use the Product and Product Category entities from the AdventureWorksLT source to demonstrate Data Vault concepts.



Data Vault review

It is important to consider the Data Vault design and modelling principles while building the Enterprise Data warehouse. There are several books and training courses available on the subject of Data Vault modelling. Varigence provides a Data Vault modelling and implementation course that combines theoretical knowledge about the Data Vault approach with implementation guides using BimlFlex.

The Data Vault Pillars

It can be prudent to revisit the Data Vault modelling pillars as defined by the CDVDM, The Data Vault modelling certification definition.

- Data Vault Models are built entirely upon Hubs, Links and Satellites
- Data Vault Hubs and Links contain no descriptive attributes
- Satellites contain all descriptive attributes (all context, details, time slices, etc.)
- Data Vault Hubs contain no Foreign Keys/relationships
- Satellites contain one-and-only-one Foreign Key/relationship (for the Hub or Link to which it is attached)
- All relationships represented in a Data Vault are manifest through Link Tables
- Satellites can attach directly to one-and-only-one Hub or Link
- Data Vault Hubs and Links all use a Sequence ID or Hash for a primary key

Base Data Vault constructs

BimlFlex generates the required artefacts to populate Hubs, Links, Satellites etc from its metadata.

Hubs

Hubs maintain a distinct list of business keys. The business key and the Hub table, as well as the source and source to target mapping are defined in the metadata repository.

The Hub Entity is at the centre of the Core Business Concept, CBC, and should be derived from Enterprise Wide Business Keys, EWBK's. A Hub is not necessarily the same as a Primary Key in the source system.

To be able to integrate across systems and versions the actual Business terms used should be identified and used as keys for the Hubs.

For the business key, it is recommended to use a wide Unicode/nvarchar datatype. That accommodates most data coming from any source. It also allows new sources to be integrated that might not adhere to the earlier assumed datatype of the Hub business key. The hub should accommodate all incoming data without judging, therefore it is recommended to use the most forgiving data type available.

Descriptive attributes about the business key in the Hub are stored in attached Satellites.

Links

Links maintain relationships between two or more Hubs. The Link is also a distinct set of all occurrences of the combination of hubs ever seen. These relationships have their effectiveness maintained through (Link) Satellites.

Some Links need to emulate the Foreign Key constraints in their source, this is managed by defining Driving Keys for the Link Satellite.

The Link forms the base for the Unit of Work, UOW. The UOW defines the required granularity to properly identify the relationship; the UOW forms the grain of the relationship.

Satellites

A Satellite is connected to either a Hub or a Link and maintain descriptive information about the Hub or Link. Satellites are the bearer of all descriptive attributes, context information and time-slice history in the Data Vault model.

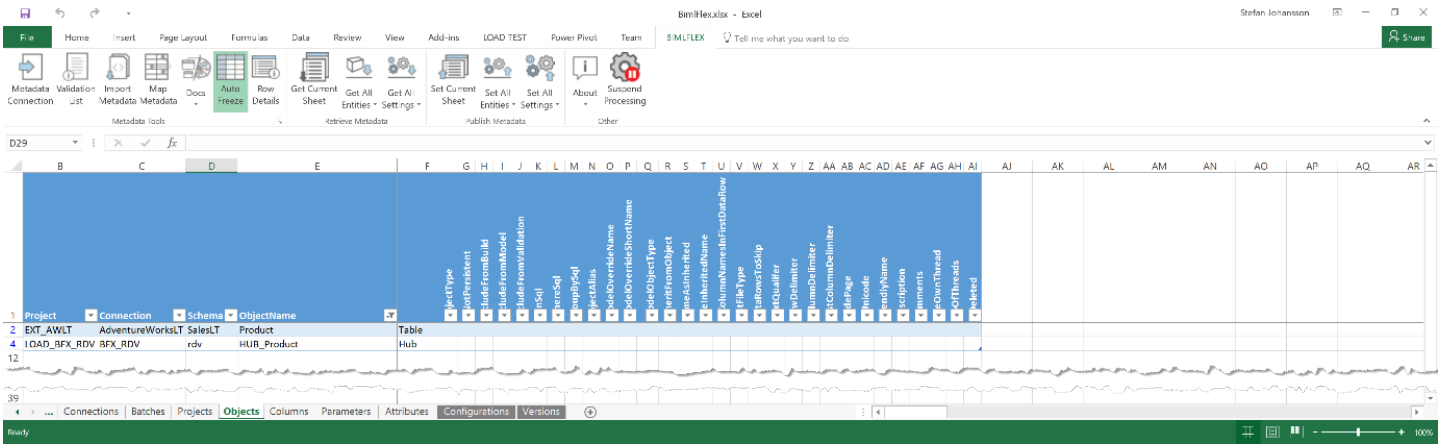
An example is the effectiveness of a relationship, when a link is created from a relationship between two entities in the source it will create two Hub rows and a Link row (if some of the Hub records already exist those existing rows will be reused). The Link Satellite tracking effectiveness will have a row added that describes when the event was first discovered from the source. Should the source remove the relationship, the Link Satellite will be closed off by adding a new row with a deleted indicator describing the delete event.

Another example is the management of descriptive attributes. For a Product stored in the Product Hub there will be Satellites storing the information about the Product. Should the List Price be changed in the source the Satellite will have another row added with this new data and the old record for the old price will be end dated as no longer current.

More on Hubs

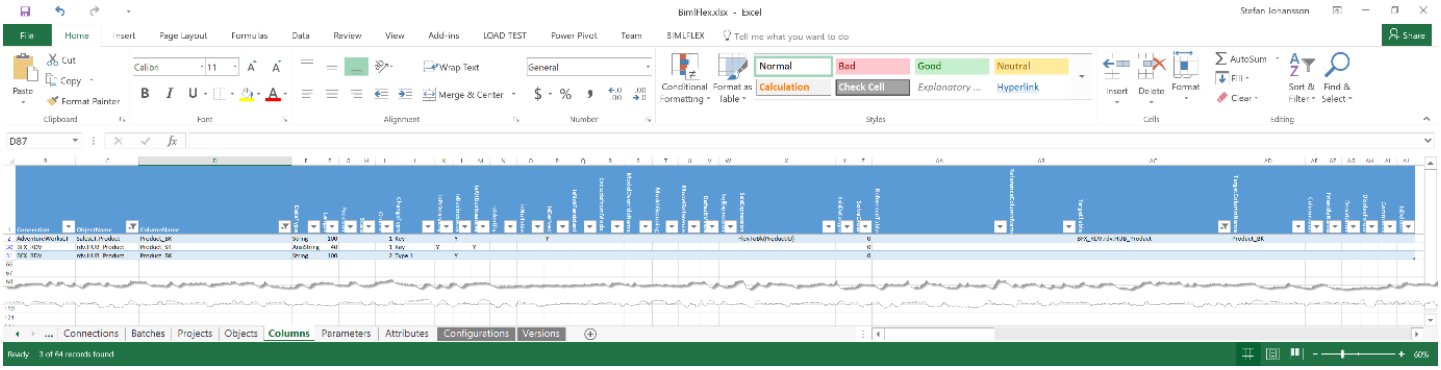
The Hub is the distinct set of business keys from the source. It is an add only table. Any effectiveness or descriptive attributes are tracked in attached Satellites.

The required metadata for a Hub is divided into the objects tab and the columns tab in the metadata editor. For the Source to Target Mapping the source and the target table and columns needs to be defined and mapped.



The source table can be any metadata construct. The normal use case would be to read from the staging tables of an external source. Once new data had been staged the Raw Data Vault process will load the data into the destination Hub table.

The object tab only requires the objects to be defined. The destination table is marked as Hub to indicate the entity type. The naming convention needs to be applied to the Object Name as it is manually added.



Only the business key needs to be mapped to the target Hub. The source table provides the source key columns and the destination Hub has a Business key and a Surrogate Key defined.

Surrogate Key

For the Hub Surrogate Key, the data type will be `AnsiString (40)`, a text representation of the SHA1 Hash bytes. BimlFlex currently support SHA1 hashing. The SK data type must match the output of the hashing algorithm.

Business Key

The Hub business key needs to accommodate the data from the existing sources as well as being able to accommodate changes in the existing sources and new sources added later. The CBC/EWBK represented by the Hub is designed to integrate multiple systems. To support this, it is most common to use text representation, either using text or Unicode. The key length needs to accommodate any reasonable business key that can be foreseen. The only reason to use a smaller key representation is performance. If Hub business key length is a performance concern it is recommended to invest in better performing systems rather than minimise key length.

The default key length generated by BimlFlex using the Data Vault Accelerator is String/NVARCHAR(100). For the demo we will create a business key from the integer representation of the ProductID in the source as a String/NVARCHAR(100).

The Business key is added to the source table as a new column. It uses the `FlexToBk(ProductID)` custom SSIS Expression to build the

business key. The default settings will concatenate the business keys together with the tilde `~` as the default concatenation character. The `Product_BK` only has a single column mapped but if we added an additional one it would create a composite key with the columns concatenated.

`FlexToBk(ProductID)` will generate `Product_BK` from the ID only (example: `680`)

`FlexToBk(Name,ProductNumber)` will generate `Product_BK` from the Name concatenated with the ProductNumber (example: `HL Road Frame - Black, 58~FR-R92B-58`) with the concatenation separator injected between (the `~`)

BimlFlex Build

Once the metadata is pushed from the Excel metadata editor to the repository it is possible to refresh the metadata into BimlFlex.

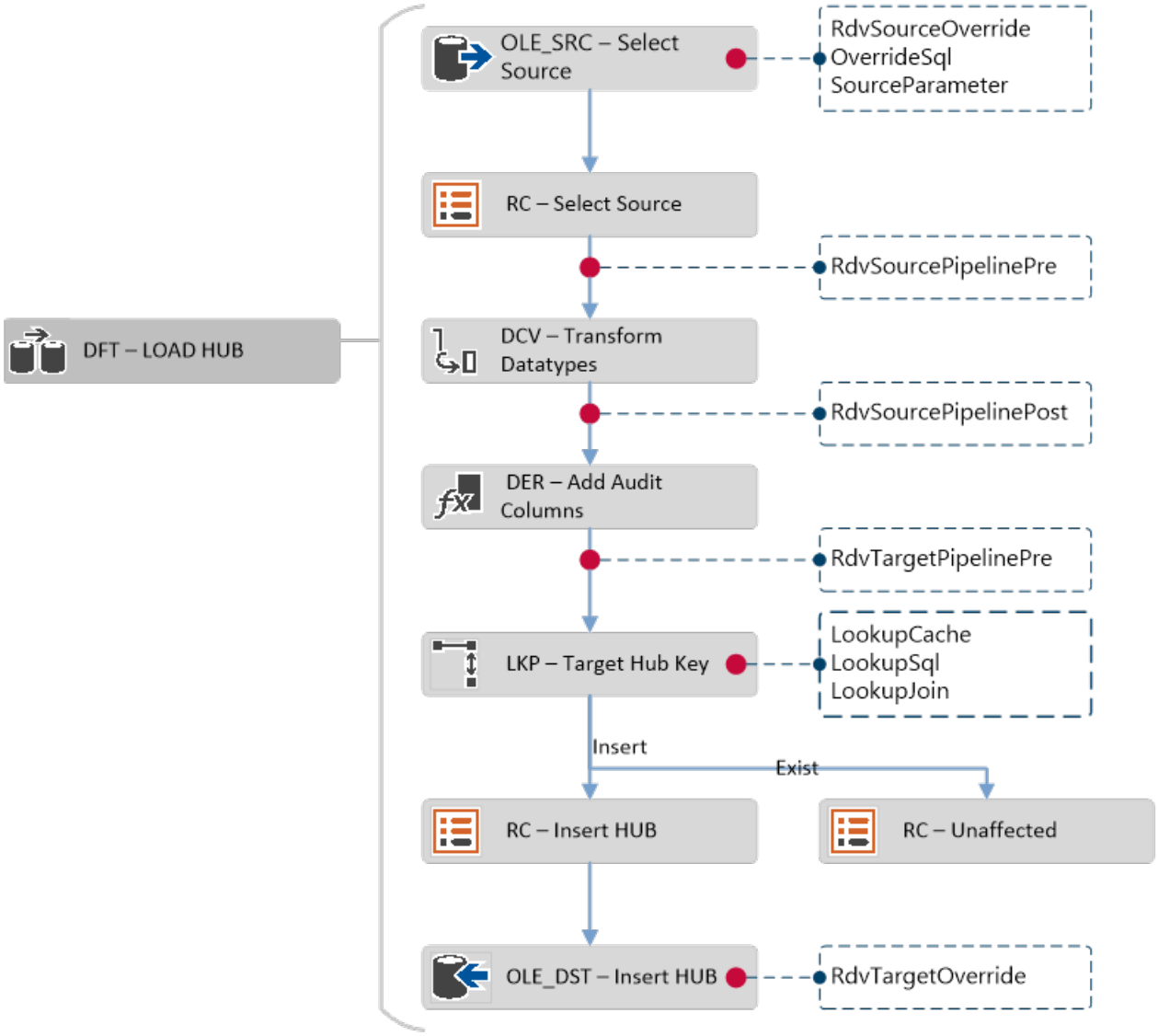
BimlFlex will indicate the tables and the SSIS artefacts that will be created. Reviewing the artefact list in the solution explorer will indicate if the solution will match expectations.

Once the solution is built through BimlStudio the normal output artefacts from the staging, persistent staging, table and database creation will include a new project with the Raw Data Vault load batch and Hub ETL process.

BimlFlex groups Raw Data Vault loads by source table. The generated Product Hub will be included in a package called `LOAD_BFX_RDV_Product_AWLT.dtsx` in the SSIS project called `LOAD_BFX_RDV`.

The Hub load process will consist of a single Data Flow going from the source `Product` staging table to the destination `HUB_Product` table

Illustrated here is the Hub load flow:



For all Hubs in the Data Vault model, the source and destination table metadata is required. The column mappings include a business key column mapped to the destination hub and a defined destination Hub with a business key column and surrogate key column. The metadata directive `FlexToBK(key column)` is used to build Business keys.

Multiple key columns

Some Hub designs require multiple source key columns to define the Hub through the Business Key. Sources with key overlap might need a system or source string added, multiple source keys might need to be combined to form a distinct Hub business key.

> Deriving the business keys for the CBC/EWBK's is one of the most important design exercises in modelling the Data Vault. This guide does not include details on the required analysis and design process.

For these columns BimlFlex concatenates them into a single string and separates them with the configured separator `~` as described above.

Link

The Link Entity is the distinct set of relationships between the involved Hubs. As the Hub, it is an add only table and any effectiveness or attributes should be tracked in a connected Link Satellite.

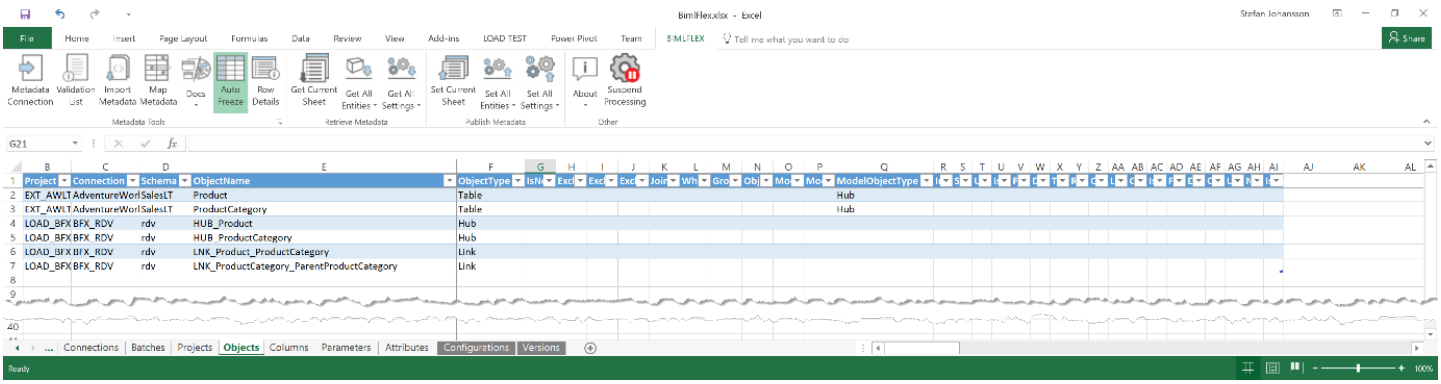
Two or more Hubs are required to build a Link. The Product source table used for the Hub has a relationship to the Product Category table, representing the category of a product. The Product Category will require a Hub for the Link to be built. The Product Category table also has a hierarchy with a parent category. This provides an interesting scenario as that Link will reference the same Hub twice and therefore will need roleplaying names.

Links are built from the metadata for a single table so if there is a requirement to add data or attributes from different tables they have to be joined into the source table in the source metadata.

It is recommended that sources provide all the required metadata for relationships between entities to be built using the identified business keys.

For this guide the technical Id's from the source are used.

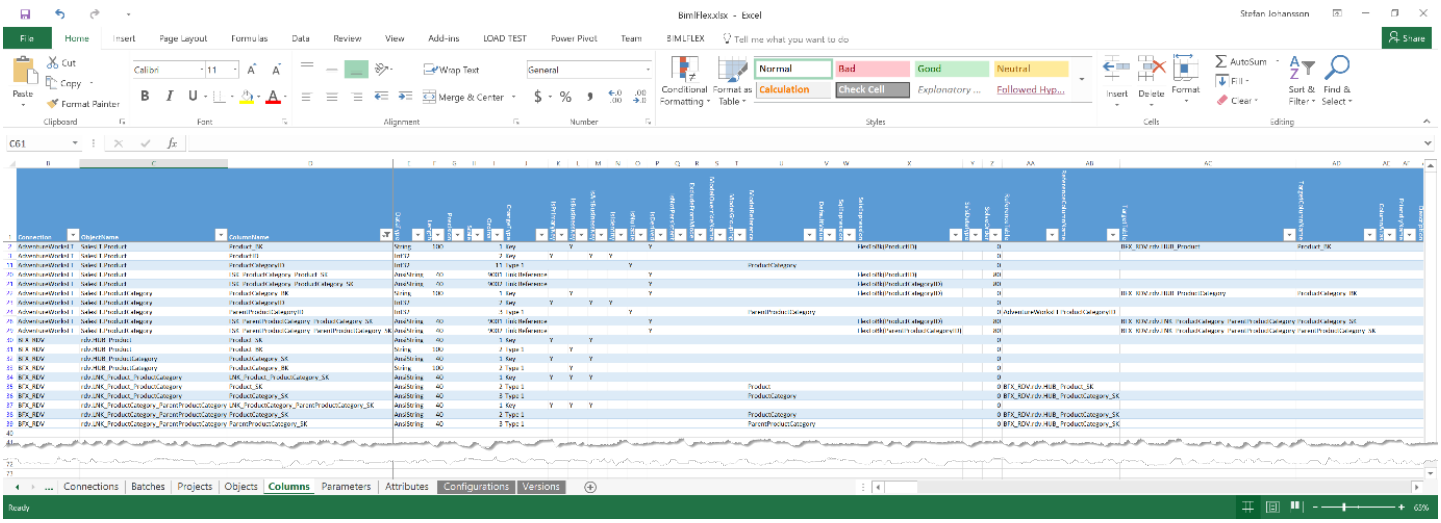
For the metadata objects tab the new Hub and the Link needs to be added



The source now includes the `ProductCategory` table and the `rdv` objects include both the Hubs and the two Links.

The default naming convention used by the Accelerator is to create the Link name as the relationship between the Hubs. This works in the example scenario but for Links will more Hubs attached the name most likely needs to be adjusted to use a more concise convention. The recommendation is to use the convention consistently and have as explicitly meaningful names as possible.

The columns metadata needs to include the mapping for the new hub as well as the Links.



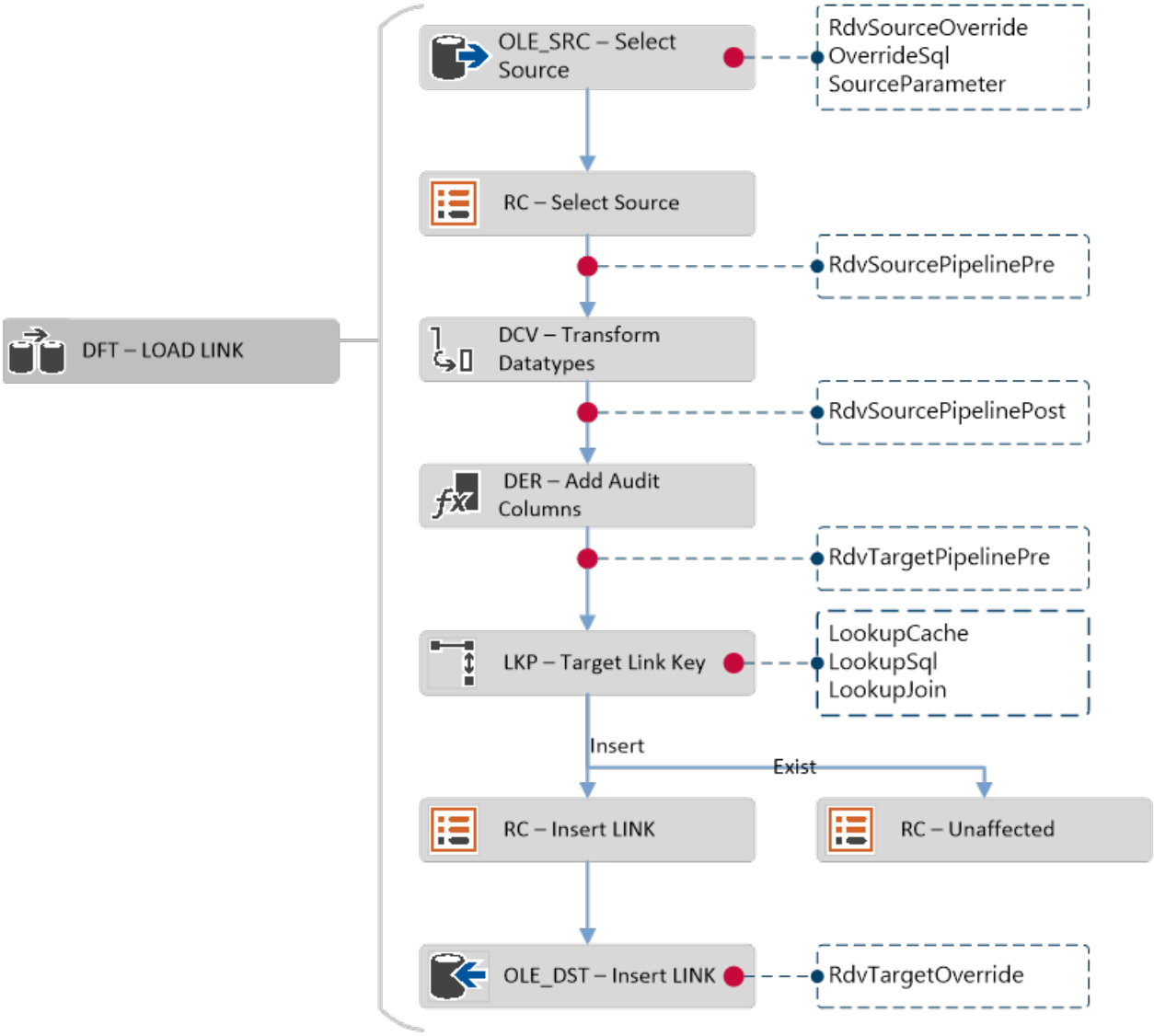
The Source tables need the Link keys for reference. The default naming convention uses `LSK_Entity_Entity_SK` naming style. The source tables have the keys for the Links added as well as the Hub keys for the relationships. The LSK column has the Link table as target table specified.

As before the Hub have a BK and an SK column.

The Link Data Vault entries have a Link SK as well as an entry for each participating Hub. The entries for the participating Hubs have an entry in the `ReferenceTable` and `ReferenceColumnName` to identify the Hubs.

Building the solution from this metadata will generate two load packages in the Data Vault project, one each for the source tables. In each package will be loads for the Hub and the Link.

Illustrated here is the Link load flow:



More on Satellites

Satellites can be attached to either a Hub or a Link, to differentiate the Satellites attached to Links are called Link Satellites and prefixed with LSAT compared to Hub Satellites that uses SAT.

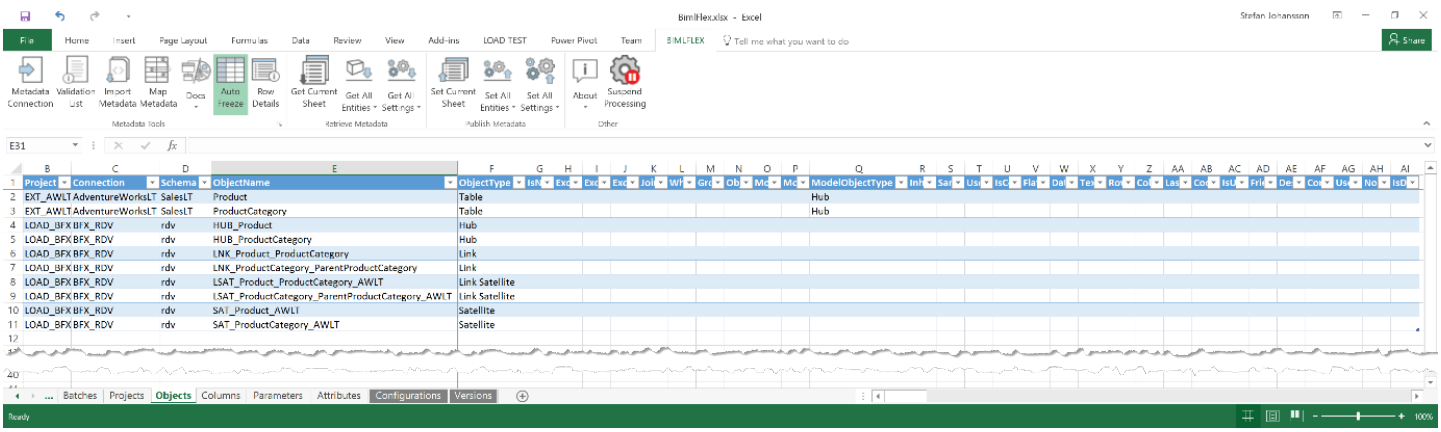
The Satellite concept is the same for both entity types. The columns and load patterns are the same. They both track changes over time. The common use for the Hub Satellite is to track descriptive attributes and their changes over time and the effectiveness of the Hub. When a Product gets deleted from the source system it will not be removed from the Hub, it will have a row in the Satellite indicating that the status of the Product is now deleted.

Link Satellites most common use case is to track effectiveness of the relationship. The Link Satellite can also have attributes. The Link Satellite attributes can sometime be attached to a Hub with a separate Satellite instead. This document does not include the modelling considerations of this, BimlFlex supports Link Satellite attributes.

The source data for the sample model contains descriptive attributes for both Product and Product Category. There are no descriptive attributes for the Link relationships so the Link Satellites will only track effectiveness of the relationships.

Hub Satellites

The required Source Object metadata is already in the metadata model, the Satellites attributes are read from the same source as the Hub. The destination Satellite tables/entities are added by adding the names and setting the Object Type to Satellite or Link Satellite

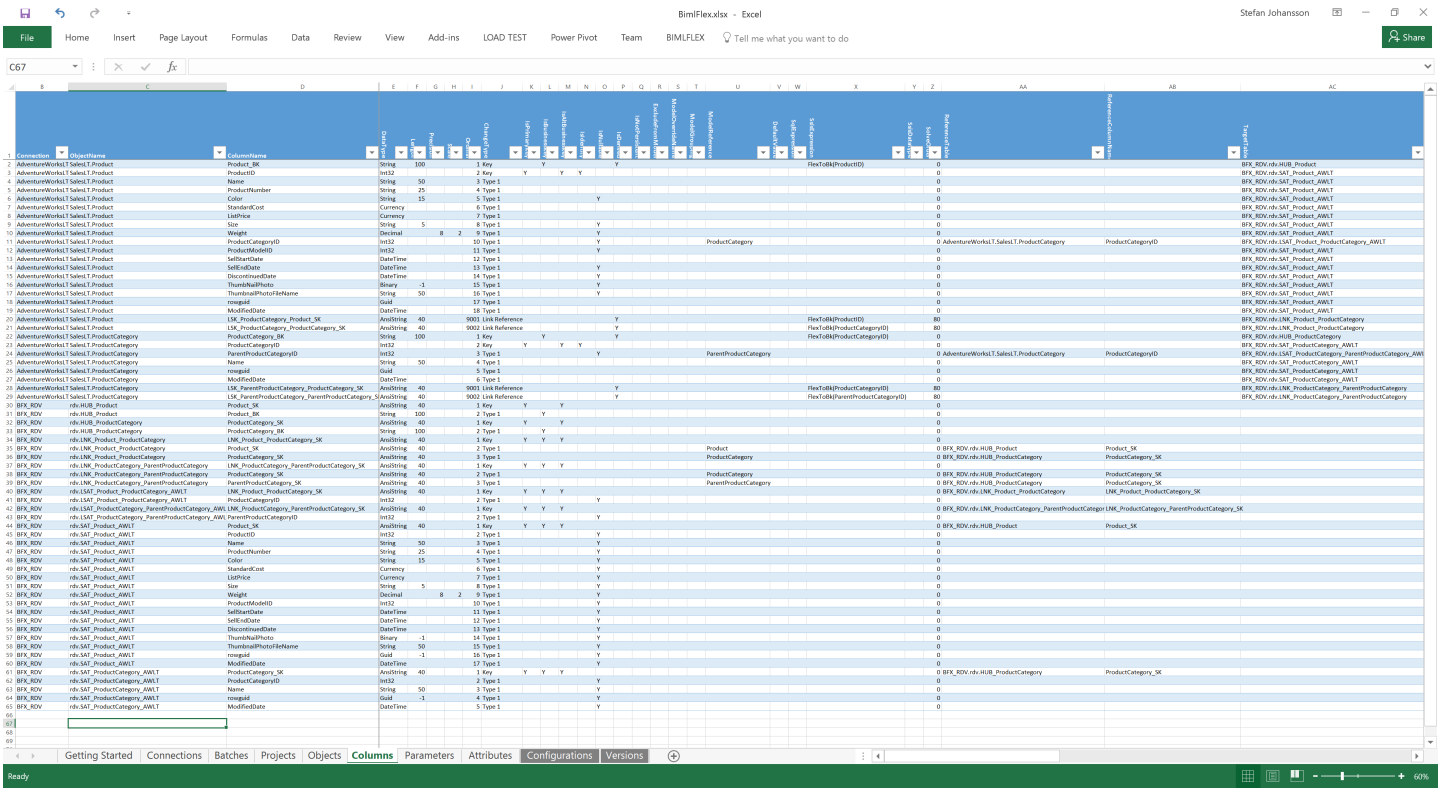


Project	Connection	Schema	ObjectName	ObjectType	ModelObjectType
EXT_AWLT	AdventureWorksLT	SalesLT	Product	Table	Hub
EXT_AWLT	AdventureWorksLT	SalesLT	ProductCategory	Table	Hub
LOAD_BFX	BFX_RDV	rdv	HUB_Product	Hub	
LOAD_BFX	BFX_RDV	rdv	HUB_ProductCategory	Hub	
LOAD_BFX	BFX_RDV	rdv	LNK_Product_ProductCategory	Link	
LOAD_BFX	BFX_RDV	rdv	LNK_ProductCategory_ParentProductCategory	Link	
LOAD_BFX	BFX_RDV	rdv	LSAT_Product_ProductCategory_AWLT	Link Satellite	
LOAD_BFX	BFX_RDV	rdv	LSAT_ProductCategory_ParentProductCategory_AWLT	Link Satellite	
LOAD_BFX	BFX_RDV	rdv	SAT_Product_AWLT	Satellite	
LOAD_BFX	BFX_RDV	rdv	SAT_ProductCategory_AWLT	Satellite	

The default naming convention uses the same name for the Satellite as the Hub or Link it is attached to with an additional suffix indicating the source system. It is recommended to use a consistent naming convention for all artefacts. The suffix naming suggests that additional Satellites can be added to the Hubs from other sources.

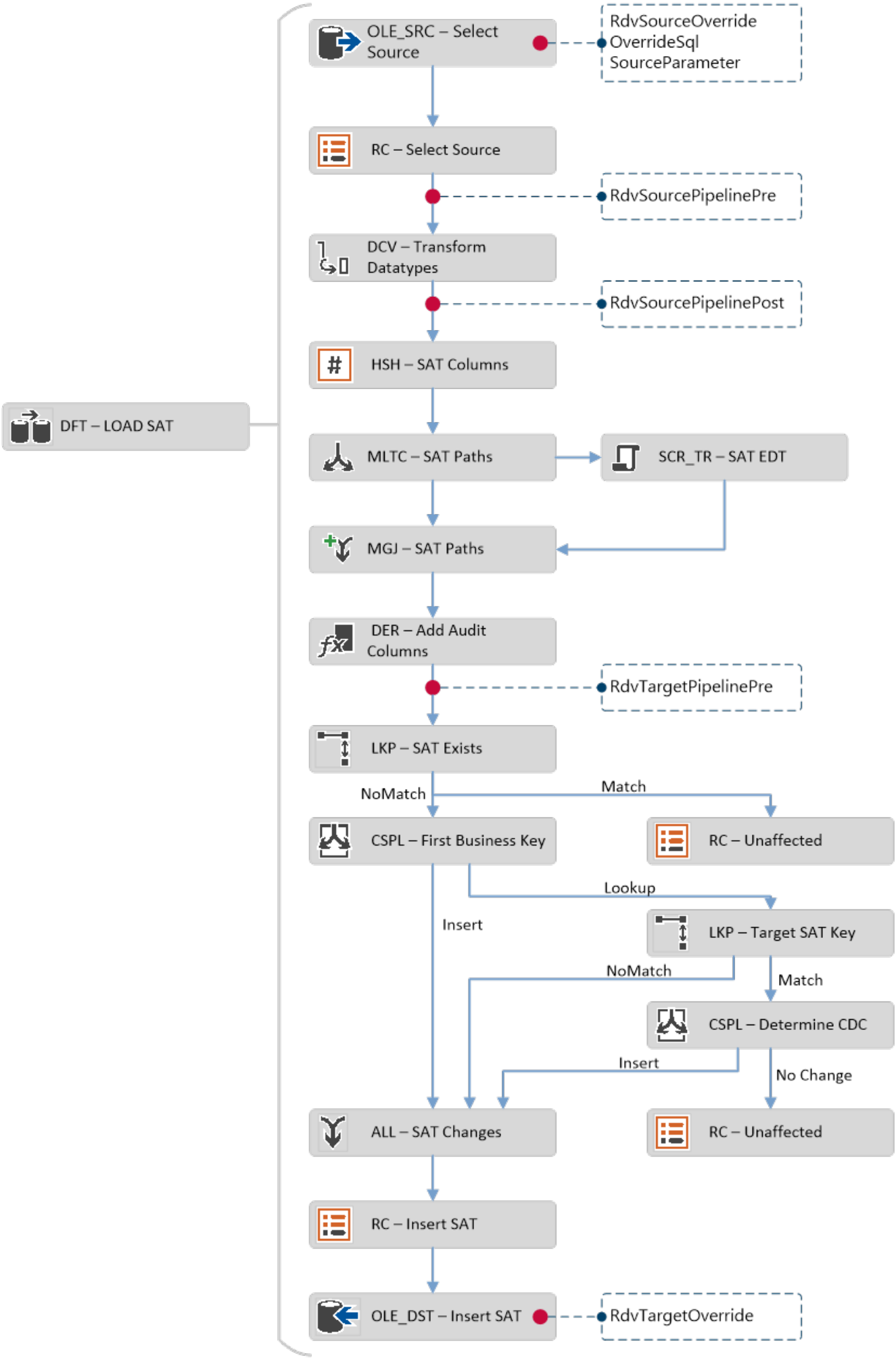
Only under very, very specific circumstances is it recommended to load data into the same Satellite from multiple different sources and even then it is always possible to still load to separate destinations. The Data Vault model supports an unlimited number of Satellites attached to Hubs and Links, this should be embraced in the modelling so that the agility and flexibility to integrate changes and new sources are maximised.

The Columns metadata requires additional data for all attributes for the Satellites. Once multiple source tables and attributes are added to the metadata editor it is helpful to filter the columns to only display the required data. The Excel headers are normal table headers and support both filtering and sorting. Note that any filtering and sorting choices are removed when the metadata is refreshed.



Connection	ObjectName	ColumnName
AdventureWorksLT	SalesLT.Product	Product_BK
AdventureWorksLT	SalesLT.Product	ProductID
AdventureWorksLT	SalesLT.Product	Name
AdventureWorksLT	SalesLT.Product	ProductNumber
AdventureWorksLT	SalesLT.Product	Color
AdventureWorksLT	SalesLT.Product	StandardCost
AdventureWorksLT	SalesLT.Product	ListPrice
AdventureWorksLT	SalesLT.Product	Size
AdventureWorksLT	SalesLT.Product	Weight
AdventureWorksLT	SalesLT.Product	ProductCategoryID
AdventureWorksLT	SalesLT.Product	ProductModelID
AdventureWorksLT	SalesLT.Product	SellStartDate
AdventureWorksLT	SalesLT.Product	SellEndDate
AdventureWorksLT	SalesLT.Product	DiscontinuedDate
AdventureWorksLT	SalesLT.Product	ThumbNailPhoto
AdventureWorksLT	SalesLT.Product	ThumbnailPhotoFileName
AdventureWorksLT	SalesLT.Product	rowguid
AdventureWorksLT	SalesLT.Product	ModifiedDate
AdventureWorksLT	SalesLT.Product	LSK_ProductCategory_Product_SK
AdventureWorksLT	SalesLT.Product	LSK_ProductCategory_ProductCategory_SK
AdventureWorksLT	SalesLT.ProductCategory	ProductCategory_BK
AdventureWorksLT	SalesLT.ProductCategory	ProductCategoryID
AdventureWorksLT	SalesLT.ProductCategory	ParentProductCategoryID
AdventureWorksLT	SalesLT.ProductCategory	Name
AdventureWorksLT	SalesLT.ProductCategory	rowguid
AdventureWorksLT	SalesLT.ProductCategory	ModifiedDate
AdventureWorksLT	SalesLT.ProductCategory	LSK_ParentProductCategory_ProductCategory_SK
AdventureWorksLT	SalesLT.ProductCategory	LSK_ParentProductCategory_ParentProductCategory_SK
BFX_RDV	rdv.HUB_Product	Product_SK
BFX_RDV	rdv.HUB_Product	Product_BK
BFX_RDV	rdv.HUB_ProductCategory	ProductCategory_SK
BFX_RDV	rdv.HUB_ProductCategory	ProductCategory_BK
BFX_RDV	rdv.LNK_Product_ProductCategory	LNK_Product_ProductCategory_SK
BFX_RDV	rdv.LNK_Product_ProductCategory	Product_SK
BFX_RDV	rdv.LNK_Product_ProductCategory	ProductCategory_SK
BFX_RDV	rdv.LNK_ProductCategory_ParentProductCategory	LNK_ProductCategory_ParentProductCategory_SK
BFX_RDV	rdv.LNK_ProductCategory_ParentProductCategory	ProductCategory_SK
BFX_RDV	rdv.LNK_ProductCategory_ParentProductCategory	ParentProductCategory_SK
BFX_RDV	rdv.LSAT_Product_ProductCategory_AWLT	LNK_Product_ProductCategory_SK
BFX_RDV	rdv.LSAT_Product_ProductCategory_AWLT	ProductCategoryID
BFX_RDV	rdv.LSAT_ProductCategory_ParentProductCategory_AWLT	LNK_ProductCategory_ParentProductCategory_SK
BFX_RDV	rdv.LSAT_ProductCategory_ParentProductCategory_AWLT	ParentProductCategoryID
BFX_RDV	rdv.SAT_Product_AWLT	Product_SK
BFX_RDV	rdv.SAT_Product_AWLT	ProductID
BFX_RDV	rdv.SAT_Product_AWLT	Name
BFX_RDV	rdv.SAT_Product_AWLT	ProductNumber
BFX_RDV	rdv.SAT_Product_AWLT	Color
BFX_RDV	rdv.SAT_Product_AWLT	StandardCost
BFX_RDV	rdv.SAT_Product_AWLT	ListPrice
BFX_RDV	rdv.SAT_Product_AWLT	Size
BFX_RDV	rdv.SAT_Product_AWLT	Weight
BFX_RDV	rdv.SAT_Product_AWLT	ProductModelID
BFX_RDV	rdv.SAT_Product_AWLT	SellStartDate
BFX_RDV	rdv.SAT_Product_AWLT	SellEndDate
BFX_RDV	rdv.SAT_Product_AWLT	DiscontinuedDate
BFX_RDV	rdv.SAT_Product_AWLT	ThumbNailPhoto
BFX_RDV	rdv.SAT_Product_AWLT	ThumbnailPhotoFileName
BFX_RDV	rdv.SAT_Product_AWLT	rowguid
BFX_RDV	rdv.SAT_Product_AWLT	ModifiedDate
BFX_RDV	rdv.SAT_ProductCategory_AWLT	ProductCategory_SK
BFX_RDV	rdv.SAT_ProductCategory_AWLT	ProductCategoryID
BFX_RDV	rdv.SAT_ProductCategory_AWLT	Name
BFX_RDV	rdv.SAT_ProductCategory_AWLT	rowguid
BFX_RDV	rdv.SAT_ProductCategory_AWLT	ModifiedDate

Illustrated here is the Satellite load flow


DFT – LOAD SAT
OLE_SRC – Select Source
RdvSourceOverride
OverrideSql
SourceParameter
RC – Select Source
RdvSourcePipelinePre
DCV – Transform Datatypes
RdvSourcePipelinePost
HSH – SAT Columns
MLTC – SAT Paths
SCR_TR – SAT EDT
MGJ – SAT Paths
DER – Add Audit Columns
RdvTargetPipelinePre
LKP – SAT Exists
NoMatch
Match
CSPL – First Business Key
RC – Unaffected
Lookup
Insert
LKP – Target SAT Key
NoMatch
Match
CSPL – Determine CDC
Insert
No Change
ALL – SAT Changes
RC – Unaffected
RC – Insert SAT
OLE_DST – Insert SAT
RdvTargetOverride

Driving Keys

All Data Vault Links are many to many relationships. If there is a Foreign Key relationship between two entities in the source, that behaviour might need to be maintained in the Data Vault.

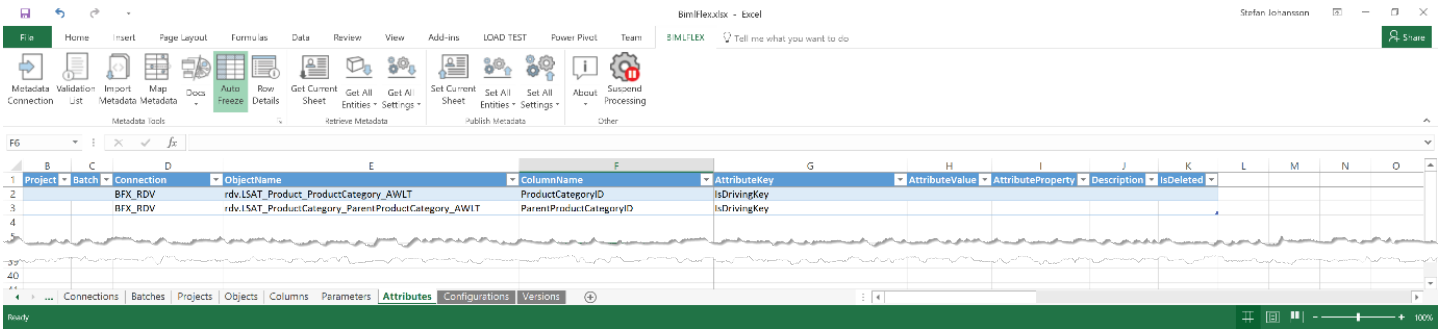
An example is the relationship in AdventureWorksLT between a Product and its Product Category. By definition, the source system only allows product to be part of a single Category at a given point in time. If the product is moved from a category to another it will cease to be part of the previous category.

Since links can maintain any number of relationship this behaviour needs to be enforced by rules. In the Metadata Attributes tab it is possible to define Driving Keys for Link Relationships. These keys define which parts of the Link drive the changing of existing relationships.

For the Product to Product Category Link this is maintained by the `LSAT_Product_ProductCategory_AWLT` Link Satellite table.

The Accelerator and the BimlFlex framework will automatically apply Driving Key type relationships for any Links derived out of a Hub, as they are based on Foreign Keys in the source and by definition imply a Driving Key scenario. This will be automatically included in the SSIS package, no separate attribute will be added in the Attributes Sheet.

If a Driving Key behaviour needs to be manually defined, such as from a Link type source table, an attribute is added to the Attributes Tab.



The Data Vault build logic will include the required processing in the Link Satellite to maintain data consistency throughout load by adding and closing relationships, emulating the behaviour of the single Foreign Key relationship from the source.

Multi Active Satellites

Multi Active Satellites are satellites with an additional key attribute meaning multiple version of the Satellite can be active/valid at a given time. This is used when there are no other straightforward ways to model a situation with multiple active records at the same time.

Multi Active Satellites are created by adding a Multi Active Key to the Satellite definition.

Multi Active Satellites break the formal Data Vault design and modelling pattern and it is recommended to use the default patterns if possible. An extra Hub roleplaying the Multi Active attribute can possibly be added to the UOW/Link to implement a similar behaviour.

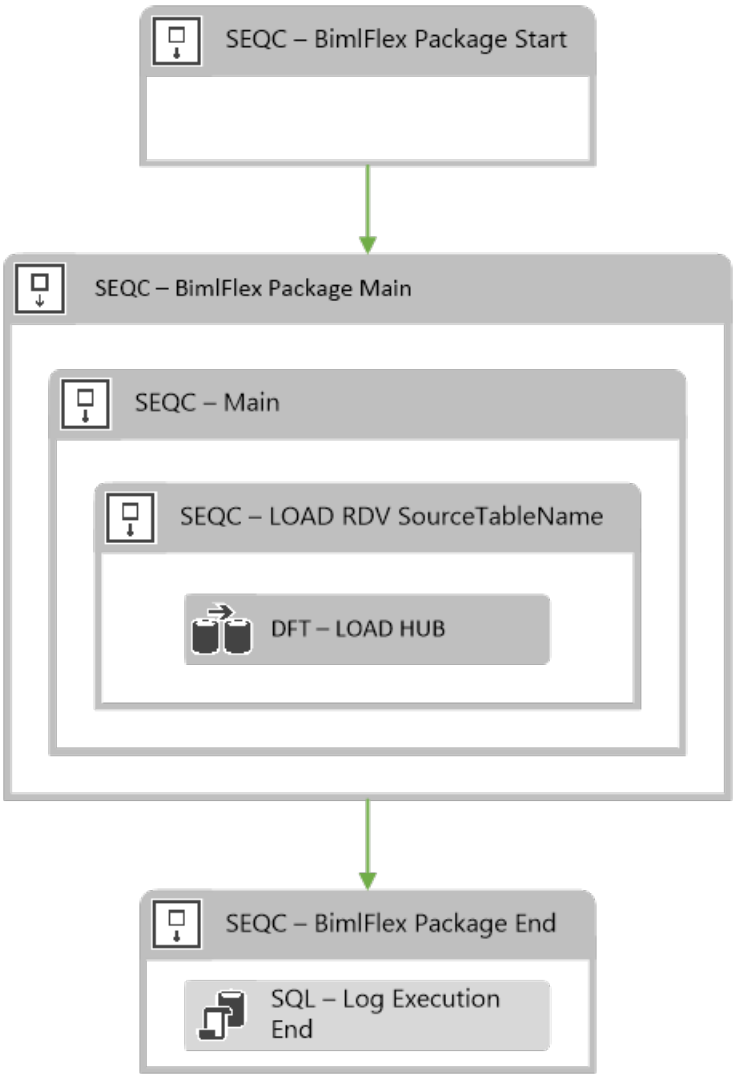
For sources that have multiple changes to the same row at the same time it is easier to define the order and override the `RowEffectiveFromDate` value to reflect that order instead of implementing Multi Activeness.

The BimlFlex generated Data Vault Entity Load SSIS Project

The generated project for loading into the Data Vault will include packages for each source entity. The Entity load package will include all Data Vault loads from the source table. Depending on the defined metadata the generated project will include different parts.

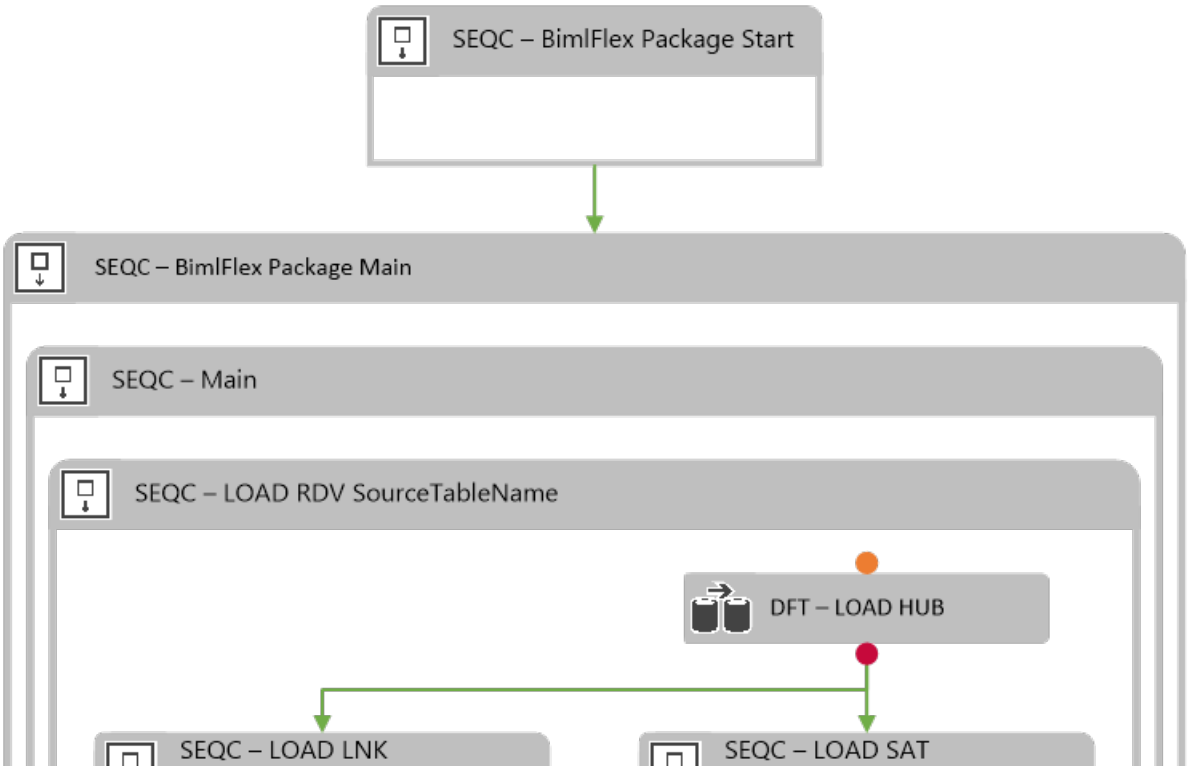
As an example, the following Packages have been generated using increasing amount of metadata.

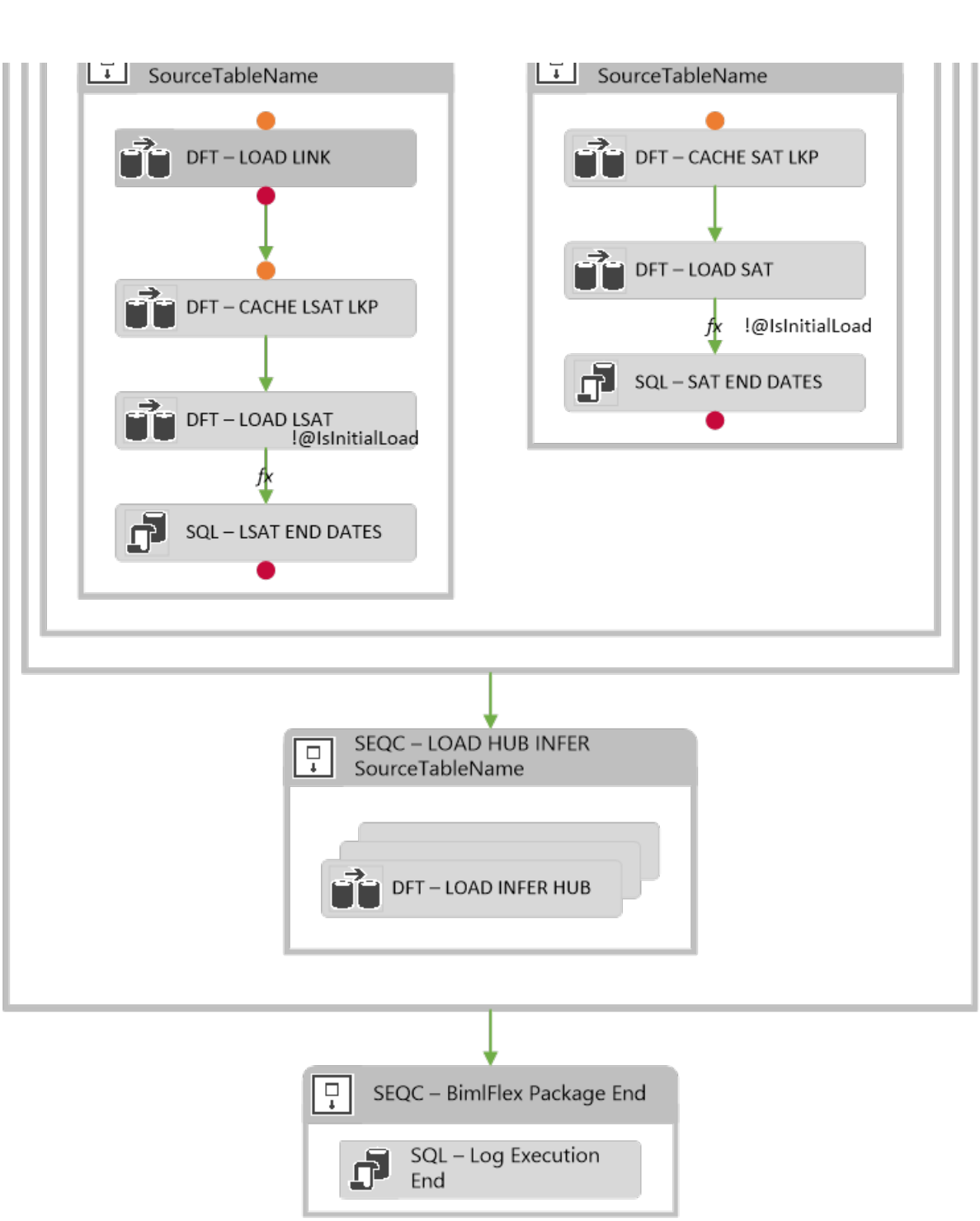
Single Hub load ETL flow sample



This package only includes the Hub entity load of the `HUB_Product` destination.

Full ETL flow sample





The completed package that loads Hub, Link, Satellites and Link Satellites.

Data Mart Templates

This document covers the process to implement a Data Mart layer using BimlFlex.

content include

- Integration to Data Mart and related ETL patterns used in BimlFlex
- Implementation scenarios
- loading data out of Data Vault and into a Data Mart via Point in Time tables and Bridge tables.

Integration to Data Mart

Introduction

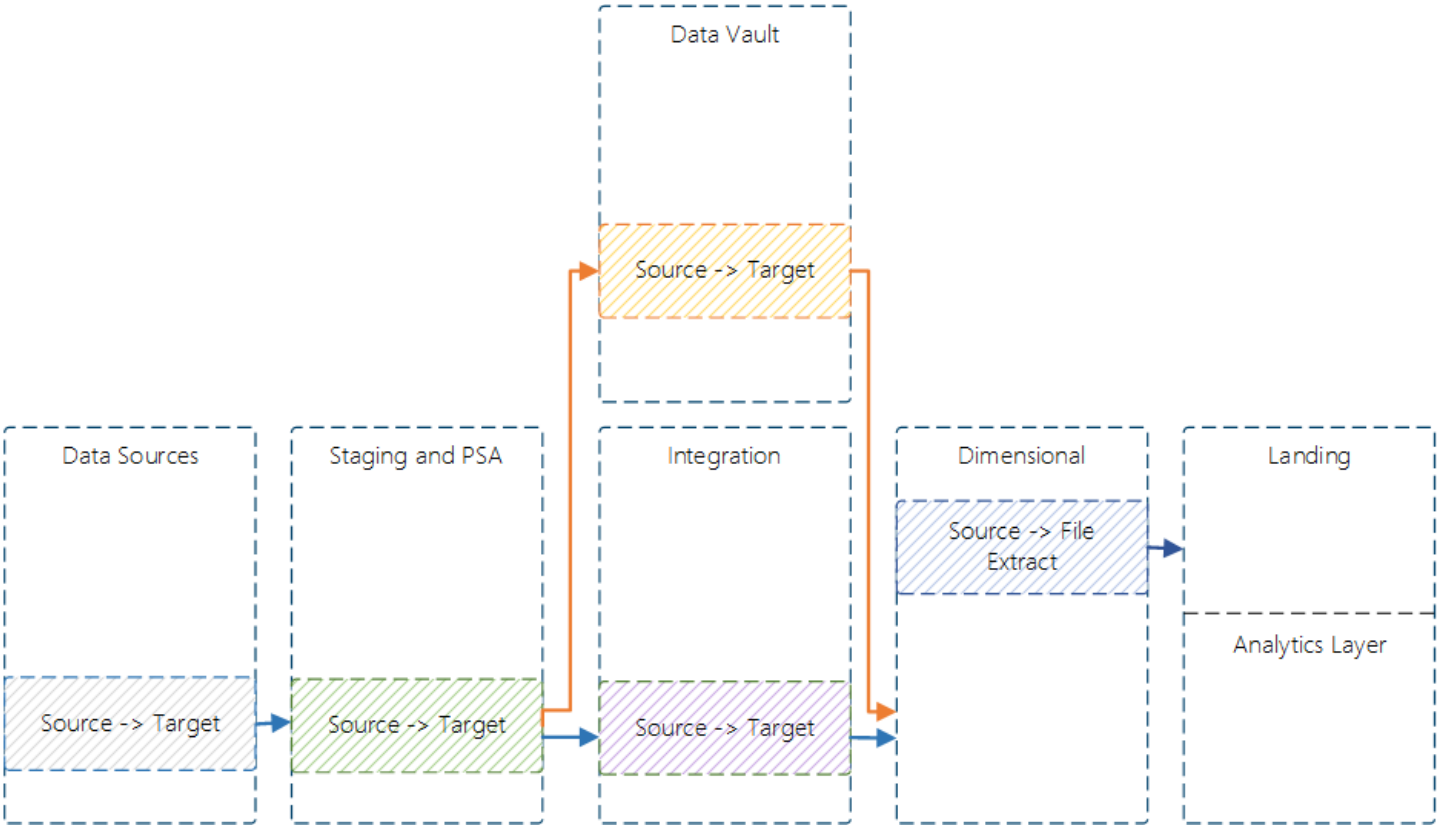
The metadata required to build the Data Mart is similar to any other source to staging project. The ETL pattern that BimlFlex will use for the output will change.

Common Architectures and The Data Mart as a Whole

This implementation guide covers the modelling and implementation of star schema facts and dimensions. It doesn't cover Kimball architecture as a concept. Best practices and basic dimensional implementation constructs, like date and time dimensions are required but considered out of scope for this documentation.

In terms of common architectures, this document begins as the point where a user wants to load the Data Mart and the previous layers of their architecture have already been determined. However, it should be stated that users can implement almost any standardised architectures.

The most common of these two is the following items.

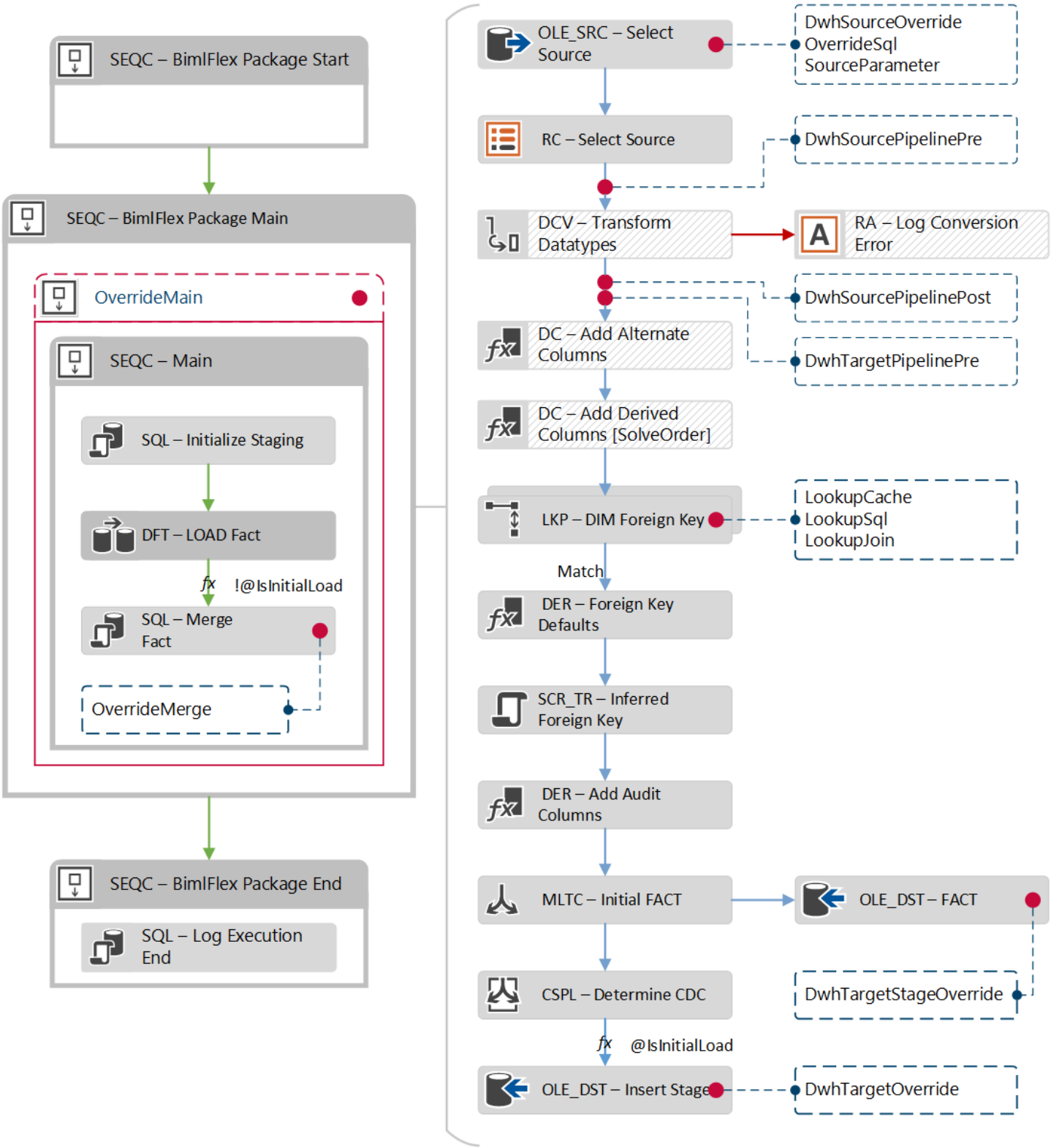


BimlFlex Fact Loading Pattern

Below is a diagram showing the output ETL structure that BimlFlex uses load fact tables in a Data Mart. The optional metadata settings

and extension points are signified by shaded rectangles and dotted rectangles. The standard BimlFlex components are shown in solid colours.

Note that the pattern shows that the orchestration components are applied. More information on the orchestration is documented here



[SQL – Initialize Staging] Inside the main sequence container begins by truncating the target staging table. This staging table will eventually be loaded inside the component `[DFT - LOAD Fact]`.

[DFT – LOAD Fact] is the main data flow and will convert the column data types as per usual in `[DCV - Transform Datatypes]` while logging any errors using `[ERR - Add Error Description]` and `[AUD - Log Error Rows]`.

[LKP – DIM Foreign Key] This component will perform a look up to see if the matching Dim record for this fact record exists in the

corresponding table in the Data Mart.

[DER – Foreign Key Defaults] This derived column component will add a column to store a foreign key populating it with a foreign key if it exists and if not, a default surrogate key will be obtained in the following component `[SCR_TR - Inferred Foreign Key]`.

[DER – Add Audit Columns] This component includes the BimlFlex auditing columns that contain values such as execution IDs.

[MLTC - Initial Fact] This multicast component determines if a given load is, in fact, the first load of a particular fact, in which case the new rows will go directly to the target destination. These rows are directed to `[OLE_DST - FACT]`.

[CSPL – Determine CDC] Here the component checks if an incoming row is Type1 or Type2. If it's determined that a row is of a particular type, then the method of CDC and insertion will be different in the following components

[OLE_DST - Insert Fact] Finally changed and new rows are stored in the fact table.

[SQL – Merge Fact] Will be loading all the data into their final target tables from the data in the initial staging tables.

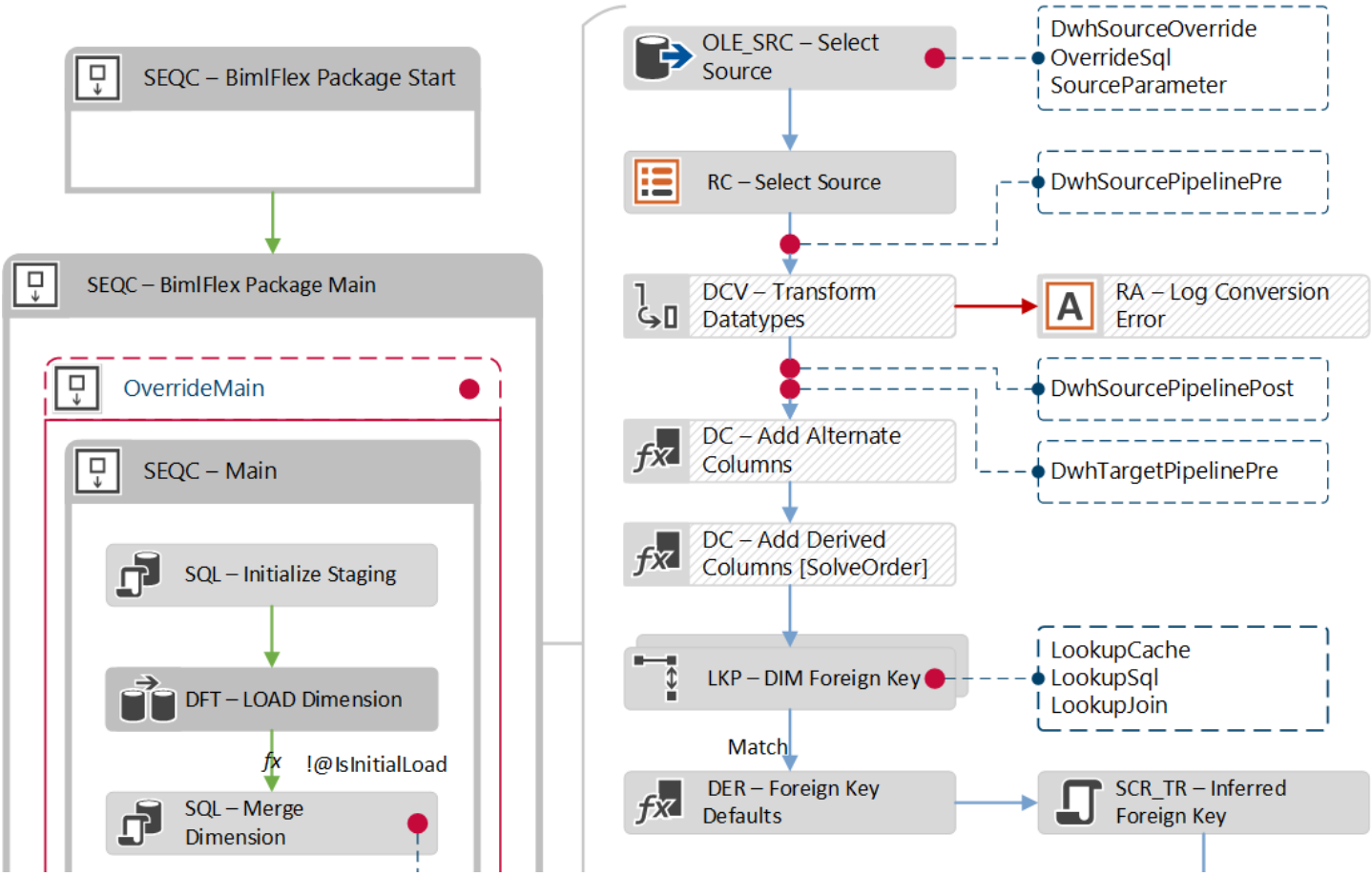
This pattern is not too dissimilar to a source to staging pattern. Key differences are that while the data is being processed by the data flow task, surrogate keys are being handled.

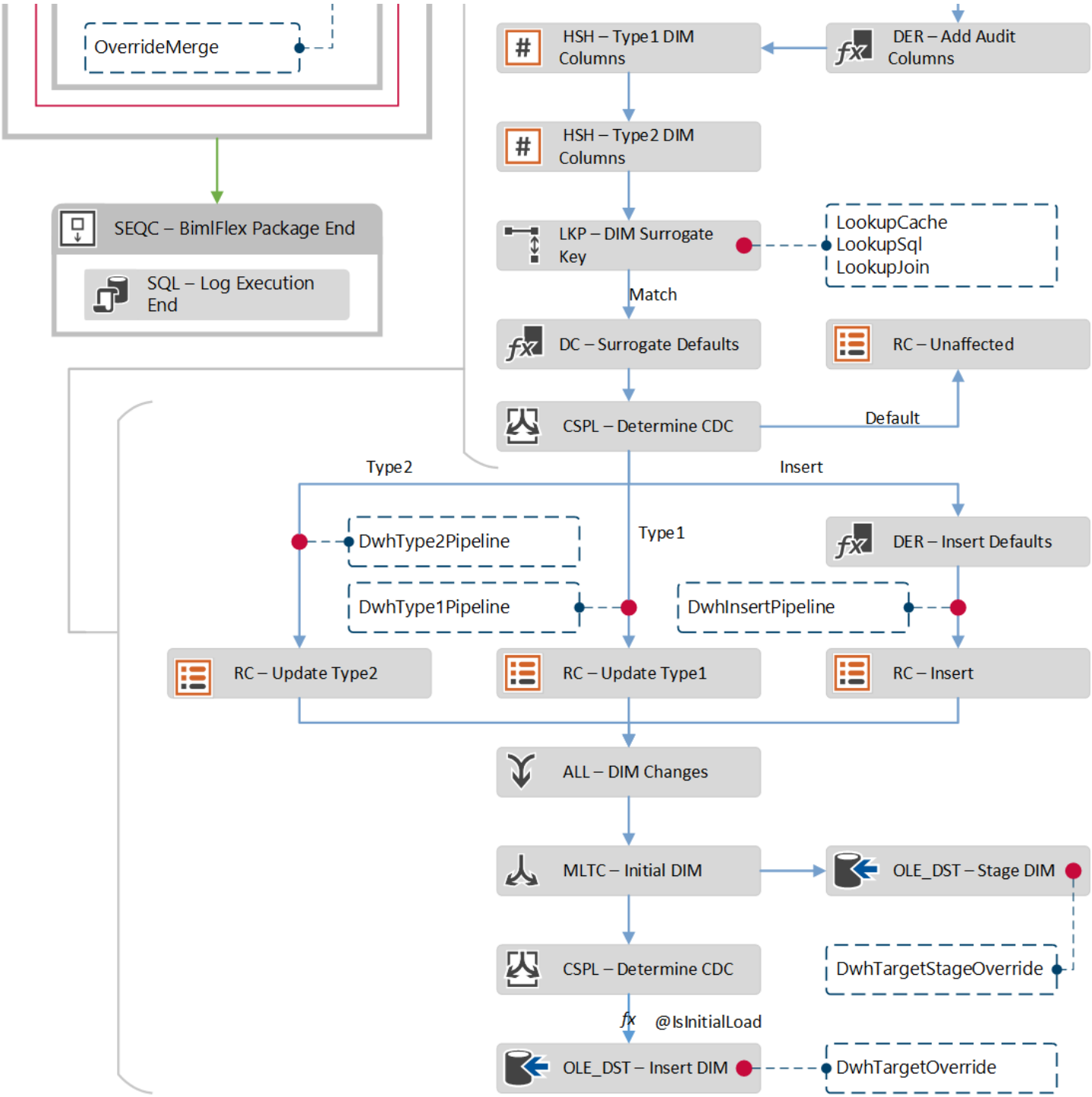
In the control flow, note that the final step is a SQL merge task. The merge take the incoming rows that have been loaded into the Fact's staging table and perform on final comparison before either inserting or updating the target Fact table.

BimlFlex Dimension Loading Pattern

Below is the output of BimlFlex ETL used to load dimension tables in the Data Mart. This is prior to using any optional metadata settings or extension points, which signified by shaded rectangles the dotted rectangles. The standard BimlFlex pattern also applies auditing and logging which is also shown.

The particulars of the dimension loading pattern are that whether the data is type one or type two is taken into account, along with whether it should be directly inserted in a new load. Hashing is also being employed to better detect changes in the column data. More on these details below.





[SQL – Initialize Staging] The main sequence container begins by truncating the target staging table. This staging table will eventually be loaded inside the component [DFT – LOAD Dimension].

[OLE_SRC – Select Source] To obtain the source columns, an OLEDB source component will be used in this example.

[RC – Select Source] For auditing purposes the row count of the previous selection is logged in the BimlCatalog database.

[DCV – Transform Datatypes] This component will be inserted into the ETL structure if a user specifies to convert the data types of the incoming columns before the rows are loaded into their destination table.

[DC – Add Alternate Columns] Add alternate columns component will include any extra derived columns in the data flow. An example of this would be defining both a `SqlExpression` and `SsisExpression` in the metadata combined with providing a `ColumnAlias` will bring the column into the pipeline as the alias name. `[DC – Add Derived Columns <SolveOrder>]` will do the same but taking in consideration the solve order of the columns.

[LKP – DIM Foreign Key] this component performs a lookup on the Dim record's foreign keys in the corresponding tables that the

Data Mart is using

[DER – Foreign Key Defaults] Will add a column to store a foreign key, populating it with a foreign key if it exists and if not a default surrogate key will be obtained in the following component.

[SCR_TR – Inferred Foreign Key] Next, there are two custom components that take the hash value of the dimension row. `[HSH - Type1/2 DIM Columns]` This is useful for improving the performance of an architecture that is required to compare changes in each dimension row. This is especially relevant in Type 2 dimensions.

[LKP – DIM Foreign Key] This component will perform a look up to see if the matching fact record for this dimension exists in the corresponding table in the Data Mart.

[DC – Surrogate Defaults] This derived column component will add a column to store a foreign key populating it with a foreign key if it hasn't received a foreign key from the previous lookup component.

[CSPL – Determine CDC] Here there is a check if an incoming row is Type1 or Type2. If it's determined that a row is of a particular type, then the method of CDC and insertion will be different in the following components

[ALL – DIM Changes] Unions all the new and changed columns of the particular type of CDC for type 1 or type 2.

[MLTC - Initial Dim] This multicast component determines if a given load is, in fact, the first load of a particular Dimension, in which case the new rows will go directly to the target destination. These rows are directed to `[OLE_DST - Insert DIM]`. If it is not an initial load the new and changed rows will be set to the Dimension's staging table first `[OLE_DST - Stage DIM]` to be merged into the destination table in a later component.

[SQL – Merge Dimension] Will be loading all the data into their final target tables from the data in the initial staging tables.

This second pattern for loading dimension tables has some similarities and some differences to the fact loading pattern. First, hashing is being applied. Hashing is used to accurately check for updates to rows, note that this is being applied to dimension rows only. Lastly, the different Type 1 or Type 2 rows are split into different pipelines and being counted by the Biml orchestration framework.

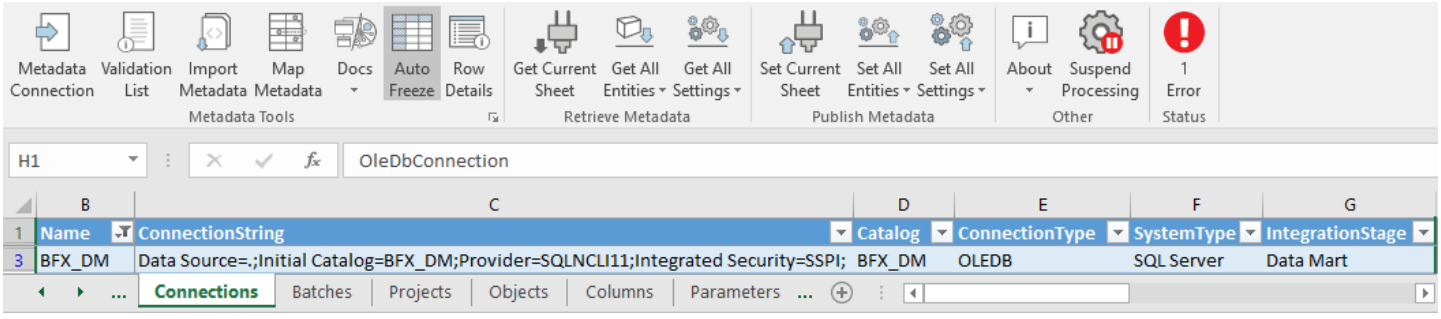
Implementing a Data Mart

Connections and Projects

When beginning to structure a Data Mart loading project in BimlFlex, there are certain metadata entities required in order to Build the solution.

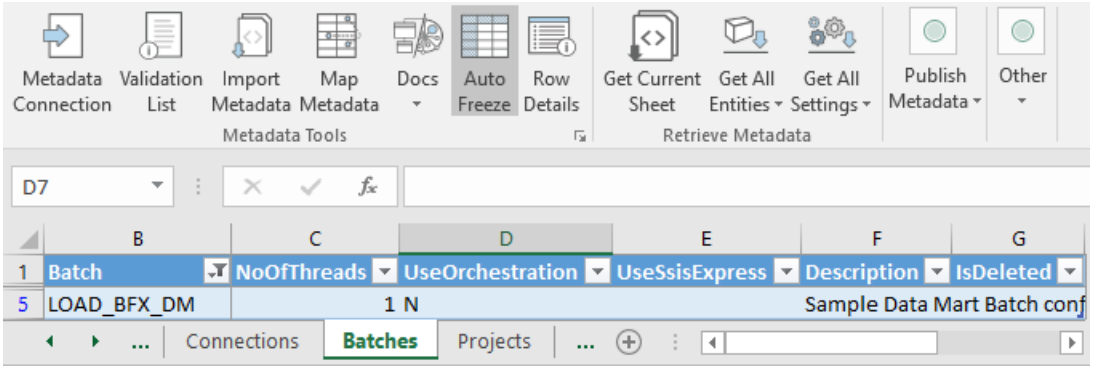
The connection has to be defined with the Integration Stage properly defined.

Below is an example connection to the Data Mart.



A Batch is required, this controls the number of threads used and if orchestration is used.

Here the Batch is named `LOAD_BFX_DM`, short for Load BimlFlex Data Mart.



The Project can be defined once the Connection and the Batch has been defined.

There are a few options for Project metadata and how a Data Mart project is loaded. These options are outlined in the table below.

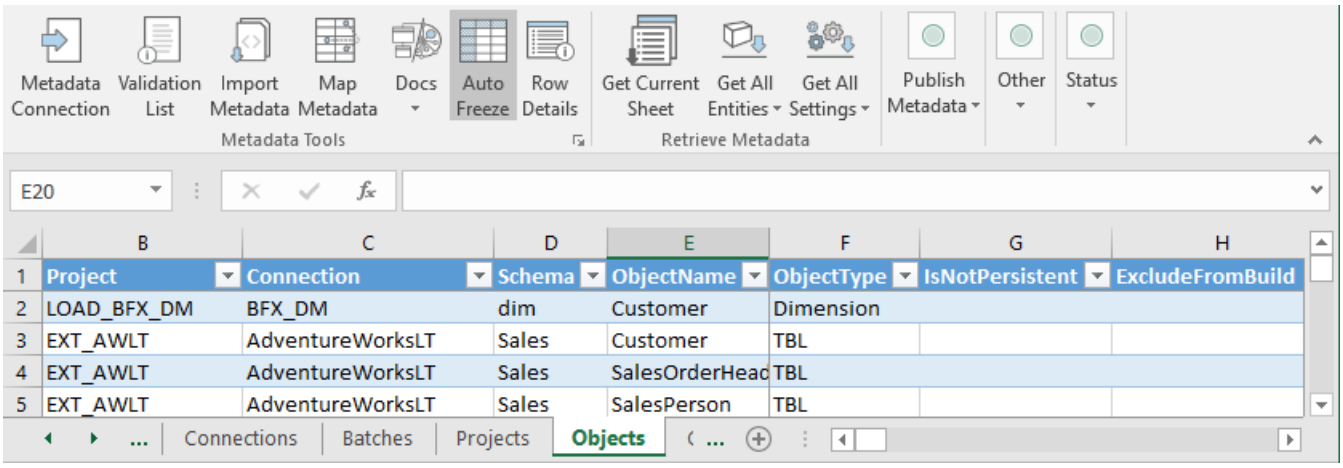
The project architecture and the source for the Data Mart depends on the solution architecture. If Data Vault methodology is being used, the source will be a connection pointing at the Data Vault. If a two tier implementation is used (Source to Staging to Data Mart), this connection is the staging database connection.

ATTRIBUTE NAME	EXAMPLE PROJECT ATTRIBUTES
Project	LOAD_BFX_DM
Source Connection	BFX_STG/BFX_DV
Stage Connection	
Persistent Stage Connection (Optional)	
Target Stage Connection	BFX_DM
Target Connection (destination of the data: DW, DV etc)	BFX_DM
Batch Name	BFX_DM
Parent Batch (Optional)	
Integration Template	Source -> Target

The example is using the Source to Target Integration Template. BimlFlex uses the Connection metadata Integration Stage set earlier, `Data Mart`, to derive what BimlFlex ETL pattern to apply.

Dimension Object and Columns

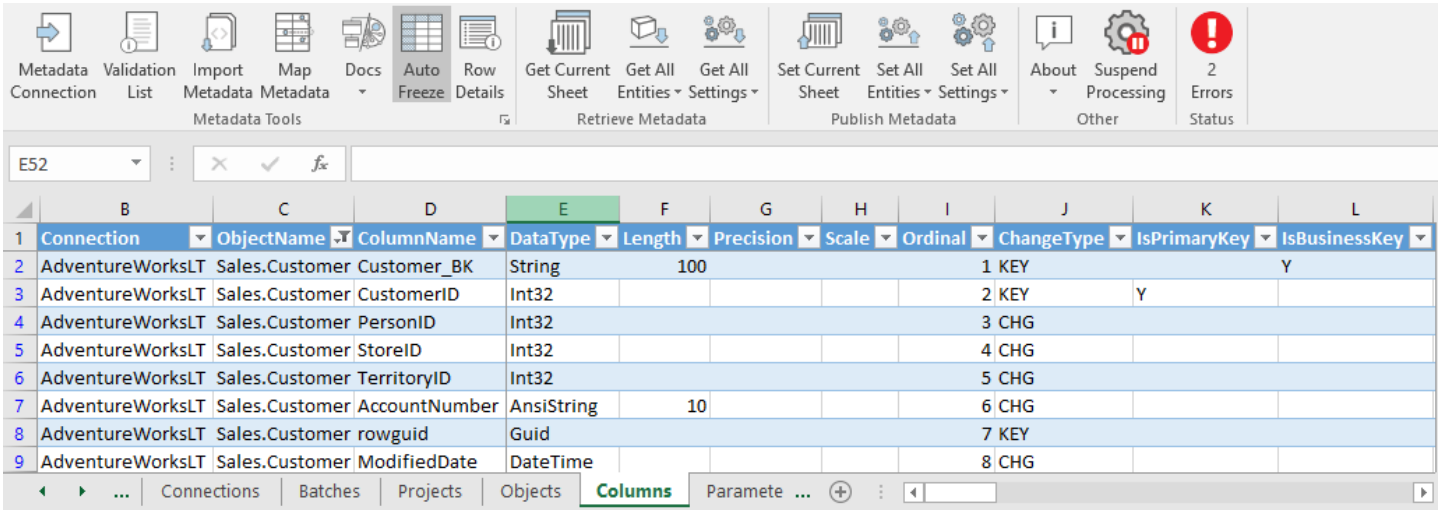
once the prerequisite metadata is completed the Objects can be added.



In the above example above there is a new Customer dimension being defined in the first row. The object type is Dimension and its connection is pointing to the Data Mart.

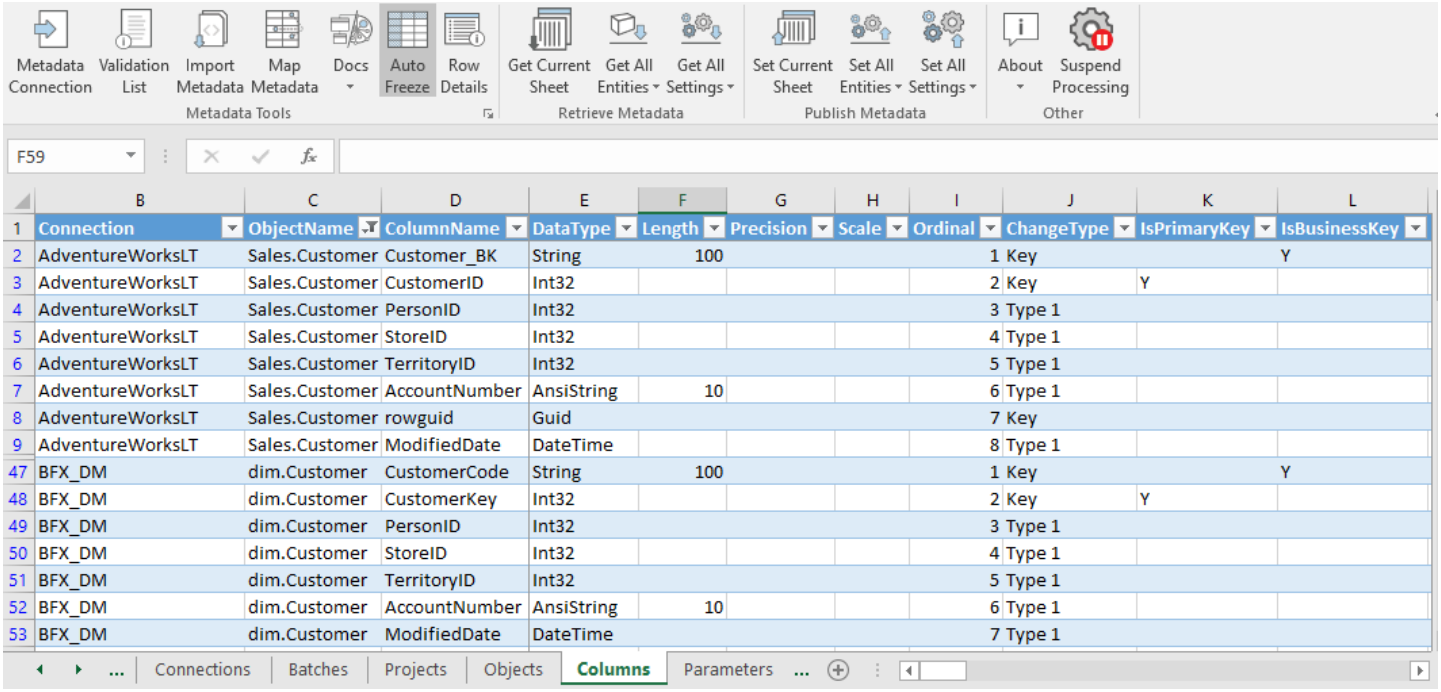
The next step is to do Source to Target mapping of columns from the source table to the destination dimension.

In the following example the columns of the source table Customer are in the Sales schema.



At this point, a user will want to define the columns of the Customer dimension.

Below is an example of the columns a user would define in the customer dimension.



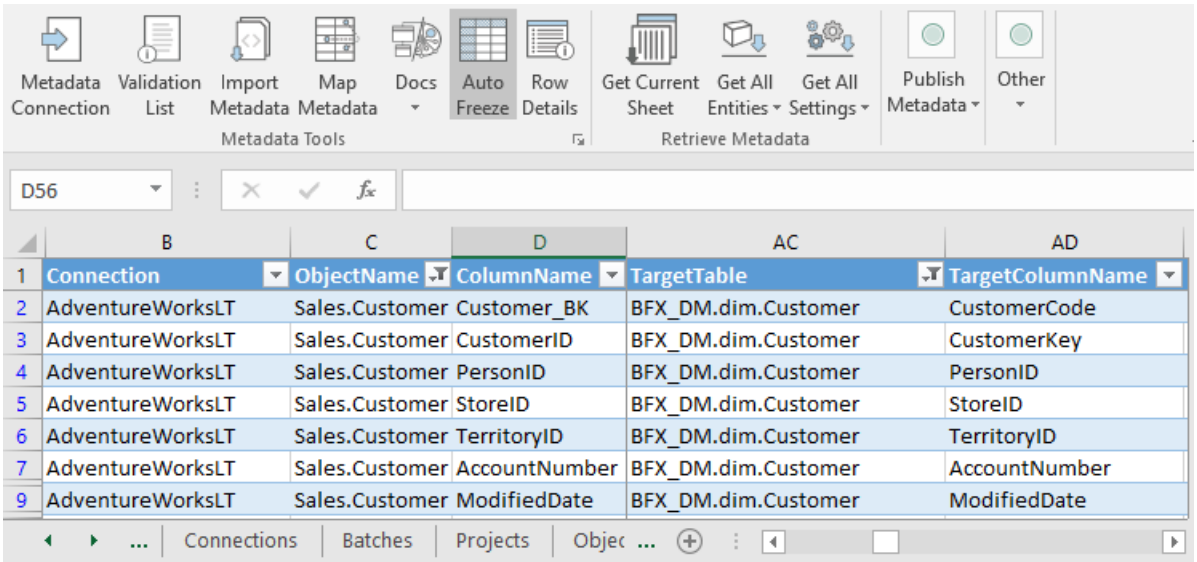
	B	C	D	E	F	G	H	I	J	K	L
1	Connection	ObjectName	ColumnName	DataType	Length	Precision	Scale	Ordinal	ChangeType	IsPrimaryKey	IsBusinessKey
2	AdventureWorksLT	Sales.Customer	Customer_BK	String	100			1	Key		Y
3	AdventureWorksLT	Sales.Customer	CustomerID	Int32				2	Key	Y	
4	AdventureWorksLT	Sales.Customer	PersonID	Int32				3	Type 1		
5	AdventureWorksLT	Sales.Customer	StoreID	Int32				4	Type 1		
6	AdventureWorksLT	Sales.Customer	TerritoryID	Int32				5	Type 1		
7	AdventureWorksLT	Sales.Customer	AccountNumber	AnsiString	10			6	Type 1		
8	AdventureWorksLT	Sales.Customer	rowguid	Guid				7	Key		
9	AdventureWorksLT	Sales.Customer	ModifiedDate	DateTime				8	Type 1		
47	BFX_DM	dim.Customer	CustomerCode	String	100			1	Key		Y
48	BFX_DM	dim.Customer	CustomerKey	Int32				2	Key	Y	
49	BFX_DM	dim.Customer	PersonID	Int32				3	Type 1		
50	BFX_DM	dim.Customer	StoreID	Int32				4	Type 1		
51	BFX_DM	dim.Customer	TerritoryID	Int32				5	Type 1		
52	BFX_DM	dim.Customer	AccountNumber	AnsiString	10			6	Type 1		
53	BFX_DM	dim.Customer	ModifiedDate	DateTime				7	Type 1		

Note that the columns are fairly similar but they have been set in such a way that the business key in the source table will line up with the CustomerCode column and the primary key of the source will be the CustomerKey of the dimension table.

This is one pattern available when modelling star schemas. Each business key in source becomes the `Code` column in the Data Mart and each Data Mart table will include a unique integer key which is the `Key` column. The keys used can either be sequence numbers from Identity columns, or for Data Vault sources, the Hashed Surrogate Keys generated in the Data Vault entities or Bridge and Point In Time tables.

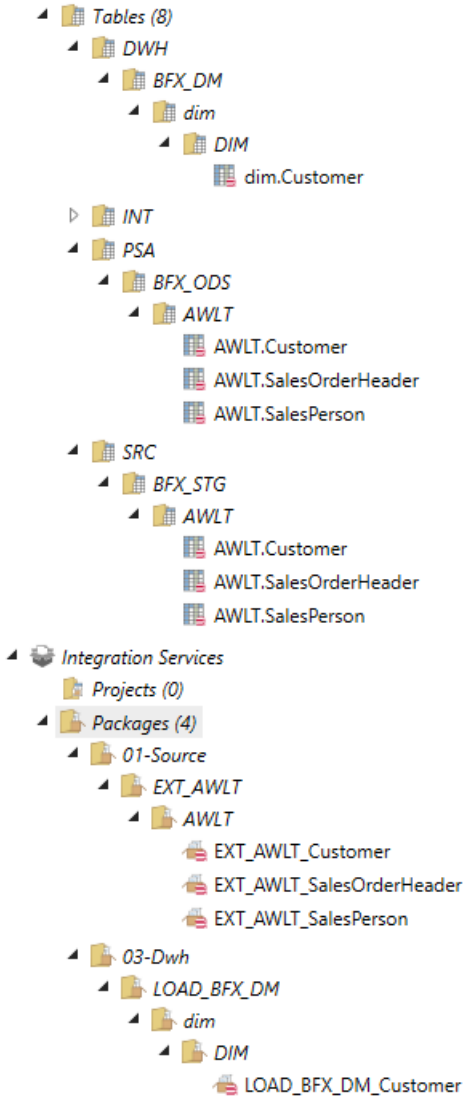
Once Source and Target metadata is available the mapping between them can be completed.

Below is an example of the Source to Target definitions of the `Sales.Customer` columns.



	B	C	D	AC	AD
1	Connection	ObjectName	ColumnName	TargetTable	TargetColumnName
2	AdventureWorksLT	Sales.Customer	Customer_BK	BFX_DM.dim.Customer	CustomerCode
3	AdventureWorksLT	Sales.Customer	CustomerID	BFX_DM.dim.Customer	CustomerKey
4	AdventureWorksLT	Sales.Customer	PersonID	BFX_DM.dim.Customer	PersonID
5	AdventureWorksLT	Sales.Customer	StoreID	BFX_DM.dim.Customer	StoreID
6	AdventureWorksLT	Sales.Customer	TerritoryID	BFX_DM.dim.Customer	TerritoryID
7	AdventureWorksLT	Sales.Customer	AccountNumber	BFX_DM.dim.Customer	AccountNumber
9	AdventureWorksLT	Sales.Customer	ModifiedDate	BFX_DM.dim.Customer	ModifiedDate

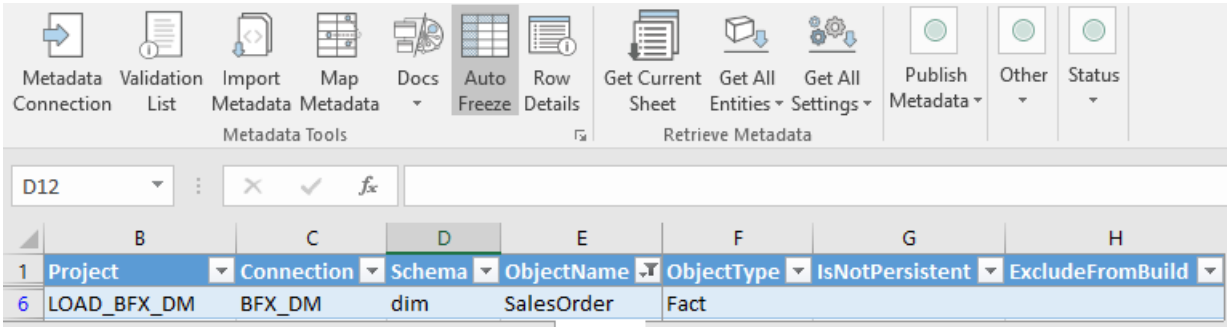
Refresh the metadata in BimlStudio to review, build and test.



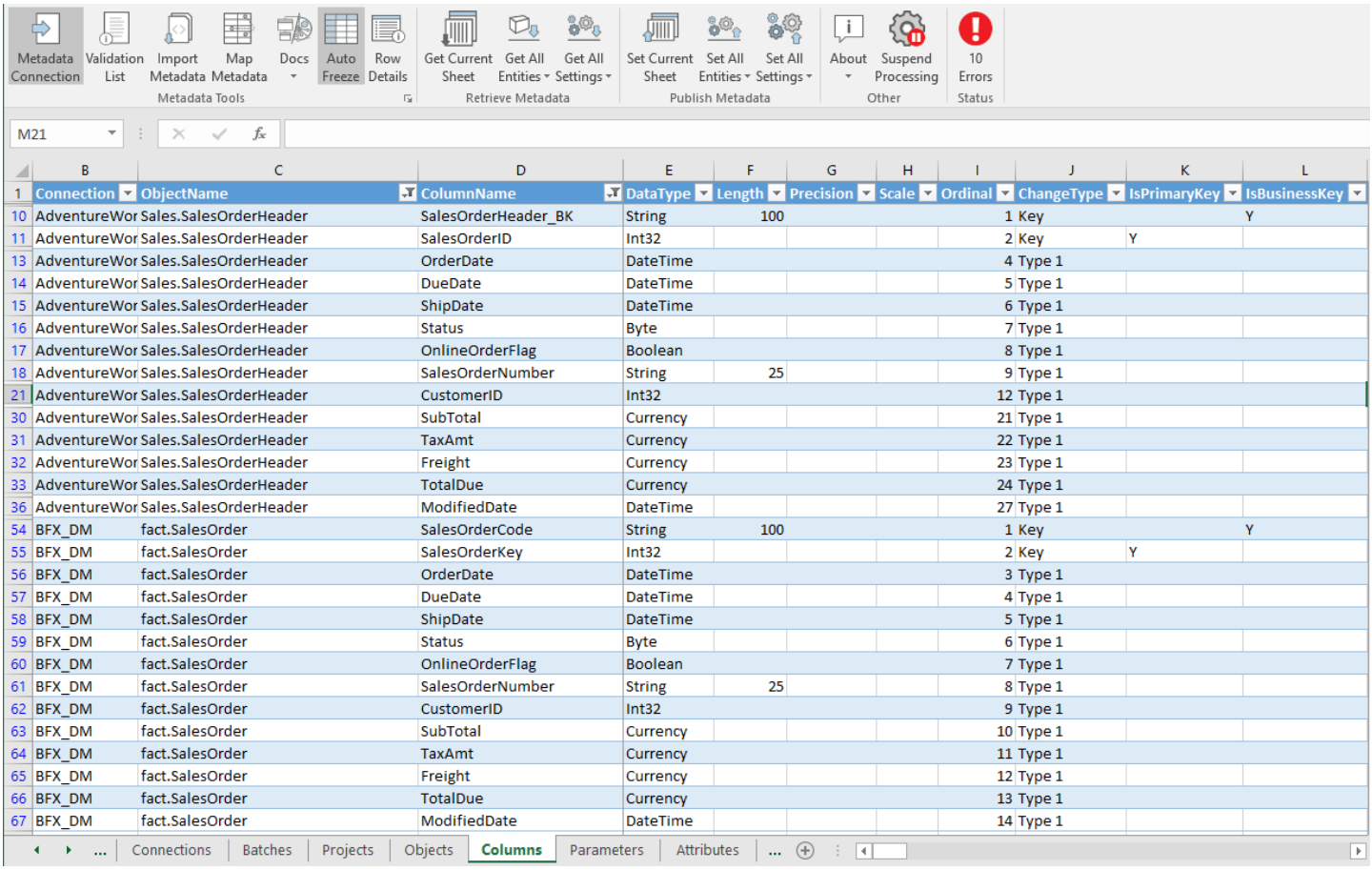
Fact Object and Columns

Similar to above the fact metadata steps are very similar to before, the example resumes at the point of creating the fact object because the fact itself can use the same Data Mart loading project as specified in the previous examples.

Below is a sample of the fact object metadata, note that the first column Project has the same `LOAD_BFX_DM` project.

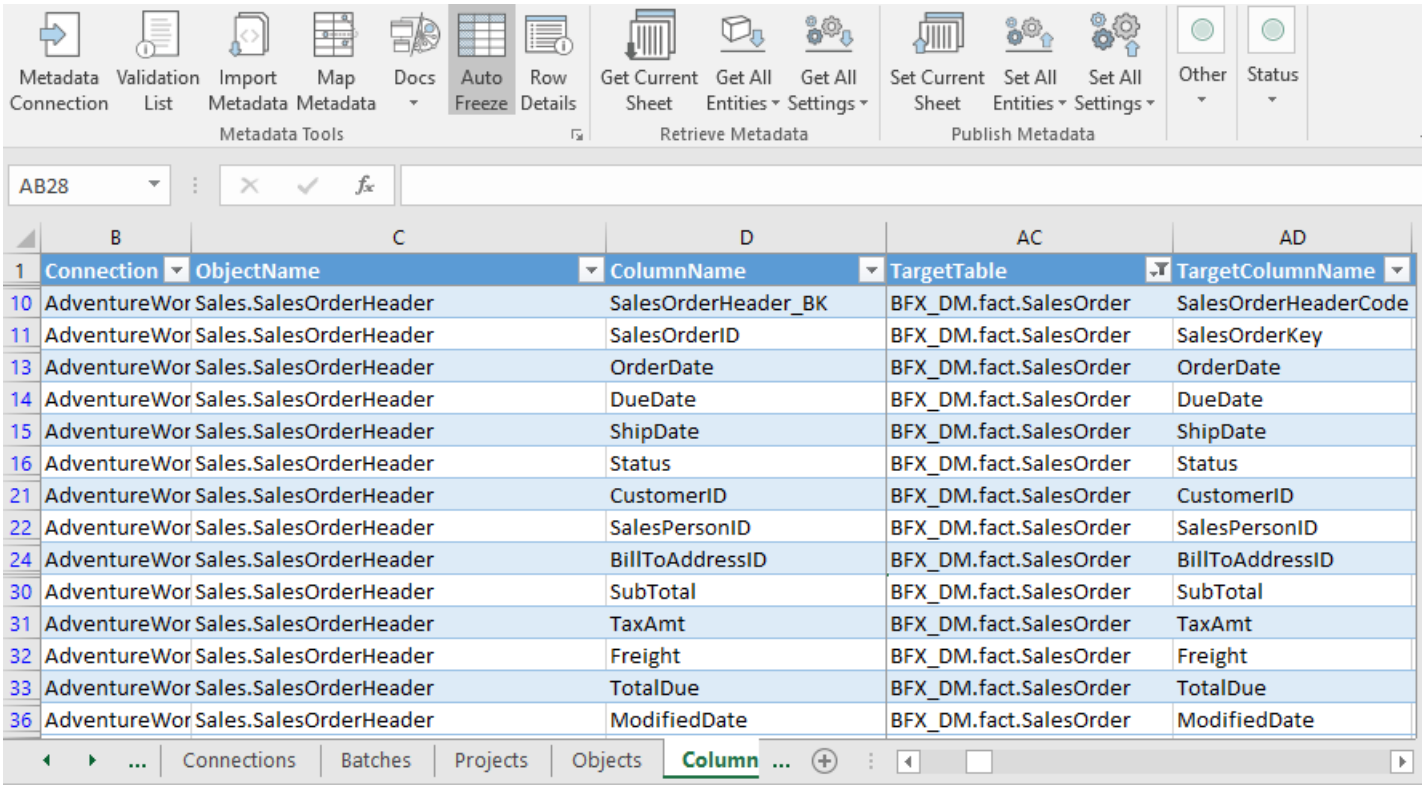


From here, the main objective is to definite the columns in the new Fact table and do Source to Target mappings for these columns. In this example, the `SalesOrderHeader` will be the source of the Fact.



	B	C	D	E	F	G	H	I	J	K	L
1	Connection	ObjectName	ColumnName	DataType	Length	Precision	Scale	Ordinal	ChangeType	IsPrimaryKey	IsBusinessKey
10	AdventureWor	Sales.SalesOrderHeader	SalesOrderHeader_BK	String	100			1	Key		Y
11	AdventureWor	Sales.SalesOrderHeader	SalesOrderID	Int32				2	Key	Y	
13	AdventureWor	Sales.SalesOrderHeader	OrderDate	DateTime				4	Type 1		
14	AdventureWor	Sales.SalesOrderHeader	DueDate	DateTime				5	Type 1		
15	AdventureWor	Sales.SalesOrderHeader	ShipDate	DateTime				6	Type 1		
16	AdventureWor	Sales.SalesOrderHeader	Status	Byte				7	Type 1		
17	AdventureWor	Sales.SalesOrderHeader	OnlineOrderFlag	Boolean				8	Type 1		
18	AdventureWor	Sales.SalesOrderHeader	SalesOrderNumber	String	25			9	Type 1		
21	AdventureWor	Sales.SalesOrderHeader	CustomerID	Int32				12	Type 1		
30	AdventureWor	Sales.SalesOrderHeader	SubTotal	Currency				21	Type 1		
31	AdventureWor	Sales.SalesOrderHeader	TaxAmt	Currency				22	Type 1		
32	AdventureWor	Sales.SalesOrderHeader	Freight	Currency				23	Type 1		
33	AdventureWor	Sales.SalesOrderHeader	TotalDue	Currency				24	Type 1		
36	AdventureWor	Sales.SalesOrderHeader	ModifiedDate	DateTime				27	Type 1		
54	BFX_DM	fact.SalesOrder	SalesOrderCode	String	100			1	Key		Y
55	BFX_DM	fact.SalesOrder	SalesOrderKey	Int32				2	Key	Y	
56	BFX_DM	fact.SalesOrder	OrderDate	DateTime				3	Type 1		
57	BFX_DM	fact.SalesOrder	DueDate	DateTime				4	Type 1		
58	BFX_DM	fact.SalesOrder	ShipDate	DateTime				5	Type 1		
59	BFX_DM	fact.SalesOrder	Status	Byte				6	Type 1		
60	BFX_DM	fact.SalesOrder	OnlineOrderFlag	Boolean				7	Type 1		
61	BFX_DM	fact.SalesOrder	SalesOrderNumber	String	25			8	Type 1		
62	BFX_DM	fact.SalesOrder	CustomerID	Int32				9	Type 1		
63	BFX_DM	fact.SalesOrder	SubTotal	Currency				10	Type 1		
64	BFX_DM	fact.SalesOrder	TaxAmt	Currency				11	Type 1		
65	BFX_DM	fact.SalesOrder	Freight	Currency				12	Type 1		
66	BFX_DM	fact.SalesOrder	TotalDue	Currency				13	Type 1		
67	BFX_DM	fact.SalesOrder	ModifiedDate	DateTime				14	Type 1		

Now the key will be to link the fact source columns and the target fact table and target columns as shown in the previous customer dimension example.



	B	C	D	AC	AD
1	Connection	ObjectName	ColumnName	TargetTable	TargetColumnName
10	AdventureWor	Sales.SalesOrderHeader	SalesOrderHeader_BK	BFX_DM.fact.SalesOrder	SalesOrderHeaderCode
11	AdventureWor	Sales.SalesOrderHeader	SalesOrderID	BFX_DM.fact.SalesOrder	SalesOrderKey
13	AdventureWor	Sales.SalesOrderHeader	OrderDate	BFX_DM.fact.SalesOrder	OrderDate
14	AdventureWor	Sales.SalesOrderHeader	DueDate	BFX_DM.fact.SalesOrder	DueDate
15	AdventureWor	Sales.SalesOrderHeader	ShipDate	BFX_DM.fact.SalesOrder	ShipDate
16	AdventureWor	Sales.SalesOrderHeader	Status	BFX_DM.fact.SalesOrder	Status
21	AdventureWor	Sales.SalesOrderHeader	CustomerID	BFX_DM.fact.SalesOrder	CustomerID
22	AdventureWor	Sales.SalesOrderHeader	SalesPersonID	BFX_DM.fact.SalesOrder	SalesPersonID
24	AdventureWor	Sales.SalesOrderHeader	BillToAddressID	BFX_DM.fact.SalesOrder	BillToAddressID
30	AdventureWor	Sales.SalesOrderHeader	SubTotal	BFX_DM.fact.SalesOrder	SubTotal
31	AdventureWor	Sales.SalesOrderHeader	TaxAmt	BFX_DM.fact.SalesOrder	TaxAmt
32	AdventureWor	Sales.SalesOrderHeader	Freight	BFX_DM.fact.SalesOrder	Freight
33	AdventureWor	Sales.SalesOrderHeader	TotalDue	BFX_DM.fact.SalesOrder	TotalDue
36	AdventureWor	Sales.SalesOrderHeader	ModifiedDate	BFX_DM.fact.SalesOrder	ModifiedDate

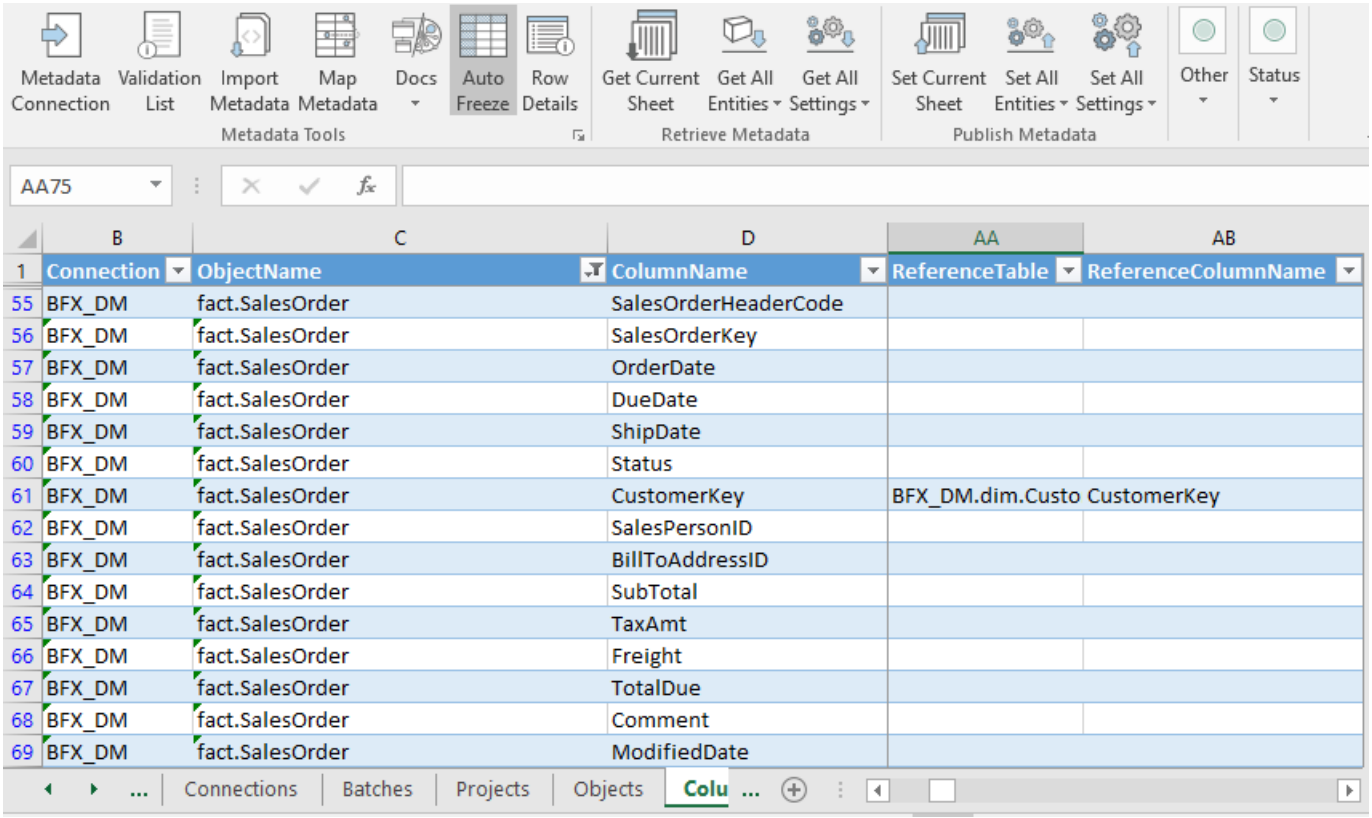
Now that the source, destination and target column metadata in place, there is one more item to cover regarding modelling Data Marts in BimlFlex and that is the use of foreign keys.

In the Data Mart, generally, each foreign key points from fact to dimension. In the loading pattern, the foreign key is set by performing a lookup on the business key in a dimension, taking the corresponding dimension's key and then setting that as the foreign key in the

fact.

BimlFlex still automates all this functionality for us but the user still needs to indicate how this is done. This is done by setting the reference columns of the fact table in metadata.

The example below will demonstrate how to add a foreign key to the customer dimension.



	B	C	D	AA	AB
1	Connection	ObjectName	ColumnName	ReferenceTable	ReferenceColumnName
55	BFX_DM	fact.SalesOrder	SalesOrderHeaderCode		
56	BFX_DM	fact.SalesOrder	SalesOrderKey		
57	BFX_DM	fact.SalesOrder	OrderDate		
58	BFX_DM	fact.SalesOrder	DueDate		
59	BFX_DM	fact.SalesOrder	ShipDate		
60	BFX_DM	fact.SalesOrder	Status		
61	BFX_DM	fact.SalesOrder	CustomerKey	BFX_DM.dim.Custo	CustomerKey
62	BFX_DM	fact.SalesOrder	SalesPersonID		
63	BFX_DM	fact.SalesOrder	BillToAddressID		
64	BFX_DM	fact.SalesOrder	SubTotal		
65	BFX_DM	fact.SalesOrder	TaxAmt		
66	BFX_DM	fact.SalesOrder	Freight		
67	BFX_DM	fact.SalesOrder	TotalDue		
68	BFX_DM	fact.SalesOrder	Comment		
69	BFX_DM	fact.SalesOrder	ModifiedDate		

Note that since there is a foreign key pointing to the customer dimension key, the Fact's CustomerID column should be renamed to CustomerKey.

Refresh the metadata in BimlStudio to review, build and test.

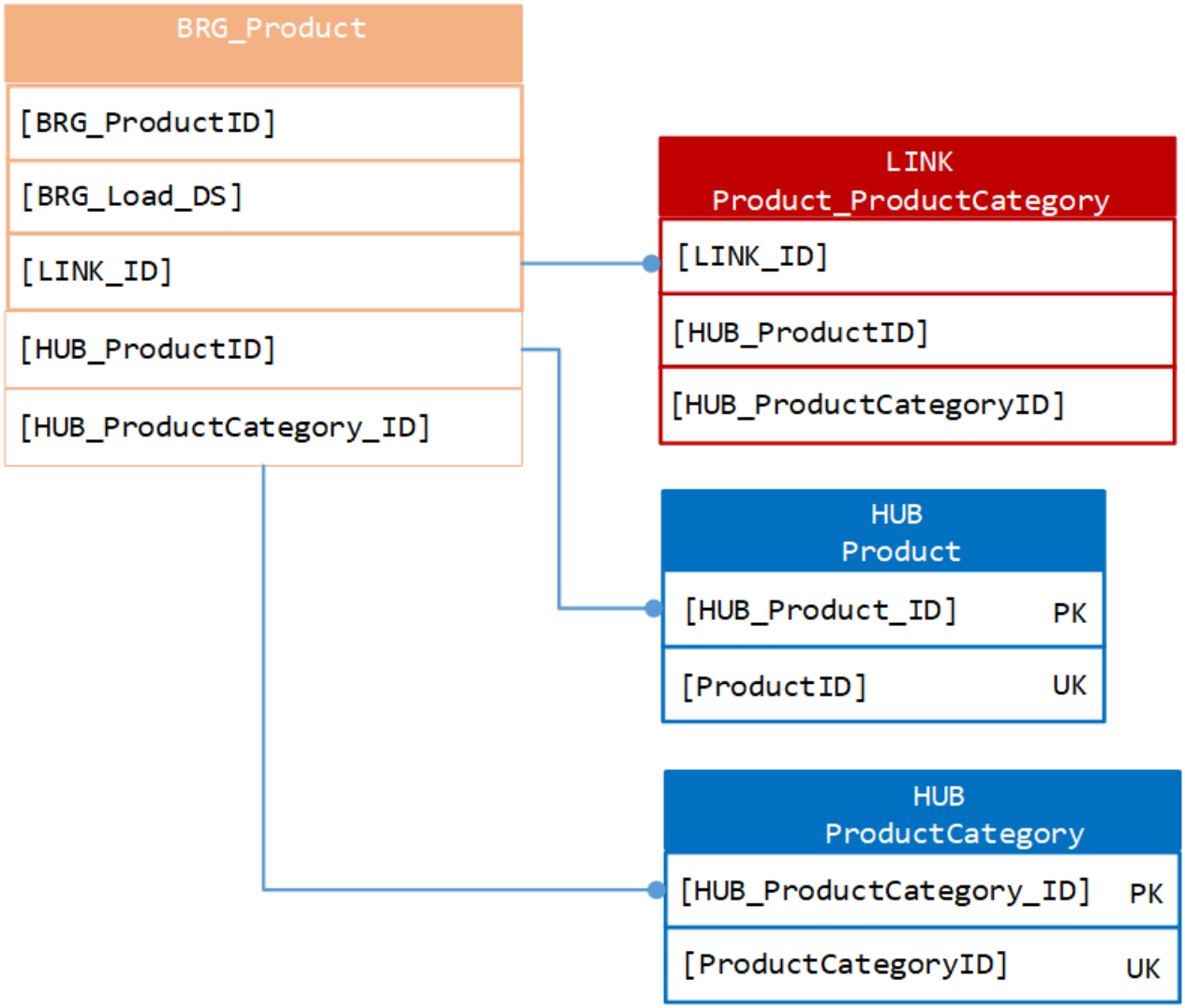
Point in Time Tables and Bridge Tables

Introduction to PIT and BRG

Point in time tables and bridge tables are ancillary tables used to make querying from a data vault significantly easier and more efficient. How these tables are utilised is more dependent on each individual BI solution but how they are implemented in BimlFlex is fairly straight forward. This example will point out the Hubs and Links of interest and BimlFlex generate the necessary components.

Bridge tables Introduction

The purpose of a bridge table is to ease the work involved in transforming and accessing data in a data vault, especially when attempting to load this data in the Data Mart. As the name implies a bridge table joins across multiple entities in the data vault and brings them together into one table. Bridge tables deal in Hubs and Links and the reason being is that generally a bridge table is needed because in the Data Mart layer it is expected to make use of the relationships that exist in the source systems, however in the data vault these relationships are decomposed across potentially multiple Hubs and Links.



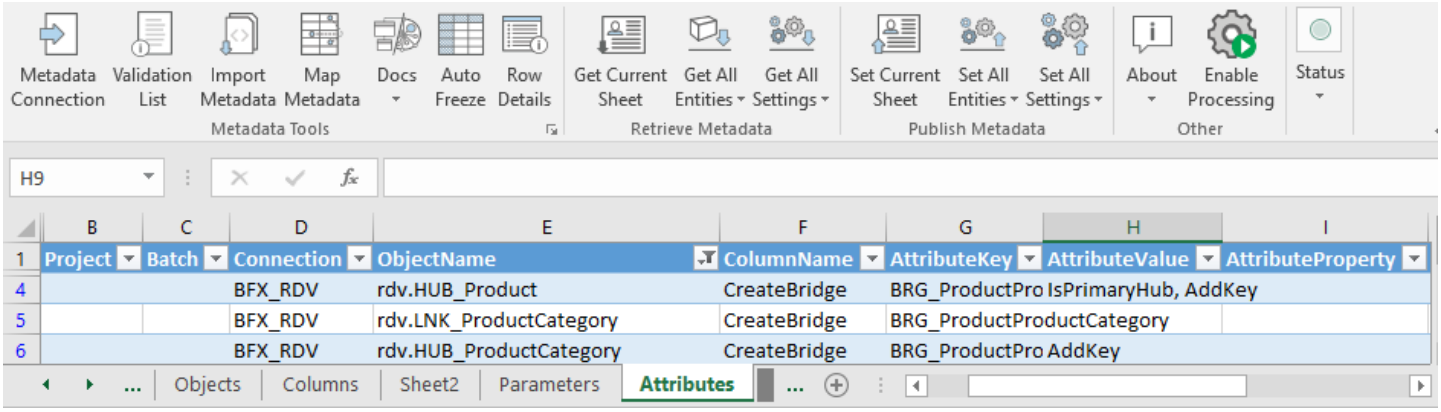
The above example shows a bridge table on the left joining two hubs and a link table together, one row in the bridge table will contain all the surrogate keys needed to start querying across these entities without a large amount of SQL being needed.

Configuring a Bridge Table

Now that it has been shown why a user may want a bridge table, let's focus on how to join these various entities in the data vault and bring the relevant keys together in a bridge table.

Fortunately, configuring and deploying a bridge table is a fairly simple process. BimlFlex is able to bring together all the data vault tables required in a given bridge table by inspecting the metadata behind the scenes that should at this stage already be in place.

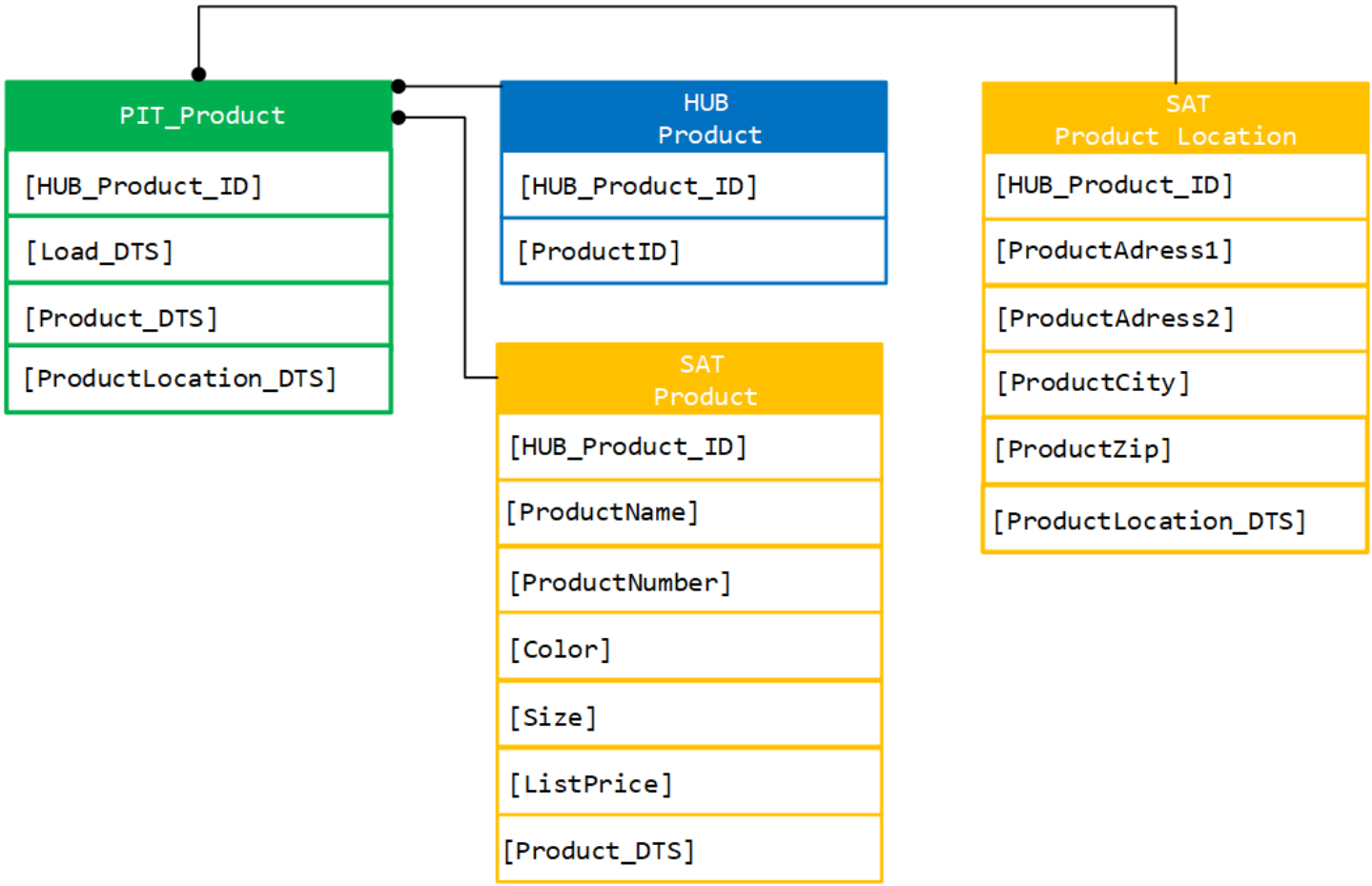
Creating a bridge table requires a user to know a couple of things about a given entity within the data vault. The first action a user needs to take is to choose the primary Hub. This means that of the set of hubs being joined together, which one are will be chosen to be the starting point. In the example, the Hubs product and product category are joined together. To do this the user will need to include the product and product category Link.



Note that as in the screen above, CreateBridge is used to indicate what is happening with these columns. In the AttributeKey column, a user can name the Bridge table and In the AttributeValue column, a user can indicate the primary Hub and chose which other Hubs need to be added with their surrogate keys to the bridge table. This is done by entering in the keywords IsPrimaryHub and AddKey for which Hubs that need to be added to this bridge table. Once this metadata is in place all is required, then, is to inspect, build and test this new structure through BimlStudio.

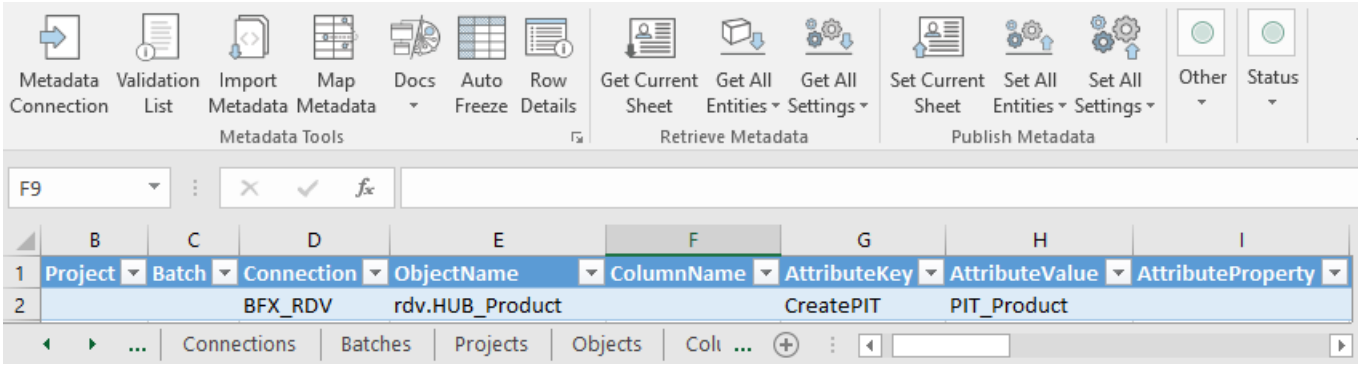
Point in Time Tables Introduction

Similar to how a bridge table brings together related data across different entities in the data vault, the Point In Time (PIT) tables job is to bring together all the contextual satellite data where each row represents a time slice. This is particularly helpful when attempting to extract historical data into the Data Mart. The way a pit table does this is by joining all the relevant surrogate keys across all the satellites surrounding one Hub and aligns these rows based on the date time stamp.



Configuring a Point In Time Table

To configure a Point in Time (PIT) table, the user needs to select the Hub that they want to create the point in time table for. BimlFlex will find all the related satellites to this hub and create the point in time table necessary to bring all the items together.



Creating a PIT table is straightforward. BimlFlex is able to obtain the majority of its required attributes from the existing Data Vault metadata. In order to create a PIT table for a given Hub, all that is needed is the name of the Hub and the connection for its Data Vault.

Using the Product entity as an example, the connection is `BFX_RDV` for the Raw Data Vault and the object name is `rdv.HUB_Project`. Set the attribute key to `CreatePIT` and the attribute value to the name of the PIT table.

Only adding the Hub to the configuration will include all its Satellites in the PIT. if a subset of the Satellites are to be included, add these Satellite tables to the attributes with the same PIT table name.

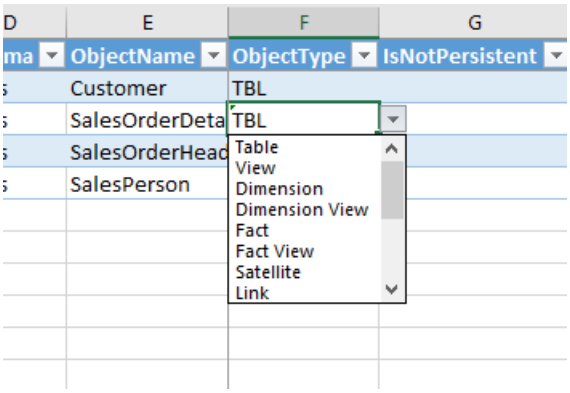
Refresh the metadata in BimlStudio to review, build and test.

Implementing Business Rules

In terms of implementing business rules, a user can apply business rules either in the business vault or in the presentation layer depending on the overall architecture. In terms of the object type to contain these rules, they can either be implemented in views or in extension points. Fortunately, in BimlFlex this doesn't change the source to target template style in any significant way. If a user chooses to implement business rules in views, the object type of the source object can change a source to be a variety of items.

The decision as to where to implement business rules doesn't necessarily change the process of implementation in BimlFlex.

Below is an example of the variety of source types available, note that view or regular tables are acceptable.



When implementing business rules through extension points, this will generally be done in the form of a post process that is entered in especially for a given object package. Post process extension points get executed at the end of a particular load. This approach is less common than storing business rules in a view but is just as feasible.

Next Steps

Once the presentation layer is completed, consideration should be made towards the approach taken in the analytical layer. The final part of this solution will be to use a tool to deliver this data to decision makers within the organisation in a format that allows them to perform a proper analysis of the organisation's data.

Possible options are to; create cubes or tabular models for Excel, Power BI and classical reporting, reporting views for Reporting

Services or snow flakes for MicroStrategy, flat views for Qlik and Tableau etc.

Export To File

An outline for using BimlFlex to export data out of a database into files. The export process supports the following template patterns:

- source to file
- source to zip file

These templates provide the necessary ETL transformations and destination components needed to successfully extract to a file.

File Options

There are several file format options available for file exports. Extension Points can override the behaviour for custom requirements.

The Objects metadata spreadsheet in BimlFlex Excel contains the following settings for file formats.

Flat File Types

ATTRIBUTE NAME	DESCRIPTION
Delimited	Fields in row are delimited by character
FixedWidth	Each field/row has a fixed width
RaggedRight	The last field in row can have variable width, delimited by new row character

Data Rows to Skip

This is the count of rows from the start of the file to the header row or the start of data.

TextQualifier

What character delimits the text in a given column. E.g. `"text"`, `'text'`, etc.

Delimiters

Column and row delimiters and Last Column Delimiter

ATTRIBUTE NAME	DESCRIPTION	
CRLF	Carriage Return, Line Feed - ASCII 13, 10	
CR	Carriage Return - ASCII 13	
LF	Line Feed - ASCII 10	
Semicolon	;	
Comma	,	
Tab	HT - ACSII 09	
VerticalBar		

ATTRIBUTE NAME	DESCRIPTION	
UnitSeparator	US - ASCII 31	
Custom	Any valid custom delimiter	

Code Page

The code page to use for the generated file. This is a standard SSIS attribute. Reference the following Microsoft documentation for more information:

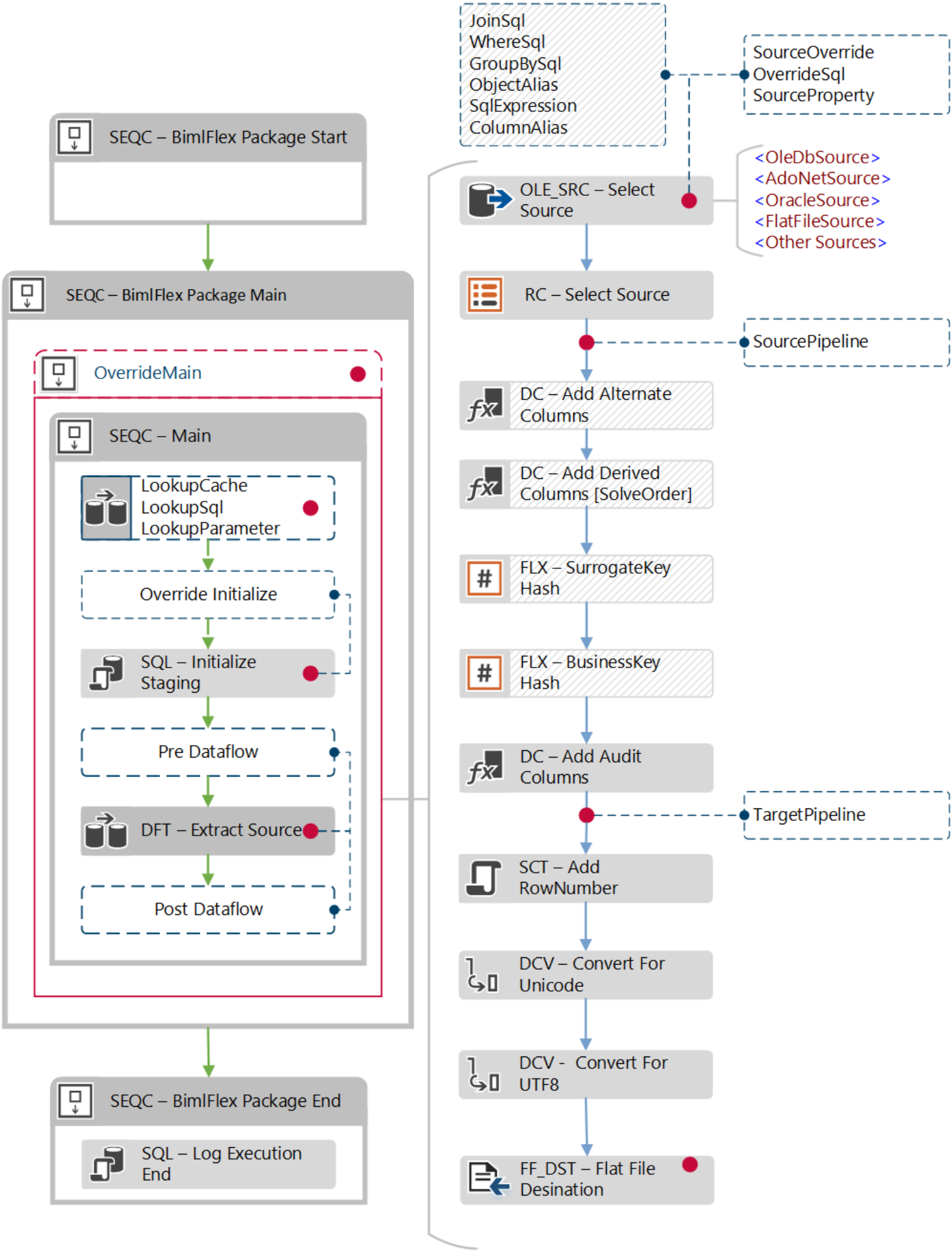
MSDN - Code Page Identifiers

Source to File

ETL Pattern Overview

The File Extract template extracts data from the data warehouse or other database source into a flat file destination.

The components and ETL pattern are similar to a regular source to target pattern. There are additional metadata requirements for defining the destination file.



Adding Source to File Metadata

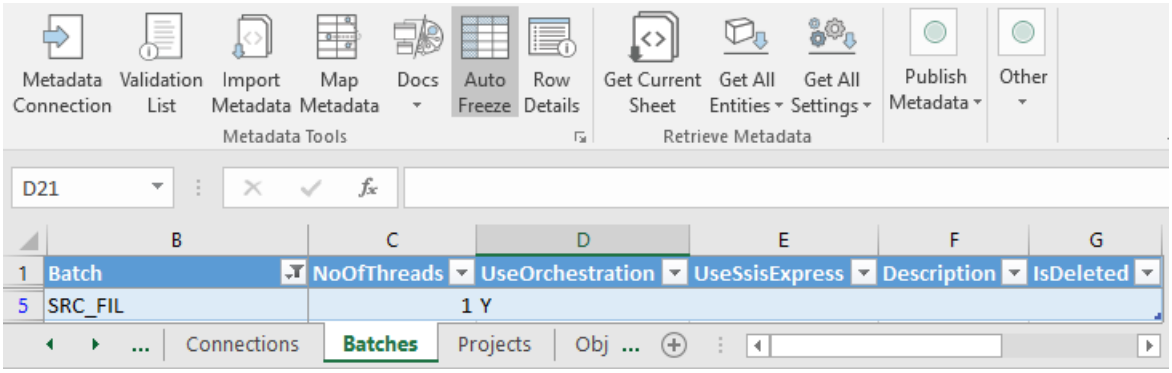
Connections

The first step is setting up the connection attributes that are relevant to setting a flat file connection. In the source to file scenario, this would be the intended destination connection. For connections ensure to choose "Staging" as the Integration stage. Some example connection attributes that mimic this are shown below

ATTRIBUTE NAME	EXAMPLE PROJECT ATTRIBUTES
Name	TransactionsFile
ConnectionString	C:\ExportFiles
Catalog	Transactions
ConnectionType	FILE
SystemType	"File Delimited"\"File Ragged Right"\"Excel"
IntegrationStage	Staging
RecordSource	TF
FilePath	C:\ImportFiles\
FilePattern	Transactions*.*
NoOfThreads	0

Batches

The second metadata entity that a user will need to fill in is the Batch entity. The Batch for a project is the parent package that executes the child packages that in turn perform the data load. Add a new batch with an appropriate name, number of threads, and option whether or not to use orchestration. An example of this batch metadata is shown below



Projects

Below is a screen shot of an example project. Generally, it will follow the usual source to target metadata with a couple of differences. In this case, ensure that `Source -> File Extract` is defined for the integration template. This is important as it indicates to the BimlFlex framework what the type of project is an its purpose.

Project	SourceConnection	StageConnection	TargetConnection
EXT_Transactions	TransactionsFile	DWH_SRC	TransactionsFile

Objects and Columns

Now to add both the source file definition and the target table definition in the objects metadata spreadsheet. Below is an example of the target transactions file that is intend to be loaded to. Note that, so far, specified in the first row, are the column names, the type of flat file, whether to skip any empty rows at the beginning of the file, row delimiter type, column delimiter, line delimiter or last column delimiter, code page and finally whether the file is Unicode or not. Supplying BimlFlex with this metadata will ensure the correct file formats are being loaded further on in the development and build stages.

Project	Connection	Schema	ObjectName	IsColumnNamesInFirstDataRow
EXT_Transactions	TransactionsFile	tra	Transactions	Y

FlatFileType	DataRowsToSkip	TextQualifer	RowDelimiter	ColumnDelimiter
Delimited	0		CRLF	Comma

LastColumnDelimiter	CodePage	IsUnicode
CRLF	1252	Y

Each destination File defined in the objects sheet require all destination columns to be defined.

Each source column need to be mapped to its respective destination column. This is done by adding the TargetTable and TargetColumnName attributes to the columns for the source table.

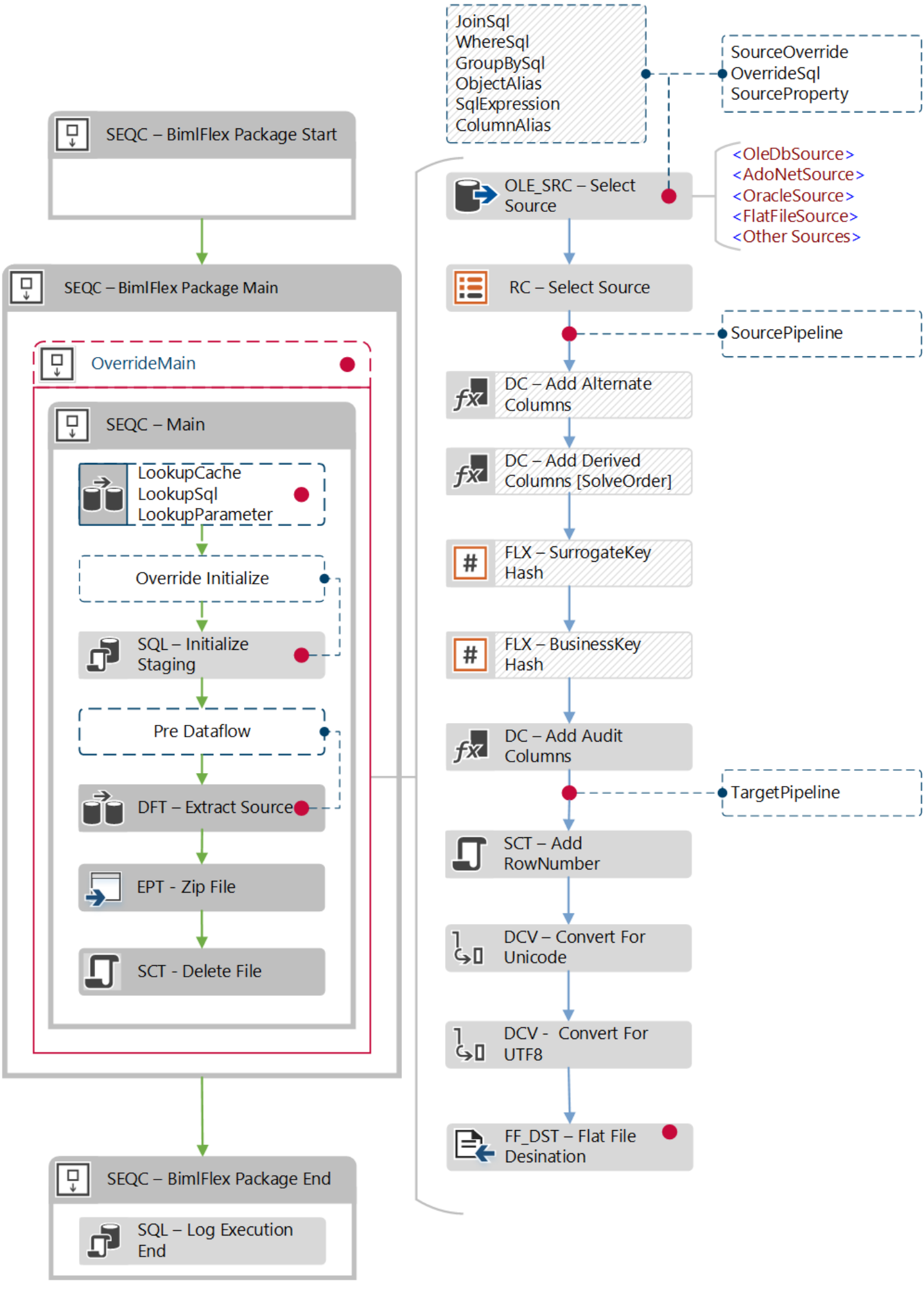
TargetTable	TargetColumnName
TransactionsFile.tra.Transactions	CustomerTransactionCode
TransactionsFile.tra.Transactions	CustomerCashCode
TransactionsFile.tra.Transactions	CustomerKey
TransactionsFile.tra.Transactions	RelatedCode
TransactionsFile.tra.Transactions	CustomerTransactionTypeKey
TransactionsFile.tra.Transactions	TransactionCreditAmount
TransactionsFile.tra.Transactions	TransactionDebitAmount
TransactionsFile.tra.Transactions	CustomerCreditAmount
TransactionsFile.tra.Transactions	CustomerDebitAmount
TransactionsFile.tra.Transactions	CustomerExchangeRate
TransactionsFile.tra.Transactions	PartClearedAmount
TransactionsFile.tra.Transactions	PartClearedBy
TransactionsFile.tra.Transactions	PartClearedDate
TransactionsFile.tra.Transactions	FeeAmount
TransactionsFile.tra.Transactions	FeeCurrencyKey

Once the metadata has been set in BimlFlex Excel, refresh the solution metadata in BimlStudio, build the solution and test.

Source to Zip File

BimlFlex provides a separate template for destinations that require the generated file to be zipped. In addition to the previous file export template, this adds a file compression step at the end of the control flow.

ETL Pattern Overview



7-Zip Application

The default BimlFlex templates use the open source and freely available 7-zip application for compressing files. The main reason is that it is freely available to use within a commercial environment, without registration or licensing. It also has an excellent command line interface that can be used through automation tools like SSIS.

A prerequisite for using the export to compressed file template is to have 7-Zip installed on the server running the packages.

7-Zip can be downloaded from http://www.7-zip.org

Adding Source to Zip File Metadata

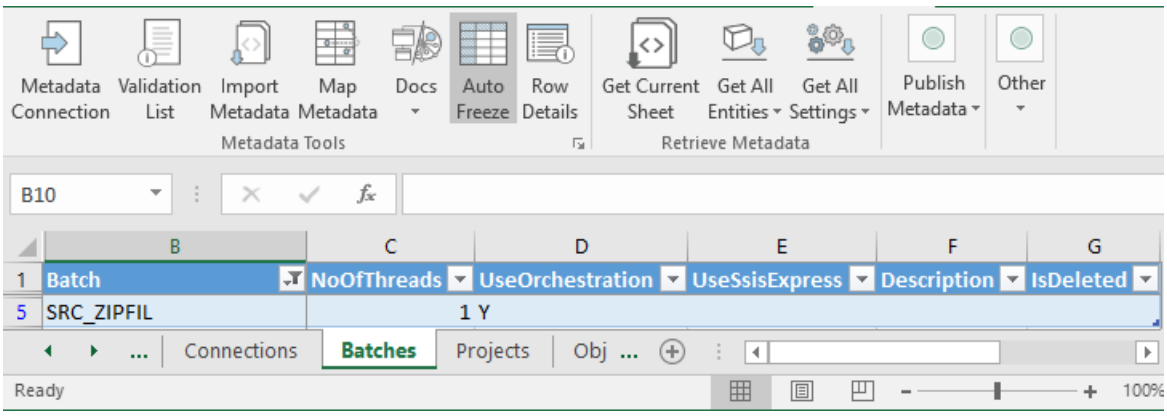
Connections

Assuming there already exist various layers in the data warehouse to extract from, the first thing to do is to create a new flat file connection in BimlFlex Excel. Some example connection attributes are shown below:

ATTRIBUTE NAME	EXAMPLE PROJECT ATTRIBUTES
Name	TransactionsFile
ConnectionString	C:\ExportFiles\Compressed
Catalog	Transactions
ConnectionType	FILE
SystemType	"File Delimited"\"File Ragged Right"\"Excel"
IntegrationStage	Staging
RecordSource	TF
FilePath	C:\ExportFiles\ Compressed\
FilePattern	Transactions*.*
NoOfThreads	0

Batches

The second metadata entity that needs to be defined is the Batch entity. The Batch will be the parent package the executes any child packages that actually export data. So, in this case, a new batch with the appropriate name. Select the number of threads to load with and whether or not to use orchestration.



Projects

The next step is to add a project. Below is a screenshot of an example project. It will generally follow the usual source to target metadata. In this case, ensure that `Source -> Zip File Extract` is chosen for the integration template. This will indicate to BimlFlex that there are extra requirements around loading to a file and compressing it etc.

Project	SourceConnection	StageConnection	TargetConnection
EXT_ZIP_Transactions	TransactionsZipFile	DWH_SRC	TransactionsZipFile

Batch	IntegrationTemplate	Description	IsDeleted
SRC_ZIPFIL	Source -> Zip File Extract		

Objects and Columns

Both the source definition and target definition are required in the objects metadata spreadsheet tab. Below is the definition of a sample target object in BimlFlex Excel.

Project	Connection	Schema	ObjectName	IsColumnNamesInFirstDataRow
EXT_ZIP_Transactions	TransactionsZipFile	tranzip	Transactions	Y

FlatFileType	DataRowsToSkip	TextQualifer	RowDelimiter	ColumnDelimiter
Delimited	0		CRLF	Comma

LastColumnDelimiter	CodePage	IsUnicode
CRLF	1252	Y

To finish implementing this pattern, the column definitions for the destination are required. In this case, the Transactions table. They need to be pointed to the destination file.

This is done by filling in the target object and target object columns from the columns that define the source table, in this case, the Transactions source table.

TargetTable	TargetColumnName
TransactionsZipFile.tranzip.Transactions	CustomerTransactionCode
TransactionsZipFile.tranzip.Transactions	CustomerCashCode
TransactionsZipFile.tranzip.Transactions	CustomerKey
TransactionsZipFile.tranzip.Transactions	RelatedCode
TransactionsZipFile.tranzip.Transactions	CustomerTransactionTypeKey
TransactionsZipFile.tranzip.Transactions	TransactionCreditAmount
TransactionsZipFile.tranzip.Transactions	TransactionDebitAmount
TransactionsZipFile.tranzip.Transactions	CustomerCreditAmount
TransactionsZipFile.tranzip.Transactions	CustomerDebitAmount
TransactionsZipFile.tranzip.Transactions	CustomerExchangeRate
TransactionsZipFile.tranzip.Transactions	PartClearedAmount
TransactionsZipFile.tranzip.Transactions	PartClearedBy
TransactionsZipFile.tranzip.Transactions	PartClearedDate
TransactionsZipFile.tranzip.Transactions	FeeAmount
TransactionsZipFile.tranzip.Transactions	FeeCurrencyKey

Finally, refresh the BimlFlex metadata in BimlStudio, build and test.

Requirements

7-zip must be installed on the server that is intended to host the SSIS packages. This is the current compression tool that BimlFlex uses as part of its standard source to zip file template.

Microsoft SQL Server Master Data Services (MDS) integration

TODO: To Complete

Microsoft SQL Server Master Data Services (MDS) is an enterprise data management tool included in the Enterprise Edition of SQL Server.

MSDN has an introduction to MDS here: https://msdn.microsoft.com/en-us/library/ff487003.aspx

Models

MDS uses models to maintain master entities in a SQL database. The models contain entities and these entities can have relationships. For administrators and Data Stewards there are two main ways to interact with the data

- Web based user interface
- Plugin based Excel user interface

The web based UI also provides management of the models and system.

The Excel plugin works similar to the BimlFlex plugin. The steward connects to a model and entity and gets data into Excel, manipulates it and publishes it back It is possible to stage data into MDS by loading source data into staging and persistent staging tables and exposing that data as the source for MDS. It is also possible to load directly from source to MDS. Once the MDS process is finished the data can be exposed through MDS export views and imported from MDS into the EDW through BimlFlex.

It is possible to architect the MDS management process in several different ways. This document demonstrates one way of loading data into MDS and one way of loading data from MDS export views into Staging tables for the data warehouse. Adjusting the patterns used can provide options to support other workflows and processes. Considering the risk for naming confusion as data is loaded from staging to staging and into MDS and into EDW it is recommended to explicitly name processes to indicate the full flow if data. “Import into MDS from source” and "import into EDW from MDS" both convey more information than "MDS Import".

Sample Model

This document uses the Model and Model Category part of the AdventureWorks LT 2012 database as the source of the MDS model data. A reporting workflow requires the data and relationship to be loaded in to MDS. Data Stewards manually manipulate attributes and relationships in MDS for further use in the organisation, including in the EDW. MDS for SQL Server includes its own sample models, including a model for AdventureWorks Products. It is possible to ruse the approach from this document to integrate into the sample models. This document uses a separate model with 2 separate entities The sample model is configured and set up as follows:

Log in through the web interface as an MDS administrator to create a model called ProductDemo 2 entities are needed for the new model:

- Product
- Category

For the optional staging tables name this demo uses a ModelName_EntityName naming convention to manage names. For larger implementations with multiple models and entities this approach will make it easy to know what table goes to what model/entity when staging. Our Product and Category entities have the following staging tables created:

- Stg.ProductDemo_Product_Leaf
- Stg.ProductDemo_Category_Leaf

Each entity need the following attributes created:

Product

- Color
- Category

Category

- Parent Category

The name and code attributes created by default in MDS is reused for the Name and code/key fields in the source entities.

The Product, Category and Category, Parent Category entities are entity based relationships to the Category Entity

Once the Model and the 2 entities are available the staging tables can be reviewed in SQL Server Management Studio. The default naming convention for the puts the model entity staging tables in the MasterDataServices database, in schema stg and uses the defined names with _Leaf suffixed.

Required metadata for import into MDS

The sourcing from somewhere into the MDS staging tables require metadata for both the source and the destination. The BimlFlex metadata import function can import the table and column definitions so that the process can be mapped. This document uses separate views for the staging tables for the product and category tables as the source. The loading of source data from the AdventureWorks LT source into Staging and Persistent Staging is considered a separate process. For a Business Data Vault implementation, the source could be a view created from the existing Raw Data Vault that exposes the required information needed in MDS. To make the source to staging and MDS processes separate and independent the MDS source views can read from staging or persistent staging. This will allow the sourcing process to be run independently and the MDS import to be complete without taking any delta loads from source into account. The metadata setup assumes the source to staging load of the AdventureWorks is completed and maintained in a separate project. The source of Product and Category data will be custom views created in the staging database. These views can rename, align datatypes and provide the right granularity for the MDS load. The sources are created in their own schema:

```
CREATE SCHEMA [MDS_SRC];
GO
```

The Product source view

```
CREATE VIEW [MDS_SRC].[Product]
AS
  SELECT DISTINCT
    CONVERT(NVARCHAR(250), p.Name) AS [Name] ,
    CONVERT(NVARCHAR(250), p.ProductNumber) AS [Code] ,
    CONVERT(NVARCHAR(100), p.Color) AS [Color] ,
    CONVERT(NVARCHAR(100), pc.Name) AS [Category]
  FROM [BFX_ODS].AWLT.Product p
  LEFT JOIN BFX_ODS.AWLT.ProductCategory pc ON p.ProductCategoryID = pc.ProductCategoryID;
GO
```

The Product Category source view

```
CREATE VIEW [MDS_SRC].[Category]
AS
  SELECT DISTINCT
    CONVERT(NVARCHAR(250), pc.Name) AS [Name] ,
    CONVERT(NVARCHAR(250), pc.Name) AS [Code] ,
    CONVERT(NVARCHAR(250), ppc.Name) AS [Parent Category]
  FROM [BFX_ODS].AWLT.ProductCategory pc
  LEFT JOIN BFX_ODS.AWLT.ProductCategory ppc ON pc.ParentProductCategoryID = ppc.ProductCategoryID;
```

(Note that the views will need some update work to take changes and effectiveness into account in a real project)

Connections

The existing `BFX_STG` staging connection can be reused for the MDS source views. A new connection to the MDS staging destination tables is added as an entry in the Connections Tab.

The new connection is called MasterDataServices and maps to the MasterDataServices database that the MDS instance has been configured to use. The new MDS destination connection uses the IntegrationStage attribute “Master Data Services” to indicate the use to BimlFlex.

The load from source staging table to MDS staging table is defined in its own batch.

The batch is included in its own project

The object tab will include the source tables/views used for loading into MDS and the destination staging tables in MDS.

For all objects the full set of column metadata will have been imported into the Columns sheet. Some columns are not required from the MDS tables. They can be ignored or removed in the metadata sheet. Depending on the source structure, not all columns will be mapped from source to MDS. This document uses prepared views that are aligned with the MDS destination. In an implementation project the master data modelling sessions would form the base for model design and attribute mappings. The entity load from source to MDS can also use different approaches to updates/changes to existing attributes. In most implementation scenarios, there will be some cases where an entity (identified by the code used) will only be loaded from source once. Other entities will be reloaded and updated with new attributes from source. It is important to decide how this process will be implemented as both the data management process and the technical implementation will be different depending on the requirements. In general, if the attributes are manually maintained in MDS there is no point in updating them from source ever again. Only the attributes maintained in source should be reloaded and updated. The source to target mappings of attributes from the source table to the destination MDS staging table is done in the columns tab in the Excel metadata editor. For the product entity, the name column is mapped to the name MDS standard column, the product number is mapped to the code (in this case the product number has been identified as the Business Key to be used for managing the products in MDS). The color column is mapped to the color destination staging column. The Category reference is to the code value from the Category entity. As the Code will be the name the source needs a join from the Product to the ProductCategory table to include the category name instead of the id number that is the technical key form the source. For the Product Category, the name is mapped to the code column and name column. The Parent Category will reference the Code column (the name, as that has been identified as the business key for the Category). Since the source only has the technical key here it is also required to join the Category source with itself to get the parent category name. The AdventureWorks source enforces a unique category name meaning it is possible to use it for the code column. For sources where names can be repeated in the hierarchy another approach would be necessary. The joins to derive this extra data are added either to the Objects tab for the 2 source to staging tables in the sourcing project. This would enforce the joins in the source query and add the expanded data in the staging and persisted staging tables. That builds a dependency between the projects where a clean separation of concern might be better. The example

views will join the data in persistent staging instead. Another option is to enforce the joins in the source by exposing data through prepared views. This moves the dependency on proper modelling all the way to the source and would require additional considerations. Once the source data views are created and imported into the metadata the mapping between the source and destination can be done.

SSIS Generation

The source views are TODO

Considerations for pre/post extension points for truncating staging tables and applying business rules/model validation before exporting.

Data vault export views and validation success filtering (sample metadata. Point out that only validated rows are safe to export and work with)

Considerations for hierarchies in mds and export to DW/DV.

Introduction to Data Warehousing using Azure Data Warehouse

TODO: Coming Soon

Orchestration

BimlFlex includes an Orchestration engine that controls failure scenario management, auditing and logging of runs. All Orchestration data is stored in the BimlCatalog database.

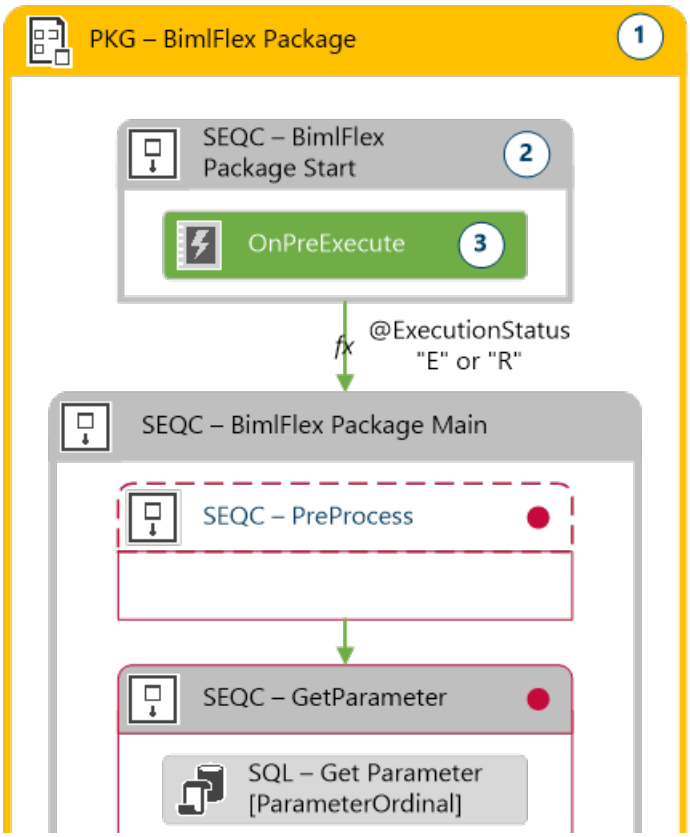
Batch Orchestration is controlled per Batch in the Batches metadata sheet.

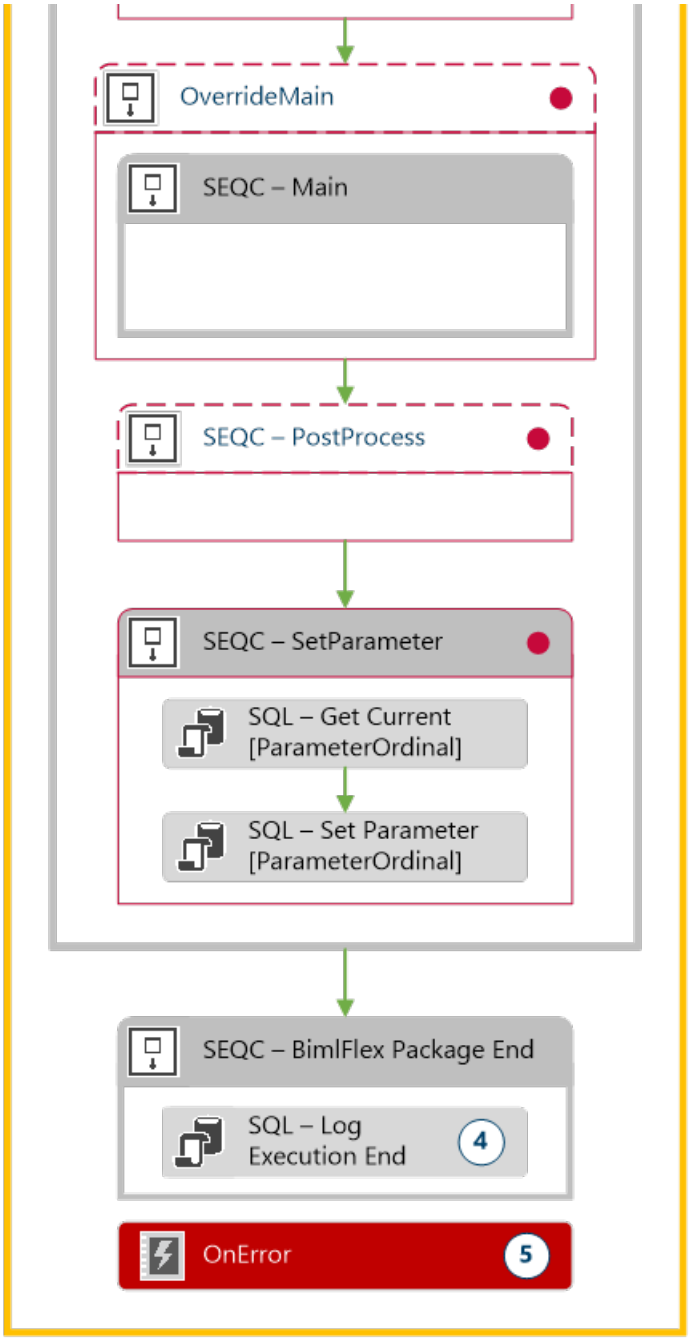
Legend

KEY	DESCRIPTION
	The red dot indicates a placeholder for extensions.
	The blue dashed line box with diagonal grey lines depicts "Overrides" derived from metadata as per the following example.
	The blue dashed line box without diagonal grey lines specifies "Overrides" derived from an ExtensionPoint defined within the BimlStudio project.
	The orange dashed line box with diagonal grey lines depicts a subprocess that is described in a separate diagram as per the following example.
	The red dashed line box depicts a control flow ExtensionPoint that is either injected in the relevant position or in some cases overrides the original BimlFlex process.

Orchestration

The following diagram illustrates the package control flow.





The Orchestration Framework enables packages to maintain process and data integrity in case of an error condition. If a Package or Batch fail the Orchestration manages what runs in the next sequence and can, optionally, handle rollback of inconsistent data.

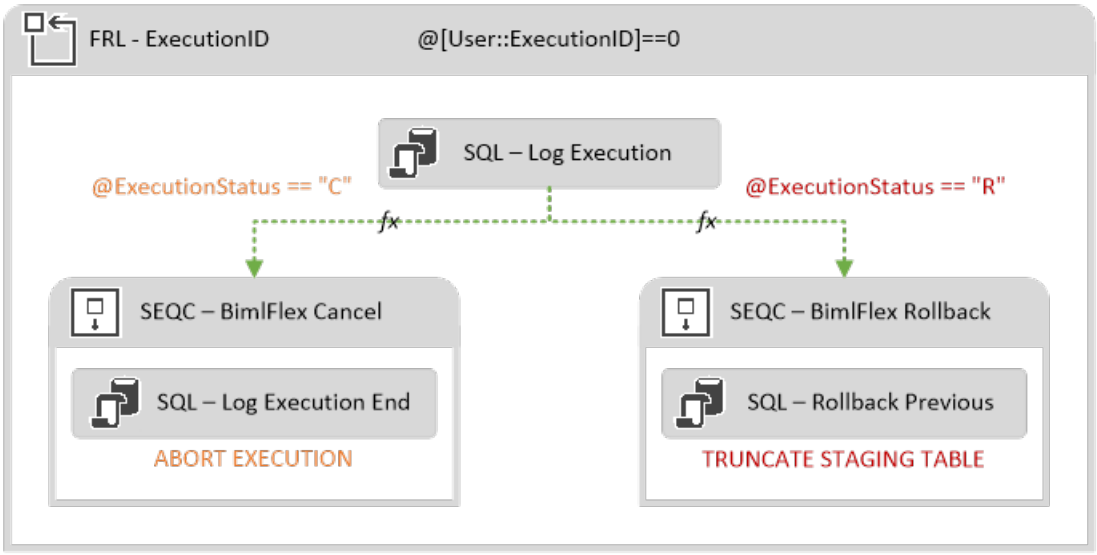
The Orchestration Framework includes audit and performance logging.

Package Segments

Every package is broken up into the following three main segments

SEQC – BimlFlex Package Start

The start sequence container holds the OnPreExecute event handler that controls the Orchestration path. The diagram below illustrates the event handler logic.



The following Stored Procedure is executed by the Start Event

```
EXEC [ssis].[LogExecution]
    @ExecutionGUID,
    @SourceGUID,
    @PackageName,
    @IsBatch,
    @ParentSourceGUID,
    @ParentExecutionID,
    @ServerExecutionID,
    @ExecutionID,
    @ExecutionStatus,
    @NextLoadStatus,
    @LastExecutionID,
    @BatchStartTime
```

SEQC – BimlFlex Package Main

Package Main is where all the template specific logic is implemented.

More information on this process:

- Source to Staging Templates
- Data Vault Templates
- Data Mart Templates

SEQC – BimlFlex Package End

Log successful execution of the package and set the next run status to 'P' for process. The following Stored Procedure is executed by the End Event

```
EXEC [ssis].[LogExecutionEnd] @ExecutionID, @IsBatch
```

Package Level Logging

Error Event Handler

Every package, including batches, has an OnError event handler configured. This is illustrated above as number 5 [OnError]

The Event Handler executes a stored procedure in the BimlCatalog database that logs the error

```
EXECUTE [ssis].[LogExecutionError]
   @ExecutionID
  ,@IsBatch
  ,@ErrorCode
  ,@ErrorDescription
```

The Orcestration passes the ExecutionID, a Boolean, that identifies if the package is a batch or a child package as well as the error code and description.

Execution Status

Two main status flags determine the process flow, ExecutionStatus and NextLoadStatus:

Execution Status Legend

CODE	DESCRIPTION
E	Executing. Only one instance of the package can be running. If another instance is started it will be aborted by [OnPreExecute]
A	Aborted. If the package is executed while already running it gets aborted by [OnPreExecute] and set the status to 'A'.
F	Failure. If there is a failure in the package or child packages the [OnError] Event Handler is invoked and the status is set to 'F'

Next Load Status Legend

CODE	DESCRIPTION
P	Process: The next time the package is executed it may start normally. 
C	Cancel/Abort: The next time the package is executed it should be aborted or skipped. 
R	Rollback: The next time the package is executed it should invoke the Rollback process if defined and execute. 

Start logic

Logs the start of execution. Checks the results of the previous execution before proceeding.

Rollback logic

Data and process integrity rollback for failed loads.

End logging

Logging successfully completion.

Error logging

Logs error condition and status as well as error code and description.

Scenarios

Orchestration behavior at individual package level and batch level

Package Level Success

If a single package makes it through the "SEQC - BimlFlex Package Main" container, the final container called "SEQC - BimlFlex Package End" will be invoked, this, in turn, executes the Execute SQL task "SQL - Log Execution End"

Where the logged execution id matches that of the successful package the ExecutionStatus will be set to `S` for success and the NextLoadStatus (the status that controls the actions of the next execution) is set to `P` for process.

This means that the package is ready for a normal run on the following execution and doesn't need to perform a rollback.

Batch Level Success

If all of the child packages in this batch successfully execute the result is similar to the success of an individual child package. After the main package sequence container, "SEQC - BimlFlex Package End" will be invoked, this, in turn, executes the Execute SQL task "SQL - Log Execution End", setting the `S` and `P` for the ExecutionStatus and NextLoadStatus

Package Level Failure

If a single package fails in the "SEQC - BimlFlex Package Main" container, meaning an error occurs, the OnError event handler will execute the execute SQL task called "SQL - Log Execution Error"

When the package is restarted the OnPreExecute event in the starting sequence container "SEQC - BimlFlex Package Start" will handle Orchestration. If rollback is enabled, the data from the previously failed execution of the package will be managed depending on what integration stage the package is used in.

Batch Level Failure

Batch level failures manages Orchestration and rollback for entire load across all packages in the batch.

Batch package failures trigger the OnError event handler that run the execute SQL task "SQL - Log Execution Error". This time the parameter `@IsBatch` will be true. Within `[ssis].[LogExecutionError]` there will be a conditional check made on `@IsBatch`, where if it is true we will update the execution table to make the NextLoadStatus to be `R` for all packages of the same ExecutionID, prompting all packages in that batch to rollback on their next execution.

Any configuration values associated with the failed ExecutionID are reverted to their previous values.

Regular statements for updating the ExecutionStatus and NextLoadStatus in `[ssis].[Execution]` to `F` and `R` respectively and adding a row to the `[ssis].[ExecutionError]` table.

Rollback

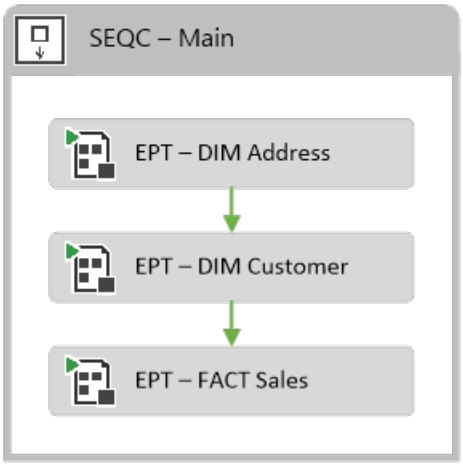
Rollback will remove partially loaded rows from the target tables. Depending on what integration stage the package are used in will determine what kind of roll back technique will be used.

- If the integration stage is Source the target table will be truncated.
- If the integration stage is not in the data warehouse, an ExecuteSQL task with a `DELETE` is executed on all records in the target table where the LastExecutionID is greater than the latest successful batch execution.
- If the integration stage is Source and the data is being persisted in a PSA environment, an ExecuteSQL task with a `DELETE` statement is executed on all records in the target table in Persisted Staging where the LastExecutionID is greater than the latest successful batch execution.
- If the integration stage is data vault (RDV/BDV) and the target tables are either a Satellite or Link Satellite, an update is executed on the table, setting the `RowEffectiveToDate` to `9999-12-31` and the `RowIsCurrent` to `true` where the `LastExecutionID` is greater than the latest successful batch execution.
- If the integration stage is in the data warehouse or data mart, the regular loading of data via hashed rows will detect the

data that needs to be updated and will be synced as normal operation.

Sample Execution Error workflow

The following illustrates the Orchestration of a main load sequence



An error condition in the second run will generate the following Status Flag updates for the run.

First Execution (Successful)

PROCESS	TASK	EXECUTION STATUS	NEXT EXECUTION
Start	PKG -> Batch	E	C
Start	PKG -> DIM Address	E	C
Success	PKG -> DIM Address	S	P
Start	PKG -> DIM Customer	E	C
Success	PKG -> DIM Customer	S	P
Start	PKG -> FACT Sales	E	C
Success	PKG -> FACT Sales	S	P
Success	PKG -> Batch	S	P

Second Execution (Failure)

PROCESS	TASK	EXECUTION STATUS	NEXT EXECUTION
Start	PKG -> Batch	E	C
Start	PKG -> DIM Address	E	C
Success	PKG -> DIM Address	S	P

PROCESS	TASK	EXECUTION STATUS	NEXT EXECUTION
Start	PKG -> DIM Customer	E	C
Failure	PKG -> DIM Customer	F	R
Skipped	PKG -> FACT Sales	S	P
Failure	PKG -> Batch	F	R
Success	PKG -> DIM Address	S	C

In the above scenario, when the Batch is executed again, it will skip `DIM Address` and Rollback `DIM Customer` if the rollback process has been enabled.

Rollback configuration

BimlFlex supports rollback for terminated or failed loads.

Process management for failed loads and data integrity checks is an architecture and process decision.

The rollback function is controlled per layer in the Configurations and all are disabled by default.

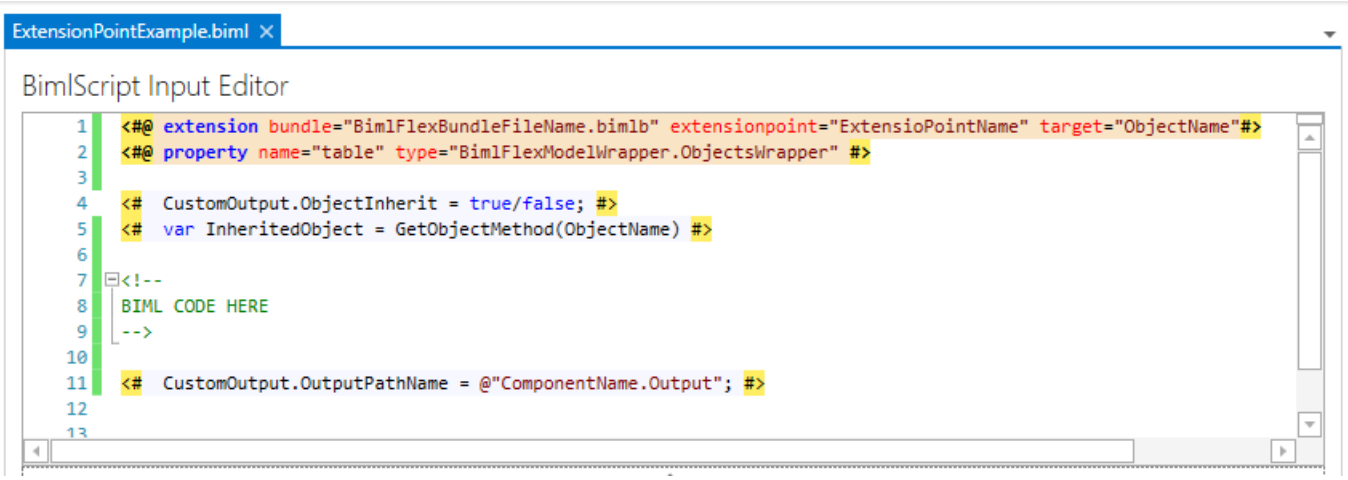
CONFIGURATIONKEY	CONFIGURATIONVALUE
EnableRollbackStg	N
EnableRollbackPsa	N
EnableRollbackRdv	N

Extension Points

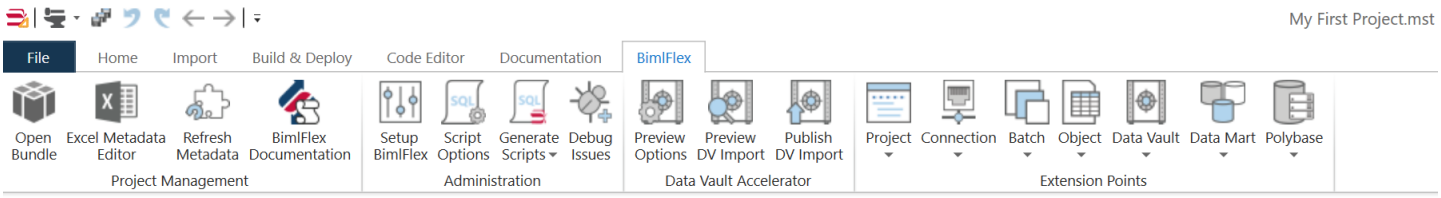
Extension Points are used to extend the default functionality of BimlFlex using standard Biml code. It can extend and override many different areas of the BimlFlex framework.

Extension Points have four key components:

- Extension Point directives that control what is injected and where
- Inheritance options code that defines any object inheritance
- Custom code that implements the required behaviour
- Input and output path variables for connecting the Extension Point in the SSIS package.



Extension Points are created in BimlStudio. In the BimlFlex Ribbon tab there are several Extension Point areas with a large number of different Extension Points available. Each Extension Point template will generate a code block that targets a specific point of the project.



Creating an Extension Point file and applying the required target attribute is all that is needed for it to be included into the project. when BimlFlex builds the solution any Extension Point code is injected into the resulting Packages.

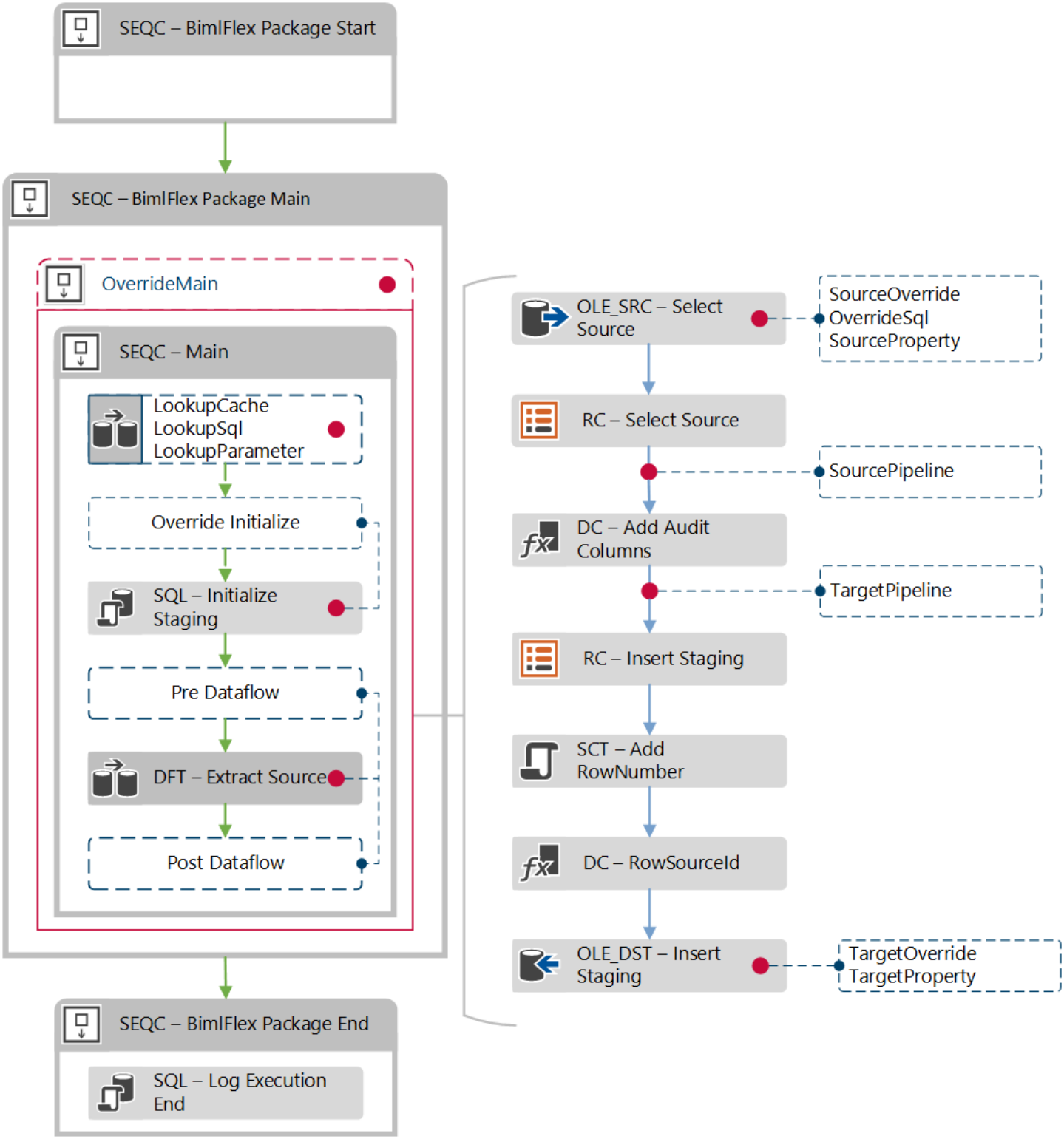
The Extension Points are saved as Biml files in the BimlStudio Project and should be treated as source code for the solution.

Default BimlFlex behaviour

Note that there are many features and functions built into BimlFlex that don't require Extension Points. Parameters, custom source queries with joins and filters, inherited tables and columns, data type expansions etc are all available without the need for Extension Points.

Extension Point Targets

Below is an example of where Extension Points can be added to the ETL structure. The red dots indicate injection points.



It is also possible to completely override the main container with an Extension Point. This can be used with pre-existing, bespoke packages. By importing the existing package into BimlStudio the Biml code version of the same package can be injected into the Extension Point.

Extension Point Directives

Special attention should be paid when editing the directives of an extension point because a user must have them correctly defined before they will be able to test the output of their Biml code. The purpose of a directive is to indicate to the compiler exactly what the purpose of the Biml code is that you are writing and where it should be injected. To write a directive simply insert the appropriate line of code between the following tags <#@ #>

The following table outlines the attributes of these directives.

EXTENSION POINT DIRECTIVE ATTRIBUTE	ATTRIBUTES DESCRIPTION
bundle	File name of the bundle being used
extensionpoint	Key work defining the type of extension point
target	The name of the object that the extension point will be applied to

The next directive is the property directive which is specific to the type of object you are trying to modify. Below is a table with the attribute definitions.

PROPERTY DIRECTIVE ATTRIBUTE	ATTRIBUTES DESCRIPTION
name	Name of the entity this extension point targets: "sourceTable", "targetTable", "connection" etc.
type	This is the entity type: "string", "BimlFlexModelWrapper.ObjectsWrapper", "BimlFlexModelWrapper.ConnectionsWrapper", etc.

Extension Point Inheritance Code

Here we will describe how a BimlFlex extension point can inherit the attributes and related items from an object that we are trying to modify. This is useful because, in some cases, properties relating to a certain object are required as part of what is needed to effectively do what we want to in an extension point.

Below are two of the required directives that need to be in place in order to use inheritance.

```
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper"  #>
<#@ import namespace="Varigence.Biml.Flex"  #>
```

A small block of BimlScript code will enable us to gain access to the items we need. A BimlScript is a piece of C# that is injected into a Biml file that gives us the ability to use Biml in a programmatic way. We will use C# methods to create variables that contain the objects and information we need.

First, we will need to write the first line that will allow us to obtain a copy of the targeted object. This is done by entering the following line.

```
<# CustomOutput.ObjectInherit = true; #>
```

Next, we will declare the variables we need to hold our information and use methods to obtain the objects of interest in this example. First, create a variable by using the `var` keyword and then giving the variable an appropriate name. In this example, we will get the source connection of a given table and add it as the value of the variable. Because we have already added the table as a property in our directive we can reference it directly.

```
<# var sourceConnection = EntityHelper.GetSourceConnection(table); #>
```

From here if we want to pull out any of the information about the object, we can insert it directly into our Biml code using the following tags: <#= #>. This will take the value stored in any variable, convert it to a string and insert in place of where the original BimlScript tags were.

For example:

```
<#=sourceConnection.Name#>
```

Output Path and Input Path Variables - Optional

Output path variables hold the name of the current task that you are modifying. The input path variable contains the previous task's name in the control flow or data flow so that the task you are working on can reference the previous item in your biml code and remain attached.

These two path variables are required when using extension points that either completely replaces tasks or when you are inserting a new task into an existing data flow or control flow.

If you are replacing or modifying a task at the end of a data flow or control flow then an output path variable is not required. In a similar fashion if you are replacing or modifying the first task in a control flow or data flow then an input path variable is not required.

We will see more examples of where we need to declare an output/input path variable later on.

```
<# CustomOutput.OutputPathName = @"TaskName/SequenceContainerName"; #>
```

Extension Points Examples

This section of the document covers the biggest variation of how extension points are used. However, there is a fairly exhaustive list of extension points available within BimlFlex. Here we will mainly be looking at demonstrating why a certain type of extension point will be used in a solution.

Project Based

Project based extension points are helpful, particularly because, depending on the type of solution being implemented, the Project at hand will require items that are not practical to maintain in metadata such as script files and bespoke project parameters. Generally, it is quite useful to have these items centralised within Biml code.

Project Parameter

Including parameters with extension points, allows greater reusability and logical separation in our solutions. Because of this, the project parameter extension point is useful for storing parameter definitions for high-level project parameters to do with databases and connections. This is important to note because, over the lifespan of a given data warehouse solution, there is a high chance of needing to repurpose the output of BimlFlex.

Extension Point Target

The extension point target, in this case, will be to point to the name of the project we are adding a parameter to.

Example/default Extension Point code generated from BimlStudio

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="ProjectParameter" #>

<Parameter Name="ServerName" DataType="String" IsRequired="true">localhost</Parameter>
<Parameter Name="UserName" DataType="String" IsRequired="true">varigence</Parameter>
<Parameter Name="UserPassword" DataType="String" IsRequired="true">P@ssw0rd!</Parameter>
```

Project Script File

As mentioned, one of the advantages of using extension points is to centralise code in Biml. This is relevant for script tasks as it is for any other complex ETL tasks.

For more information on how to implement a custom script task in Biml use the following resource.
https://www.varigence.com/Documentation/Language/Element/AstTaskScriptProjectNode

Extension Point Target

The target attribute of the extension point is the project that the script task belongs to.

Example/default Extension Point code generated from BimlStudio

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="ProjectScriptFile" #>
<#@ property name="project" type="BimlFlexModelWrapper.ProjectsWrapper" #>

<ScriptTaskProject ProjectCoreName="SC_SAMPLE_SCRIPT" Name="SC_SAMPLE_SCRIPT">
  <AssemblyReferences>
    <AssemblyReference AssemblyPath="Microsoft.SqlServer.ManagedDTS.dll" />
    <AssemblyReference AssemblyPath="Microsoft.SqlServer.ScriptTask.dll" />
    <AssemblyReference AssemblyPath="System.dll" />
    <AssemblyReference AssemblyPath="System.AddIn.dll" />
    <AssemblyReference AssemblyPath="System.Data.dll" />
    <AssemblyReference AssemblyPath="System.Windows.Forms.dll" />
    <AssemblyReference AssemblyPath="System.Xml.dll" />
  </AssemblyReferences>
  <Files>
    <File Path="ScriptMain.cs">
      using System;
      using System.Data;
      using Microsoft.SqlServer.Dts.Runtime;
      using System.Windows.Forms;

      [Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSISScriptTaskEntryPointAttribute]
      public partial class ScriptMain : Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
      {
        #region VSTA generated code
        enum ScriptResults
        {
          Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
          Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
        };
        #endregion

        public void Main()
        {
          MessageBox.Show("Sample Script File");
          Dts.TaskResult = (int)ScriptResults.Success;
        }
      }
    </File>
  </Files>
</ScriptTaskProject>
```

Connection Based

Connection based extension points are useful when adjustments are required to a connection, that is not part of the standard connection that can be implemented through the metadata connections in BimlFlex excel. There are a variety of scenarios where this may be required from Dynamic connection strings to Connections that are derived from custom components.

Connection Expression

Connection expressions are useful for creating dynamic connections, particularly when the connection string or source that we are connecting to is not known until runtime. For instance, a flat file connection may need to connect to a file which contains a date stamp in the name. Here just like in SSIS, we can enter in the expression we require for this functionality.

Extension Point Target

The target, in this case, is the name of a pre-existing connection that is in a solution.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="ConnectionExpression" target="ConnectionName" #>
<#@ property name="connection" type="BimlFlexModelWrapper.ConnectionsWrapper"  #>

<Expressions>
  <Expression ExternalProperty="ConnectionString">"Dsn=SRC_ODBC;Uid=" + @[$Project::UserName] + ";Pwd=" +
@[$Project::UserPassword] + ";"
  </Expression>
</Expressions>
```

Batch Based

Batch based extension points are generally only different from the other types of extension points in that their purpose for being implemented will be quite different than that say, of a package extension point or a project extension point. Batch extension points focus more on the processing of groups of files or data sources prior or post data load. We will see examples of this in the section to follow.

Parameter Bindings

Parameter bindings as in regular SSIS are used to gain access to a given parent parameter. In the same way, we can create these bindings in the following examples

Extension Point Target

The target will be the name of the batch that a user is trying to modify.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="ParameterBindings" #>
<#@ property name="batch" type="BimlFlexModelWrapper.BatchesWrapper" #>

<ParameterBindings>
  <ParameterBinding Name="SnapshotDate" VariableName="User.SnapshotDate" />
</ParameterBindings>
```

For further information on creating parameter bindings in Biml see the following resources.
https://www.varigence.com/Documentation/Language/Element/AstVariableParameterMappingNode

Batch Variable

Batch variables are fairly common, particularly if there are child packages that intend to inherit these variables. In terms of how the variables are defined these are effectively the same as any other package variable.

Extension Point Target

The target is the name of the batch package.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="BatchVariable"  #>
<#@ property name="batch" type="BimlFlexModelWrapper.BatchesWrapper"  #>

<Variable Name="CurrentModifiedDate" DataType="String" Namespace="User">1900-01-01</Variable>
```

For further resources on how to create variables in Biml, see the following resources
https://www.varigence.com/Documentation/Language/Element/AstVariableNode

Batch Package Configurations

This is another example where the differences in code between regular package configurations and batch configurations are minimal.

Below is an example of how to implement batch level package configurations.

Extension Point Target

The target is the name of the batch package

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="BatchPackageConfigurations"  #>
<#@ property name="batch" type="BimlFlexModelWrapper.BatchesWrapper"  #>

<# CustomOutput.ObjectInherit = true; #>
<PackageConfiguration Name="LOAD_MY_Configurations">
  <ExternalFileInput ExternalFilePath="C:\Varigence\Configurations\MY_BATCH_Configurations.dtsConfig"
FileUsageType="ExistingFile" RelativePath="false" />
</PackageConfiguration>
```

For more on how to create batch configurations in Biml see the following resources.
https://www.varigence.com/Documentation/Language/Element/AstPackageConfigurationNode

Batch Connection

Sometimes it is useful to add references in a batch package to a connection order to make a connection available to a given batch. This example will show how this can be implemented via extension point.

Extension Point Target

The extension point target is the name of the batch package we are trying to modify.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="BatchConnection"  #>
<#@ property name="batch" type="BimlFlexModelWrapper.BatchesWrapper"  #>

<Connection ConnectionName="MY_SOURCE" />
```

For more on how to create connection references in Biml see the following resources.

https://www.varigence.com/Documentation/Language/Element/AstConnectionReferenceNode

Batch PreProcess

Batch pre-process extension is used when there is a need to perform an overall process prior to executing a large set of packages. For instance, this can be to prepare a source data set or arrange a sequence of files before extraction. Either way, there is a wide variety of application for this particular extension point.

Extension Point Target

The extension point target is the name of the Batch.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="BatchPreProcess"  #>
<#@ property name="batch" type="BimlFlexModelWrapper.BatchesWrapper"  #>

<Container Name="SEQC - Get SnapShotDate Parameters" ConstraintMode="Parallel">
  <Tasks>
    <ExecuteSQL Name="SQL - Get SnapshotDate" ConnectionName="BimlCatalog" ResultSet="SingleRow">
      <Results>
        <Result Name="0" VariableName="User.SnapshotDate" />
      </Results>
      <DirectInput>EXEC [ssis].[GetConfigVariable] 'LOAD_DATAMART', 'LOAD_DATAMART.SnapshotDate',
'SnapshotDate', @VariableValue, @ExecutionID</DirectInput>
      <Parameters>
        <Parameter Name="@VariableValue" VariableName="User.SnapshotDate" Length="-1" DataType="String" />
        <Parameter Name="@ExecutionID" Direction="Input" DataType="Int64" VariableName="User.ExecutionID" />
      </Parameters>
    </ExecuteSQL>
  </Tasks>
</Container>
```

For more information on how to develop packages tasks in Biml see the following resources.

https://www.varigence.com/Documentation/Language/ChildCollection/AstContainerTaskBaseNode/Tasks

Batch Post Process

Similar to the pre-process extension point in BimlFlex, the post-process extension point is useful for large operations or tasks that require processing items after a load has completed. Examples of this could be moving process files, archiving processed flat files or uploading data to and external source after a load has completed. Again, a number of applications here are many and varied.

Extension Point Target

The extension point target is the name of the batch package.

Example Extension Point code

```
<#@ extension** bundle="BimlFlex.bimlb" extensionpoint="BatchPostProcess"  #>
<#@ property** name="batch" type="BimlFlexModelWrapper.BatchesWrapper"  #>

<Container Name="SEQC - Post Process" ConstraintMode="Parallel">
  <PrecedenceConstraints>
    <Inputs>
      <Input OutputPathName="SEQC - Main.Output" EvaluationOperation="Constraint" EvaluationValue="Success"/>
    </Inputs>
  </PrecedenceConstraints>
  <Tasks>
    <ExecuteSQL Name="SQL - Set SnapshotDate" ConnectionName="BimlCatalog" ResultSet="None">
    <DirectInput>EXEC [ssis].[SetConfigVariable] 'LOAD_DATAMART', 'LOAD\_DATAMART.SnapshotDate',
'SnapshotDate', @VariableValue</DirectInput>
    <Parameters>
      <Parameter Name="@VariableValue" VariableName="User.SnapshotDate" Length="-1" DataType="String" />
      </Parameters>
    </ExecuteSQL>
  </Tasks>
</Container>
```

For more information on how to develop SSIS tasks in Biml see the following resources.
https://www.varigence.com/Documentation/Language/ChildCollection/AstContainerTaskBaseNode/Tasks

Object Based

Object base extension points are generally used for making adjustments to packages that load only one table in its execution. This

particular segment of extension points contains the most variety in terms of what is available in extension points as this where most of the modifications are needed. Not only do we have the ability to modify what happens before and after a load, but we have the ability to adjust and add sequence containers for a particular purpose within the data flow itself.

Object Script File

The object script file extension point is mainly used because it allows us an easier way to centralise our script tasks in Biml code whilst not having to hold its entire definition with the package templates. This then can be referenced from the BimlFlex code that manages the generation of SSIS packages.

Extension Point Target

The target of this extension point is the name of the table object that is the source of this package.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="ProjectScriptFile" #>
<#@ property name="project" type="BimlFlexModelWrapper.ProjectsWrapper" #>

<ScriptTaskProject ProjectCoreName="SC_SAMPLE_SCRIPT" Name="SC_SAMPLE_SCRIPT">
  <AssemblyReferences>
    <AssemblyReference AssemblyPath="Microsoft.SqlServer.ManagedDTS.dll" />
    <AssemblyReference AssemblyPath="Microsoft.SqlServer.ScriptTask.dll" />
    <AssemblyReference AssemblyPath="System.dll" />
    <AssemblyReference AssemblyPath="System.AddIn.dll" />
    <AssemblyReference AssemblyPath="System.Data.dll" />
    <AssemblyReference AssemblyPath="System.Windows.Forms.dll" />
    <AssemblyReference AssemblyPath="System.Xml.dll" />
  </AssemblyReferences>
  <Files>
    <File Path="ScriptMain.cs">
      using System;
      using System.Data;
      using Microsoft.SqlServer.Dts.Runtime;
      using System.Windows.Forms;

      [Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSISScriptTaskEntryPointAttribute]
      public partial class ScriptMain : Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
      {
        #region VSTA generated code
        enum ScriptResults
        {
          Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
          Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
        };
        #endregion

        public void Main()
        {
          MessageBox.Show("Sample Script File");
          Dts.TaskResult = (int)ScriptResults.Success;
        }
      }
    </File>
  </Files>
</ScriptTaskProject>
```

For more information on how to define a script project in Biml, see the following resources.
https://www.varigence.com/Documentation/Language/ChildCollection/AstRootNode/ScriptProjects

Package Variable

This extension point will be used to define new variables in a given package on the object level. It should be noted that the way a user defines a package, in this case, is no different to how a batch package variable is defined, is it simply the way in which the extension point is targeted that is different.

Extension Point Target

The extension point target is the name of the source table object

Example Extension Point code

```
<#@ extension** bundle="BimlFlex.bimlb" extensionpoint="PackageVariable"  #>

<#@ property** name="table" type="BimlFlexModelWrapper.ObjectsWrapper"  #>

<Variable Name="TenantCode" DataType="String">UNK</Variable>
<Variable Name="CurrentModifiedDate" DataType="String" Namespace="User">1900-01-01</Variable>
```

For more information on how to implement package variables in Biml see the following resources.
https://www.varigence.com/Documentation/Language/Element/AstVariableNode

Package Parameter

Package parameters can be implemented through extension points also, this is generally done when there is a bespoke parameter required for a particular purpose. The parameters defined in packages are usually used in conjunction with other extension points that make use of these parameters later in the ETL pattern.

Extension Point Target

The target is the source object name.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="PackageParameter"  #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper"  #>

<Parameter Name="BatchId" DataType="String"></Parameter>
<Parameter Name="BatchInstanceId" DataType="String">0</Parameter>
```

For more on how to code these items in Biml please see the following resources.
https://www.varigence.com/Documentation/Language/Element/AstParameterNode

Override SQL

The Override SQL extension point is a more common extension point that is used to fully replace the select statement used in a source component. This is useful for scenarios where more complex source queries are required.

Extension Point Target

The target is the source object name.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="OverrideSql"  #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper"  #>

<# CustomOutput.ObjectInherit = false; #>
  SELECT DISTINCT
     [PropertyId]
    ,[EnterpriseId]
    ,CONVERT(INT, ISNULL([MemberId], - 1)) AS [MemberId]
  FROM
    [dbo].[MyTable]
```

For more on how to code these items in Biml please see the following resources.
https://www.varigence.com/Documentation/Language/Element/AstDirectResourceNode

Override Main

The Override Main extension point is used for completely replacing the main sequence container of an object package. The reason this can be useful to a user is in scenarios where it is easier to take a pre-existing package, convert this into Biml code through the BimlStudio import package tool and insert that new Biml code into the Override Main extension point. This way we can easily add an existing package to BimlFlex and also centralise the code for a given BI solution, which helps with maintainability.

Figure 9, Override Main Extension Point

Extension Point Target

The target is the source object name.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="OverrideMain" #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper" #>

<Tasks>
  <Dataflow Name="DFT - Load MySource" DelayValidation="true">
    <Transformations>
      <OleDbSource Name="OLE\_SRC - MySource" ConnectionName="MySourceConnection">
        <DirectInput>SELECT * FROM [dbo].[MySourceTable]</DirectInput>
      </OleDbSource>
      <OleDbDestination Name="OLE\_DST - MyTarget" ConnectionName="MyTargetConnection">
        <InputPath OutputPathName="OLE\_SRC - MySource.Output" />
        <ExternalTableOutput Table="\[dbo\].\[MySourceTable\]" />
      </OleDbDestination>
    </Transformations>
  </Dataflow>
</Tasks>
```

For more on how to code these items in Biml please see the following resources.
https://www.varigence.com/Documentation/Language/ChildCollection/AstContainerTaskBaseNode/Tasks

Pre Dataflow

As the name suggests this extension point is used for scenarios where a user needs to insert logic just before the data flow begins. In some scenarios when we use connection expressions or dynamic variables that change over multiple executions, it is useful to have the ability to add custom logic just before the data flow begins. Note that if this extension point is used, it will make it the first task in the control flow for the main sequence container as so the output pathname variable will need to be included.

Extension Point Target

The target is the name of the source object.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="PreDataFlow" #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper" #>

<#@ property name="precedenceConstraint" type="String"  #>
<# CustomOutput.ObjectInherit = false; #>

  <ExecuteSQL Name="SQL - Get CurrentModifiedDate" ConnectionName="BimlCatalog" ResultSet="SingleRow">

  <# if (!string.IsNullOrEmpty(precedenceConstraint)) {#>
    <PrecedenceConstraints>
      <Inputs>
        <Input OutputPathName="<#=precedenceConstraint #>" />
      </Inputs>
    </PrecedenceConstraints>
  <# } #>

  <Results>
    <Result Name="0" VariableName="User.CurrentModifiedDate" />
  </Results>

  <DirectInput>EXEC [ssis].[GetConfigVariable] 'MY_SRC', 'MY_SRC.dbo.Table.CurrentModifiedDate',
'CurrentModifiedDate', @VariableValue, @ExecutionID</DirectInput>

  <Parameters>
    <Parameter Name="@VariableValue" VariableName="User.CurrentModifiedDate" Length="-1" DataType="String" />
    <Parameter Name="@ExecutionID" Direction="Input" DataType="Int64" VariableName="User.ExecutionID" />
  </Parameters>

  <# CustomOutput.OutputPathName = @"SQL - Get CurrentModifiedDate.Output"; #>

</ExecuteSQL>
```

For more on how to code these items in Biml please see the following resources.
https://www.varigence.com/Documentation/Language/ChildCollection/AstContainerTaskBaseNode/Tasks

Post Dataflow

Similar to the Pre-Dataflow task, the post data flow tasks purpose is to add logic to the main sequence container just after the data flow task has ended. Again, if we are using connection expressions or variables that change after each execution it is helpful to be able to add additional logic at this location.

Extension Point Target

The target is the name of the source object.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="PostDataFlow" #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper" #>
<#@ property name="precedenceConstraint" type="String"  #>
<# CustomOutput.ObjectInherit = false; #>

<ExecuteSQL Name="SQL - Set CurrentModifiedDate" ConnectionName="BimlCatalog" ResultSet="None">
  <# if (!string.IsNullOrEmpty(precedenceConstraint)) {#>
    <PrecedenceConstraints>
      <Inputs>
        <Input OutputPathName="<#=precedenceConstraint #>" />
      </Inputs>
    </PrecedenceConstraints>
  <# } #>

  <DirectInput>EXEC [ssis].[SetConfigVariable] 'MY_SRC', 'MY_SRC.dbo.Table.CurrentModifiedDate',
'CurrentModifiedDate', @VariableValue, @ExecutionID</DirectInput>

  <Parameters>
    <Parameter Name="@VariableValue" VariableName="User.CurrentModifiedDate" Length="-1" DataType="String" />
    <Parameter Name="@ExecutionID" Direction="Input" DataType="Int64" VariableName="User.ExecutionID" />
  </Parameters>

</ExecuteSQL>
```

For more on how to code these items in Biml please see the following resources.
https://www.varigence.com/Documentation/Language/ChildCollection/AstContainerTaskBaseNode/Tasks

PreProcess

The pre-process extension point is located before the main container in the control flow. This is a fairly common extension point mainly because it is helpful to have logic that can be added where we may need to do some work in order to obtain the data that we want to load. This is the case in scenarios where we must download flat files or run some SQL code to help prepare the data we want to load. Note that because this extension point encloses the new logic inside a sequence container, input and output path variables are not required.

Extension Point Target

The target is the source object name.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="PreProcess" #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper" #>
<# CustomOutput.ObjectInherit = false; #>

<Container Name="SEQC - Get SnapShotDate Parameters" ConstraintMode="Parallel">
  <Tasks>
    <ExecuteSQL Name="SQL - Get SnapshotDate" ConnectionName="BimlCatalog" ResultSet="SingleRow">
      <Results>
        <Result Name="0" VariableName="User.SnapshotDate" />
      </Results>
      <DirectInput>EXEC [ssis].[GetConfigVariable] 'LOAD_DATAMART', 'LOAD_DATAMART.SnapshotDate',
'SnapshotDate', @VariableValue, @ExecutionID</DirectInput>
      <Parameters>
        <Parameter Name="@VariableValue" VariableName="User.SnapshotDate" Length="-1" DataType="String" />
        <Parameter Name="@ExecutionID" Direction="Input" DataType="Int64" VariableName="User.ExecutionID" />
      </Parameters>
    </ExecuteSQL>
  </Tasks>
</Container>
```

For more on how to code these items in Biml please see the following resources.

https://www.varigence.com/Documentation/Language/ChildCollection/AstContainerTaskBaseNode/Tasks

Post Process

The post process extension point is useful for custom logic that needs to be applied after the end of the main execution of a given package. Scenarios that require archiving or uploading of data to a different location are typical cases for when this extension point is used.

Note that because this extension point encloses the new logic inside a sequence container, input and output path variables are not required.

Extension Point Target

The target is the source object name.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="PostProcess" #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper" #>
<# CustomOutput.ObjectInherit = false; #>

<ExecuteSQL Name="SQL - Set SnapshotDate" ConnectionName="BimlCatalog" ResultSet="None">
  <DirectInput>EXEC [ssis].[SetConfigVariable] 'LOAD_DATAMART', 'LOAD_DATAMART.SnapshotDate', 'SnapshotDate',
@VariableValue</DirectInput>
  <Parameters>
    <Parameter Name="@VariableValue" VariableName="User.SnapshotDate" Length="-1" DataType="String" />
  </Parameters>
</ExecuteSQL>
```

For more on how to code these items in Biml please see the following resources.
https://www.varigence.com/Documentation/Language/ChildCollection/AstContainerTaskBaseNode/Tasks

Source Override

A source override extension point is similar to the override SQL extension point, but this is used when we want to completely replace the source component rather than adjust or replace the source query.

Remember that when the source component is changed in this fashion, consideration should be made to the columns and datatypes that are expected in the remainder of the data flow. The extension point assumes that the changes made using this extension point align with the rest of the data flow.

Extension Point Target

The target is the source object name.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="SourceOverride" #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper"  #>
<# CustomOutput.ObjectInherit = false; #>

<OleDbSource Name="OLE_SRC - MyTable" ConnectionName="MY_SRC" CommandTimeout="0"
ValidateExternalMetadata="false">
  <DirectInput>SELECT [Code], [Name] FROM [dbo].[MyTable]</DirectInput>
    <Columns>
      <Column SourceColumn="Name" TargetColumn="MyTableName" />
      <Column SourceColumn="Code" TargetColumn="MyTableCode" />
    </Columns>
  <# CustomOutput.OutputPathName = @"OLE_SRC - MyTable.Output"; #>
</OleDbSource>
```

The following is a resource on how to define an OLEDB Source in BimlCode.
https://varigence.com/Documentation/Language/Element/AstOleDbSourceNode

Source Parameter

A source parameter extension point is used for when we want to use a predefined parameter as a parameter to be mapped to our source component.

Extension Point Target

The target is the source object name.

Example Extension Point code

```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="SourceParameter" #>
<#@ property name="table" type="BimlFlexModelWrapper.ObjectsWrapper" #>
<# CustomOutput.ObjectInherit = false; #>

<Parameter Name="Parameter0" VariableName="User.LastModifiedDate" />
<Parameter Name="Parameter1" VariableName="User.CurrentModifiedDate" />
```

For information on Biml code used in this extension point please see the following resources.
https://www.varigence.com/Documentation/Language/Element/AstVariableParameterMappingNode

Source Pipeline

This extension point is used for inserting a data flow or multiple data flow tasks into the standard ETL data flow. Note here that the final data flow task that you define will need to have the name in the value of the OutputPathName variable. This extension point is mainly useful for adding custom data flow transformations to the existing data flow.

Extension Point Target

The target is the source object name.

Example/default Extension Point code generated from BimlStudio

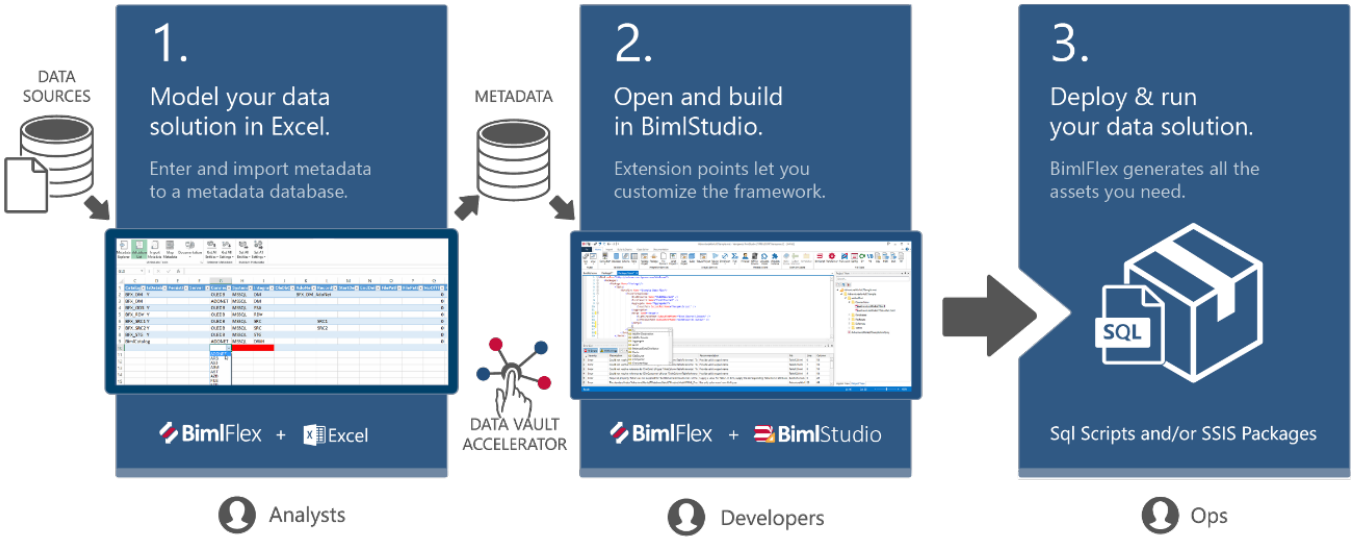
```
<#@ extension bundle="BimlFlex.bimlb" extensionpoint="SourcePipeline" #>
<#@ property name="sourceTable" type="BimlFlexModelWrapper.ObjectsWrapper" #>
<#@ property name="targetTable" type="BimlFlexModelWrapper.ObjectsWrapper" #>
<#@ property name="inputPath" type="String" #>

<DataConversion Name="DCV - Convert MyColumn">
  <Input OutputPathName="<#=inputPath #>" />
  <Columns>
    <Column SourceColumn="MyColumn" TargetColumn="cnv\_MyColumn" DataType="String" Length="100" />
  </Columns>
  <# CustomOutput.OutputPathName = @"DCV - Convert MyColumn.Output"; #>
</DataConversion>
```

For information on Biml code used in this extension point please see the following resources.
https://www.varigence.com/Documentation/Language/ChildCollection/AstDataflowTaskNode/Transformations

BimlFlex Excel Add-in

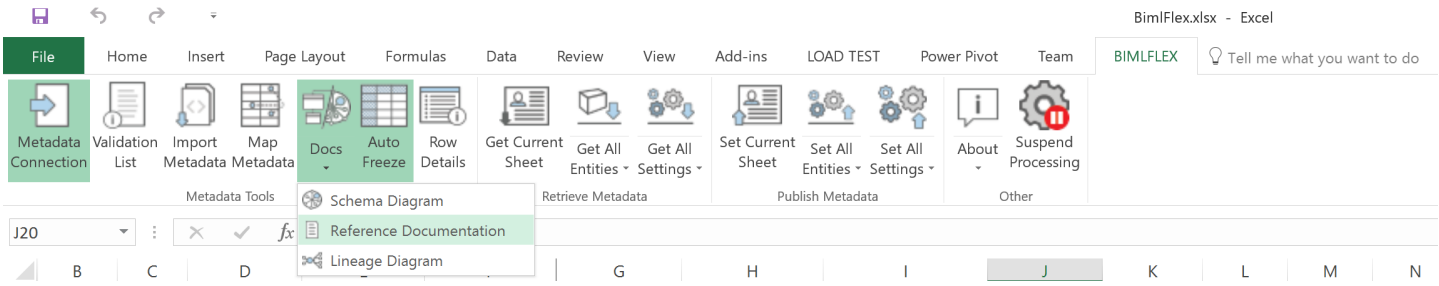
BimlFlex Excel Add-in is the tool modelers and developers use to interact with the solution metadata. BimlFlex Excel Add-in connects to the BimlFlex metadata database.



Getting Started

To install BimlFlex and the Excel Add-in refer to the installation and setup documentation

User Interface



The BimlFlex Excel ribbon and tab is located at the top right hand side of the standard Excel UI. This is where the UI for BimlFlex is located.

The BimlFlex Excel Add-in Ribbon tab is made up of 5 main groups:

- Metadata Tools
- Retrieve Metadata
- Publish Metadata
- Other
- Status (hidden when everything is ok)

Metadata Tools is as section containing tools and functionality that allows high level control over metadata. Importing metadata, connecting to various metadata sources or generating overview documentation of metadata.

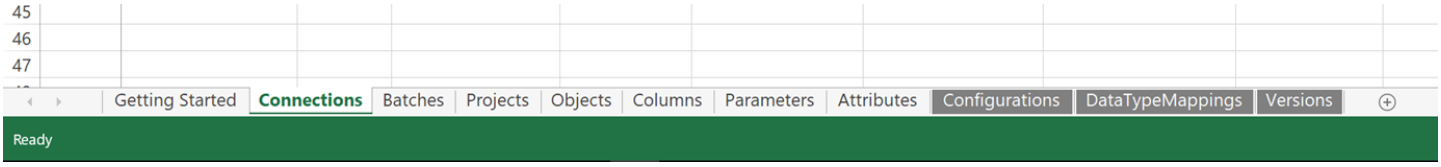
Retrieve Metadata is about getting the metadata from the metadata database (the BimlFlex Database) and bringing that metadata into Excel.

Publish Metadata allows Excel to save changed metadata back to the metadata database.

Other and Status sections apply to when BimlFlex Excel Add-in is inspecting the metadata. When an issue is encountered, the Status section will display any validation messages.

Excel Sheets

The BimlFlex Add-in displays the metadata in a set of Excel sheets. These are normal Excel sheets. To navigate between them, click the sheet tab at the bottom of the Excel window.



Getting Started Sheet

- Getting Started

Metadata Sheets

- Connections
- Batches
- Projects
- Objects
- Columns
- Parameters
- Attributes

Configuration Sheets

- Configurations
- Settings
- Data Type Mappings
- Versions

Metadata Entities

Each sheet contains columns and rows of metadata used to define entities and relationships. Managing the metadata and building the DataWarehouse is covered in the Implementation Guides and Configurations documentation.

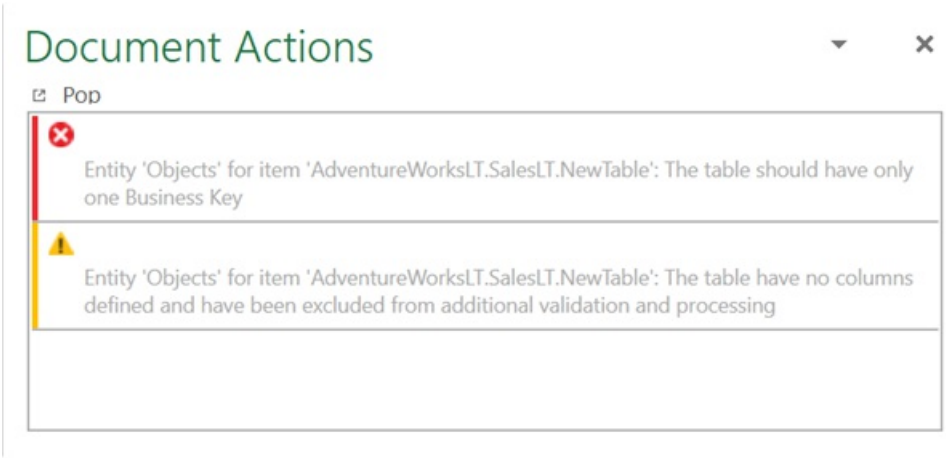
The Metadata specification is detailed in the Metadata Reference documentation

Metadata Connection

The metadata data connection button in BimlFlex Excel is located on the far left and is the starting point for connecting to a given metadata database. This button opens a pane in Excel for connecting to a metadata database.

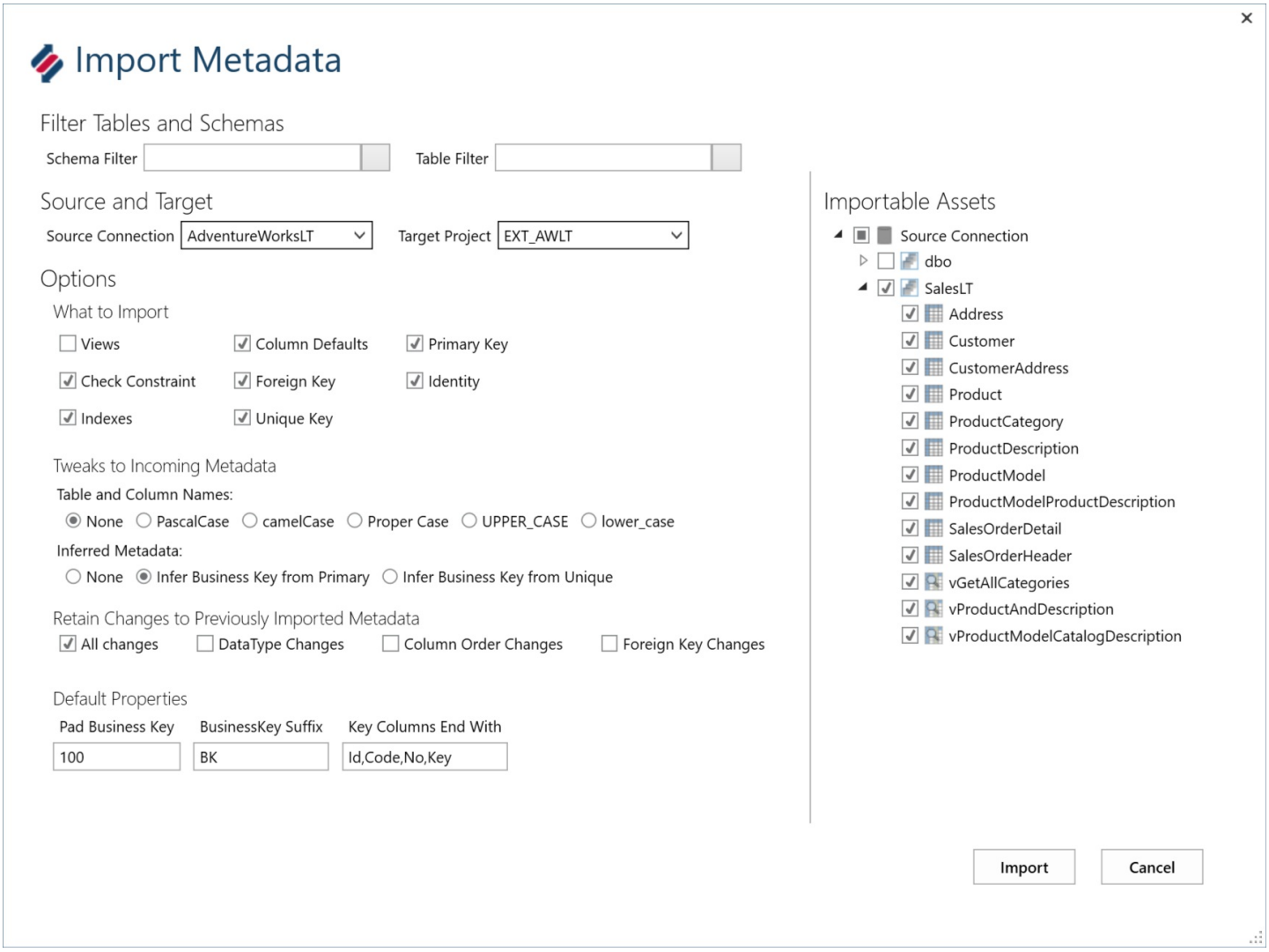
Validation List

The validation list is a pane in BimlFlex Excel Add-in where any issues with metadata are displayed in a list along the side of the screen. The issues shown on this list are generated by the validation rules in the BimlFlex bundle.



Import Metadata

BimlFlex Excel Add-in provides a metadata import function to add metadata into a project from a source that provides metadata information.



More information about the metadata import tool is found in the documentation

Auto Freeze

The Auto Freeze function freeze the left hand side and top of the sheet in Excel. By default the frozen area are the relevant columns of a metadata sheet.

Row Details

Get Current Sheet

Get current sheet will take the currently selected sheet in excel and retrieve the relevant metadata from the BimlFlex metadata database.

Get All Entities

Get all entities will go through each of the metadata spreadsheets and retrieve the current version of the metadata for each one.

Get All Settings

Get all settings will get all the

- Configurations
- Settings
- Data Type Mappings
- Versions

metadata for the current customer.

Set Current Sheet

Set Current Sheet will take the currently selected sheet in Excel and publish the relevant metadata changes to the BimlFlex metadata database.

Set All Entities

Set All Entities will go through each of the metadata spreadsheets and publish the current version of the metadata for each one.

Set All Settings

Set All Settings will publish the Configurations, Data Type Mappings and Versions metadata sheets to the metadata database

About

Clicking the About button will display to the user a drop-down box that shows two more options. The options are About and Help

Help will redirect the user to the Varigence support website, where they can access articles and forum information on the use of the various products in the Varigence suite.

About will show various product information that relates to the customer and some support options should a user experience issues with a particular version of BimlFlex Excel Add-in.

Here you can view

- product name
- version
- licensed user
- product key

The `Change Product Key` will allow changing the current product key.

The `Copy Product Information` copies product information to the clipboard. This allows a quick way to share information about the version of BimlFlex Excel Add-in if required.

The `Copy Machine Code` button will copy a code that is unique to the current machine. machine code is used to generate static product activation keys for machines not normally connected to the Internet.

Suspend Processing

The suspend processing button will pause BimlFlex Excel Add-in processing of the metadata through the Bundle. To start processing metadata, press this button again.

This function is useful in situations when larger quantities of metadata are changed and validation is applied later.

Object Inheritance

TODO: Coming Soon

Business Keys and Relationships

TODO: Coming Soon

SSIS Deployment Guide

Modern versions of SQL Server (2012-) supports project configurations and deployments to the SSIS Catalog. This document details how to use the project deployment model in BimlFlex and how to align that deployment to the SSIS Catalog environments for variable assignments. This can be used to populate passwords and connection strings for connections and inject them when the package is running on the SSIS Server

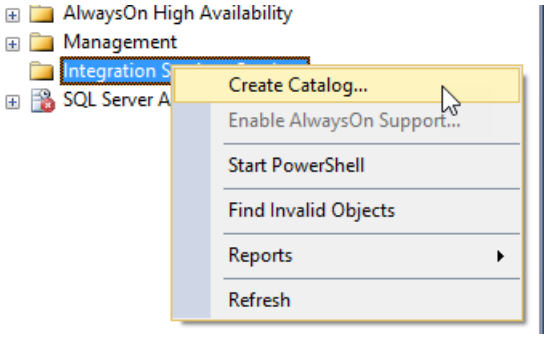
BimlFlex also supports the legacy package based deployment and its configuration approach.

It is a recommendation to:

- Use a modern, supported version of SQL Server
- Use the project configuration model
- Deploy the solutions to the SSIS Catalog
- Use Environments to configure connection strings and other parameters and configurations.

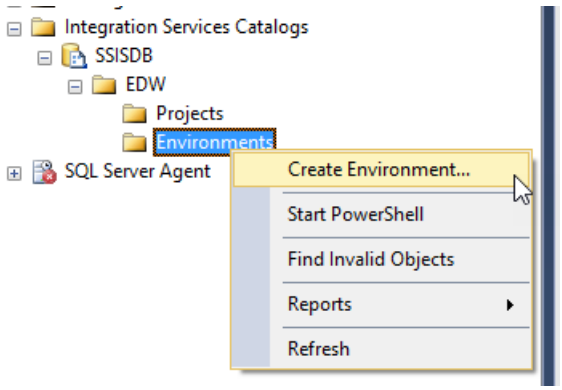
Validating the Catalog in the target server

A catalog needs to be created if the catalog node in Management Studio is empty for the target server. The mandated name for the Catalog and corresponding database is SSISDB.

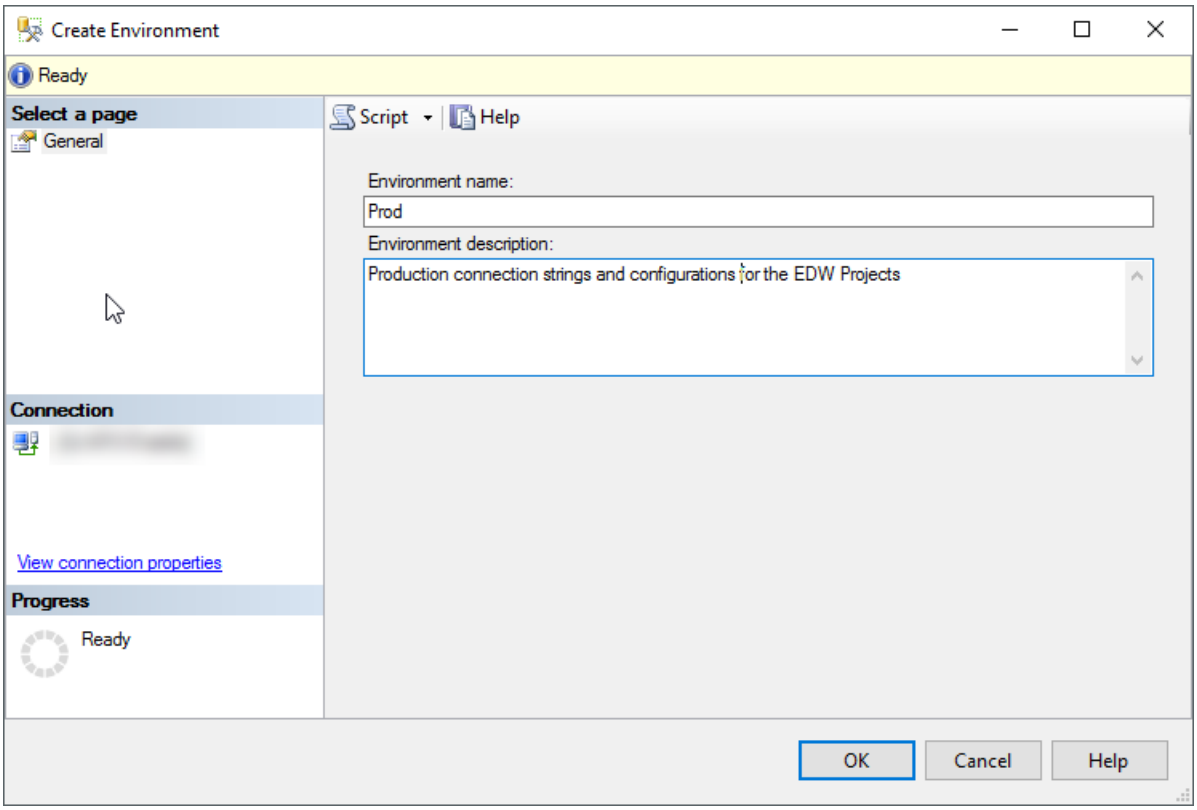


Once a catalog is available it is possible to create a folder for the projects. Multiple projects can be deployed to the same folder. The folder provides a common set of environments for all its projects.

In general, it makes sense to deploy all projects related to the Data Warehouse to the same folder so that they can share all connection string and other configuration information. It is possible to test deploy new versions to a separate folder if there is no separate test environment.



For each folder, it is possible to create one or more Environments. The environments can be for testing/production distinction for jobs or for different sets of configurations needed.



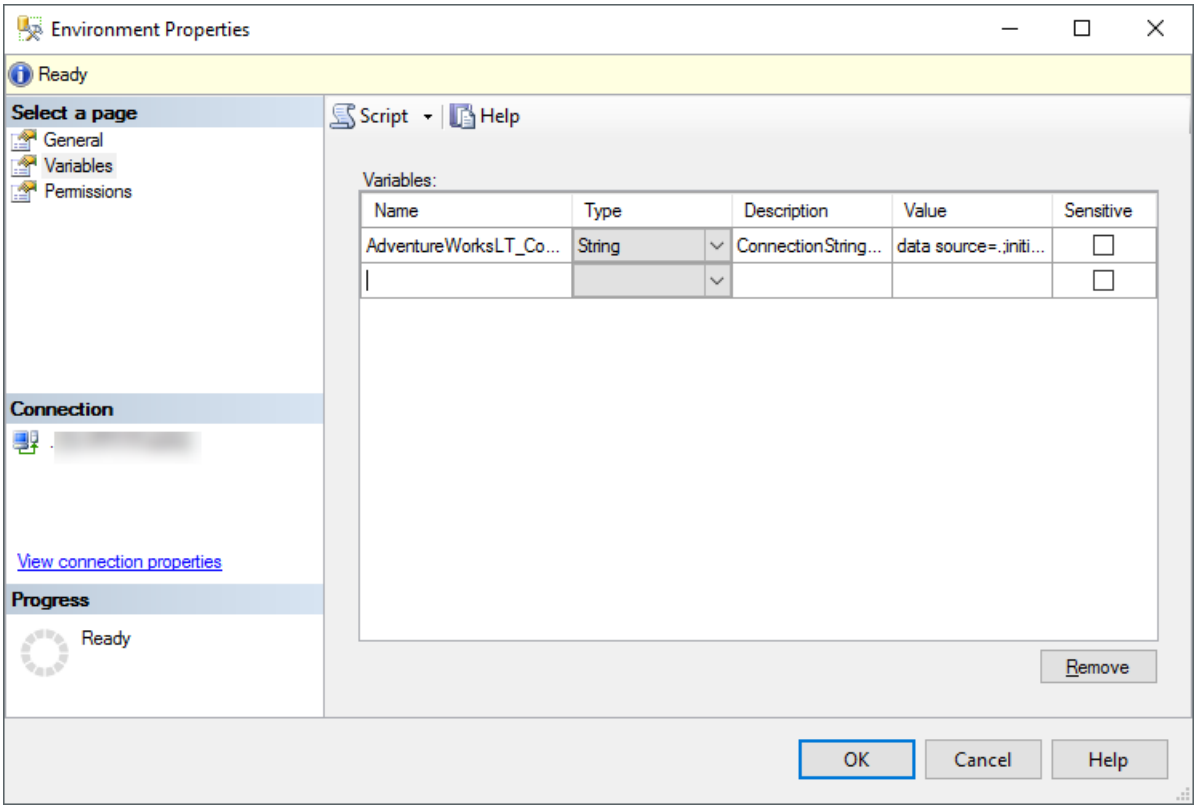
Variables can be defined in the environment once it is created. These normally correspond to the parameters and connection strings defined in the SSIS project. Retaining names across the locations makes it easy to know what is what. The default name for a connection string in the SSIS project, when defined as a parameterized connection manager is

`<connection name>_ConnectionString`

in this case, the default name for the source is

`AdventureWorksLT_ConnectionString`

By naming the environment variable the same it is easier to match them up later.



Once the variables are defined the SSIS Project needs to be deployed for the connection to be configured.

Custom SSIS Components used in BimlFlex

BimlFlex uses several custom SSIS components in the generated packages. These components are specific to the SQL server version used and need to align with both the SQL Server and the Project versions that are deployed. This is part of the installation and setup process and assumed installed on the destination server for the process described in this document.

If the project is deployed using the wrong Visual Studio SSIS Target Server/Version specification there will be errors when running these components.

Project deployment for SQL Server 2016 using the catalog

As a demonstration project, a new BimlStudio project called Demo located in the c:\Varigence\ folder will be used. The project will create a folder called Demo for the BimlStudio project contents.

This project will have a single integration to read from the AdventureWorks LT source to the Staging Database.

The approach assumes a SQL Server version that supports SSIS Catalog (2012-) and in the document SQL Server 2016 is used.

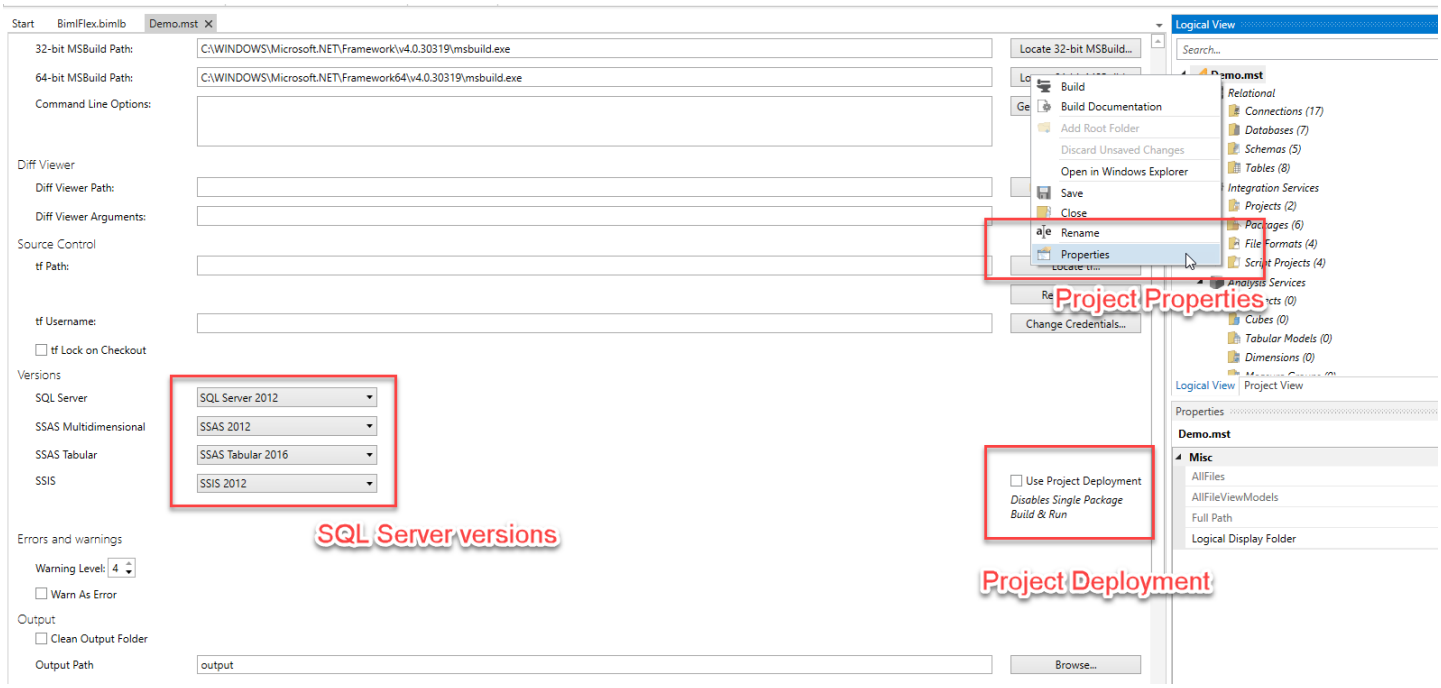
1. Before deployment, a Catalog must be available in the target SQL Server instance.
2. Once the required metadata is created BimlStudio will build the SSIS Project.
3. Once the build is successful it will be deployed to the SSIS Catalog.
4. Once deployed an environment will be used to override connection strings
5. Once configured SQL Server agent will be used to schedule a batch for a specific environment, using the environments connection strings.

Create and build a Demo project

Step 1, Create Project

From BimlStudio a project called Demo is created in the `C:\Varigence\Demo` folder.

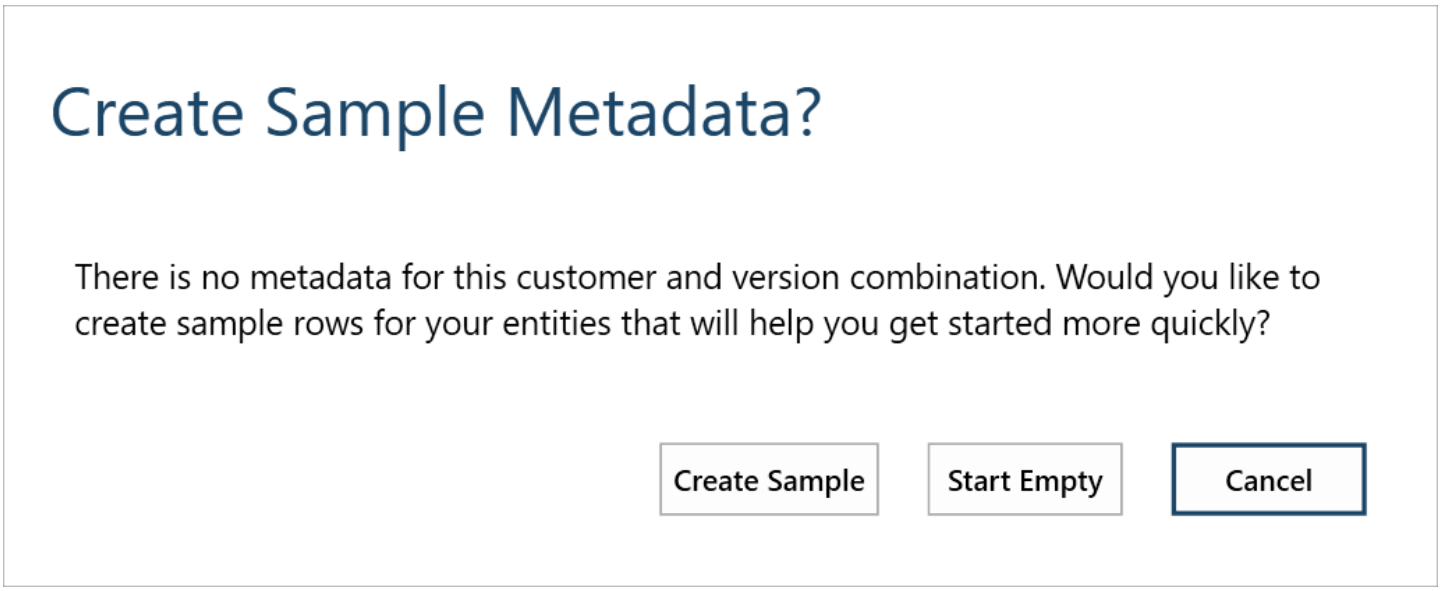
The project is configured to match the target destination



SSIS 2016 is used in the demo. The Project deployment checkbox is checked so connection strings for the whole project can be maintained through SSIS Catalog environments.

Step 2, metadata editor

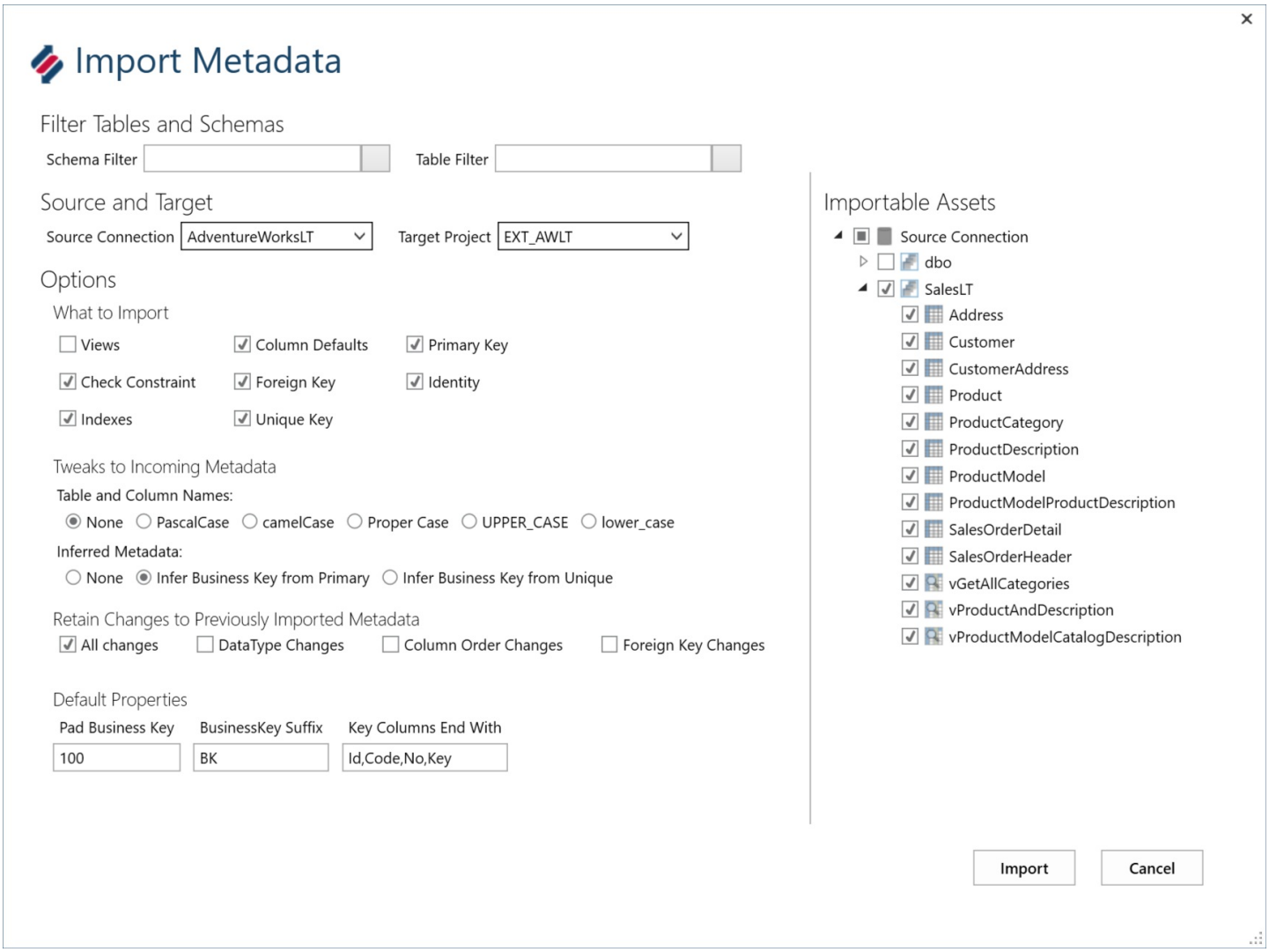
Every time a new empty customer is created in BimlFlex the Excel Metadata editor will ask if sample metadata should be created.



As the demo source is one of the sample sources it is possible to take a shortcut through the sample metadata. Once the sample metadata is created it can directly connect to the AdventureWorks LT Source to get the required source metadata.

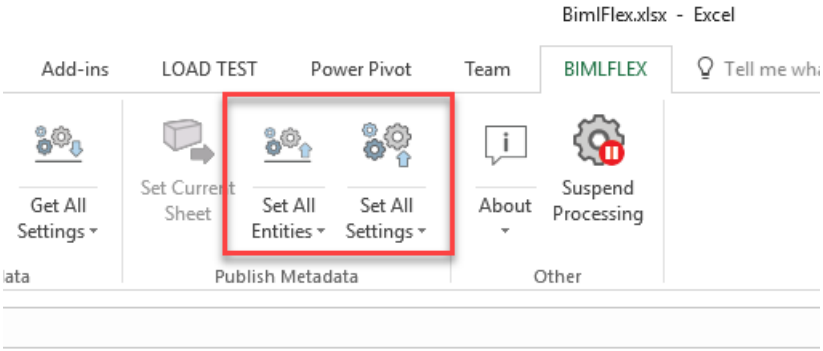
Step 3, load metadata

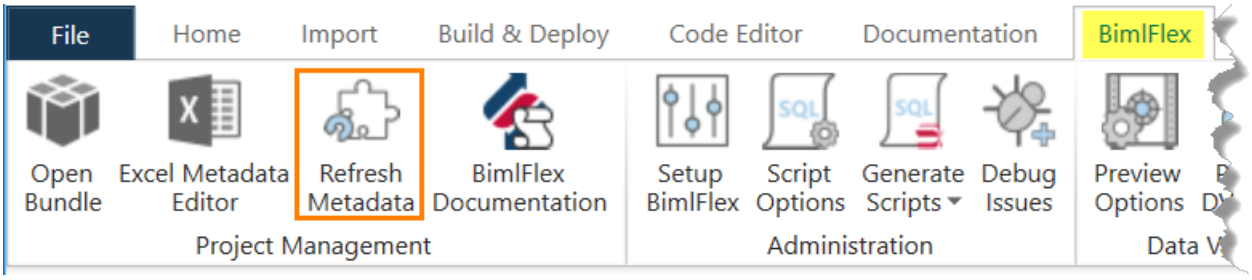
Load the metadata for all tables in scope from the AdventureWorksLT source system



Step 4, push metadata

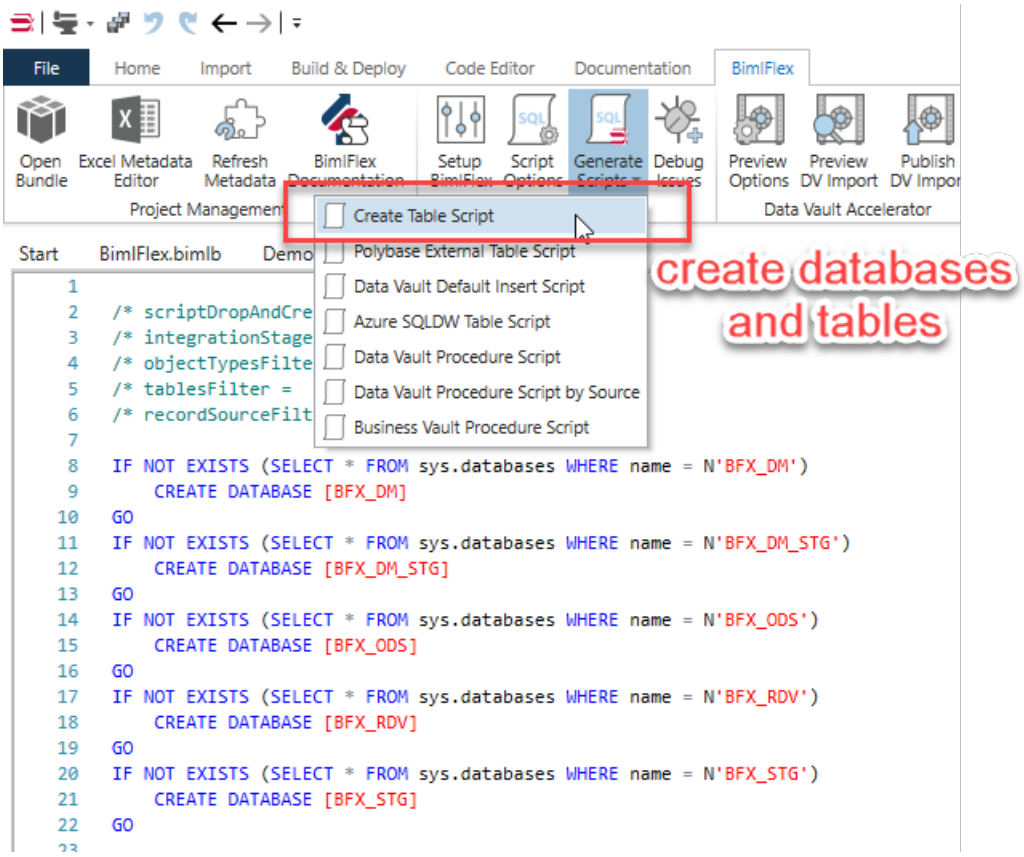
Push all metadata from Excel to the metadata database and refresh it into BimlStudio





Step 5, create databases and tables

Create all DW databases and tables from the BimlStudio generated scripts output.

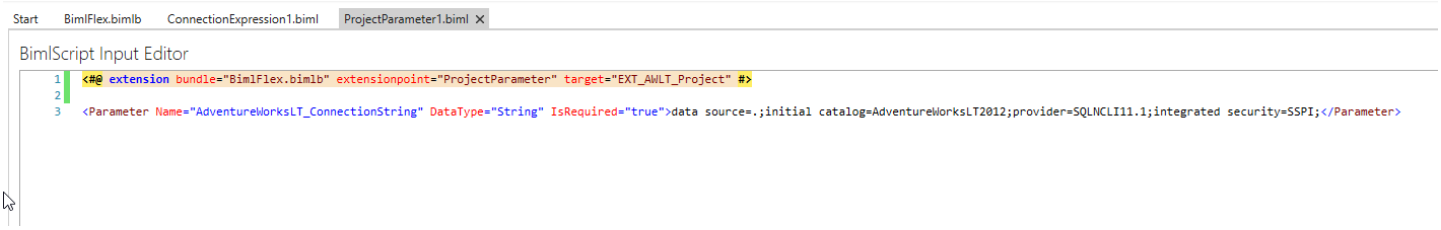


Step 6, define project parameters

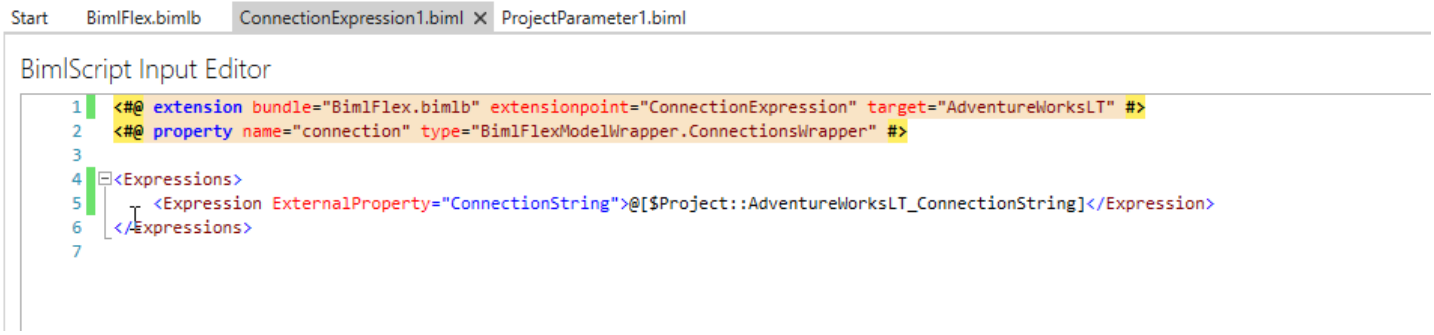
Define project parameters and connection expressions for all relevant connections in the BimlStudio solution.

There are many ways of maintaining the connection strings and required configurations in SSIS and SQL Server. BimlStudio provides

several easy ways to manage and maintain whatever is chosen for an environment. For this demo, deployment is done to the catalog and a Catalog environment variable is used to define the connection string. For the parameterised connection string to be available in the generated SSIS project it first needs to be defined through an Extension Point. The approach needs one Extension Point for the Project parameter and one for the connection string parameterisation.



The project parameter target is the SSIS project it will be used in. In this case, the `EXT_AWLT_Project` where project parameters will be used to define connection strings



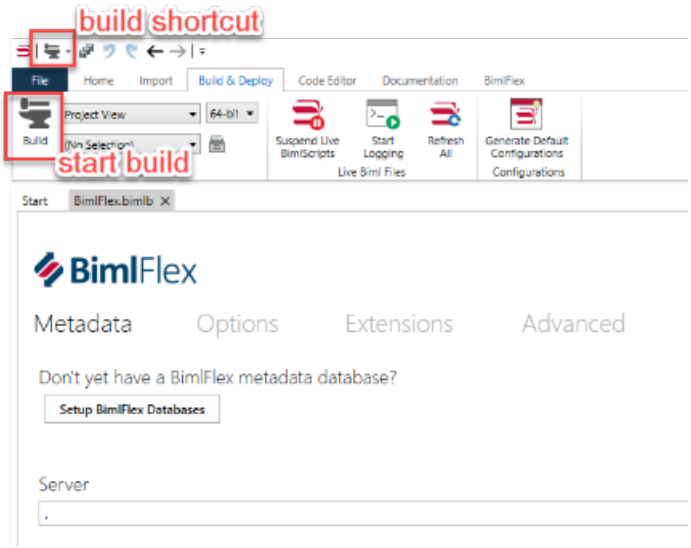
The connection expression connects the target property, in this case, the target connection manager `AdventureWorksLT` and the ConnectionString property with the defined Project parameter.

When the project is opened in Visual Studio the project connection manager can be verified as parameterized to use the project parameter.

Extension Point files should be named using a consistent naming convention. These files form part of the source code for the project itself and need to be maintained as such. The contents of the Extension Point files are not maintained in the metadata database.

Step 7, build the projects in BimlStudio

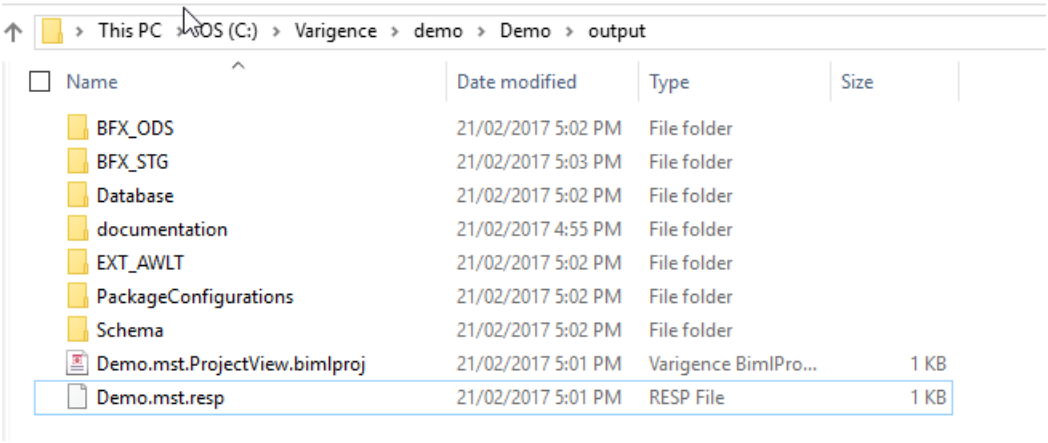
Build command buttons



Step 8, review the generated SSIS Projects

The default destination for the output is a folder called output in the project folder. In this case `C:\Varigence\Demo\Output\`

This location can be defined and changed in the project settings.



Once the build completes there will be a number of folders for the complete solution in the output folder. The folders in the screenshot above contain:

- BFX_ODS, an SSIS project to create all tables defined in the BFX_ODS database
- BFX_STG, an SSIS project to create all tables defined in the BFX_STG database
- Database, SSIS Projects to create all databases in the solution
- Documentation, generated documentation for the solution
- EXT_AWLT, the generated SSIS project to source the data from the AdventureWorks LT source to the Staging and Persistent Staging databases
- PackageConfigurations, dtsConfig files for projects that use PackageConfigurations
- Schema, SSIS projects to create all Schemas defined in the databases

The `EXT_AWLT` folder is named after the project name in the metadata and contains all packages defined in the project.

In the `EXT_AWLT` folder (and in all SSIS folders) are both the SSIS project with the individual dtsx files and a sub folder called bin that holds the compiled .ispac deployment file.

It is possible to deploy either the ispac file or the project to the catalog. Ispac files are great for automated deployment and command type approaches. Opening the project in Visual Studio allows manual, visual review, manual running of the process and manual deployment from Visual Studio to a destination Catalog.

Deploying the generated ispac file to the Catalog

The full deployment process is documented here:

- Deploy Integration Services (SSIS) Projects and Packages

for ispac files when there is a requirement to automate the deployment using command line tools in an automated fashion. This can be done either through a locally set up deployment pipe or through automation tools for server based automated deployments.

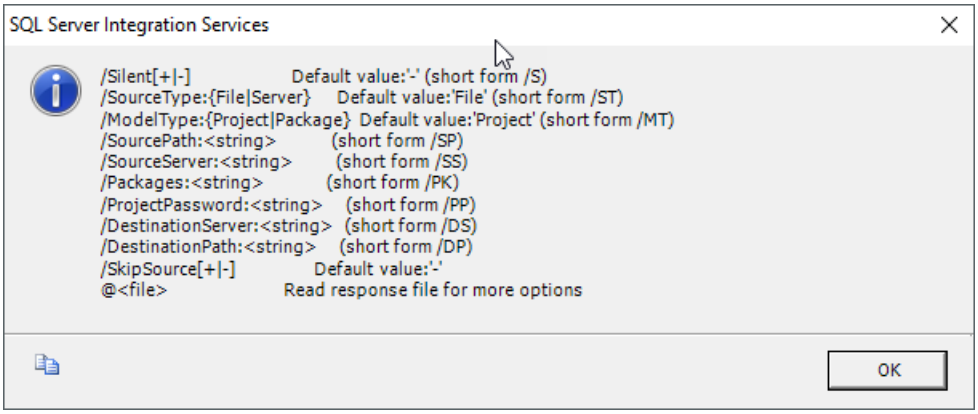
When deploying to the catalog the projects are deployed into folders. This means it is easy to organize projects. It also means different versions can be deployed to different folders if there is a need to maintain different sets of packages on the same server for testing.

There are 3 main ways of deploying to the Catalog:

1. Through the isdeploymentwizard.exe command line tool (using the ispac file)
2. Through Visual Studio (using the SSIS project)
3. Through Management Studio (using the ispac file)

Deploying through the command line

From the command prompt, run `isdeploymentwizard.exe` from the `%ProgramFiles%\Microsoft SQL Server\130\DTS\Binn`. Folder (Note the path is SQL Server version specific) with the required parameters for either interactive or silence deployments



For this example, to deploy the generated ispac quietly to the local server EDW folder run:

```
C:\Program Files\Microsoft SQL Server\130\DTS\Binn>isdeploymentwizard /S
/SP:"C:\Varigence\Demo\output\EXT_AWLT\bin\EXT_AWLT_Project.ispac" /DS:localhost /DP:"/SSISDB/EDW/Projects/"
```

The Visual Studio deploy wizard review dialog also provides the command line parameters it uses in case there is a need to mimic its behavior.

For automated deployments, this would normally be defined through a script file or a deployment tool.

Opening the project in Visual Studio 2015 and deploying to the Catalog

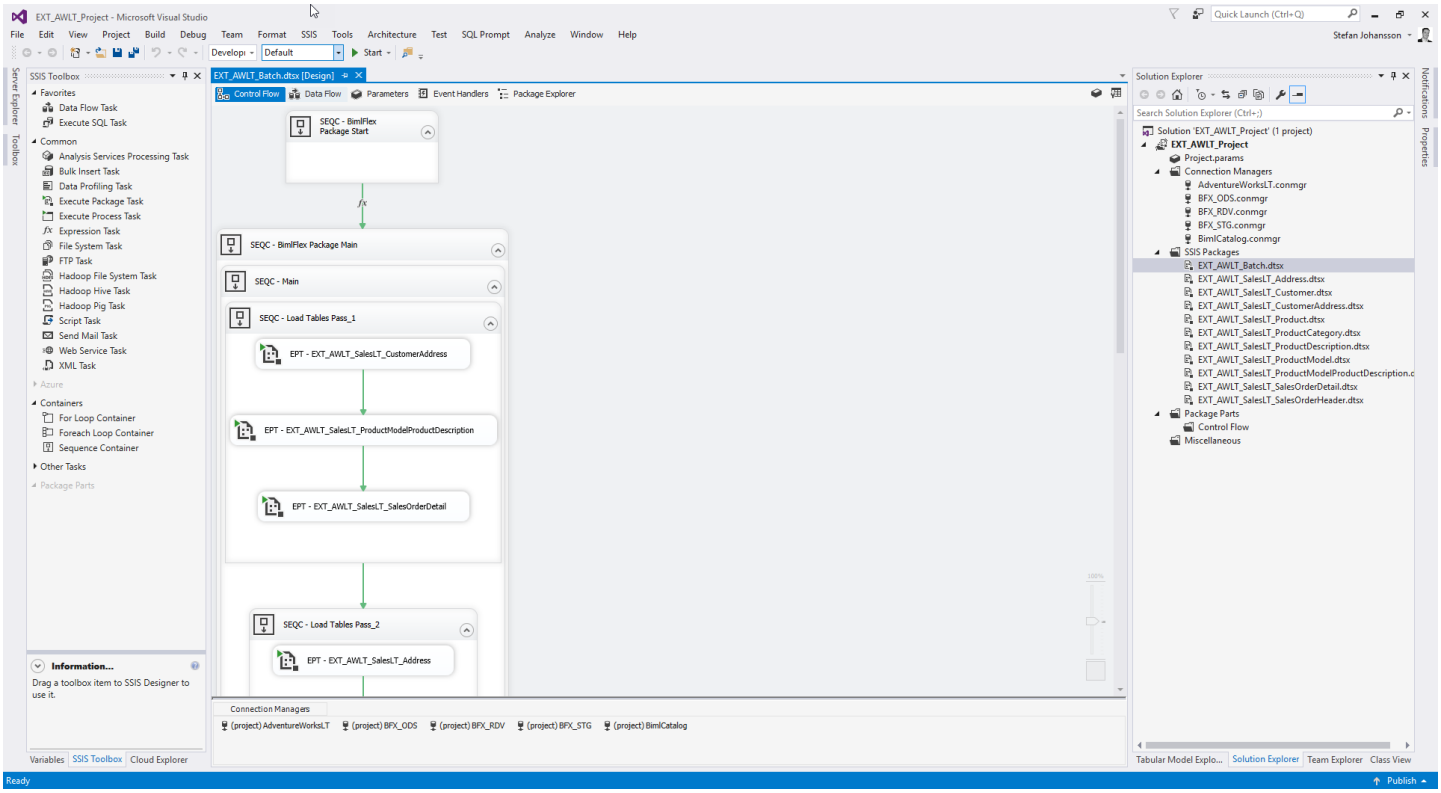
BimlStudio generates SSIS projects but not solutions. The default behavior for a project file (`.dtproj`) is not associated with Visual Studio SQL Server Data Tools for Business Intelligence so in a new environment it will need to be opened from Visual Studio or the extension needs to be associated to open with SSDT BI.

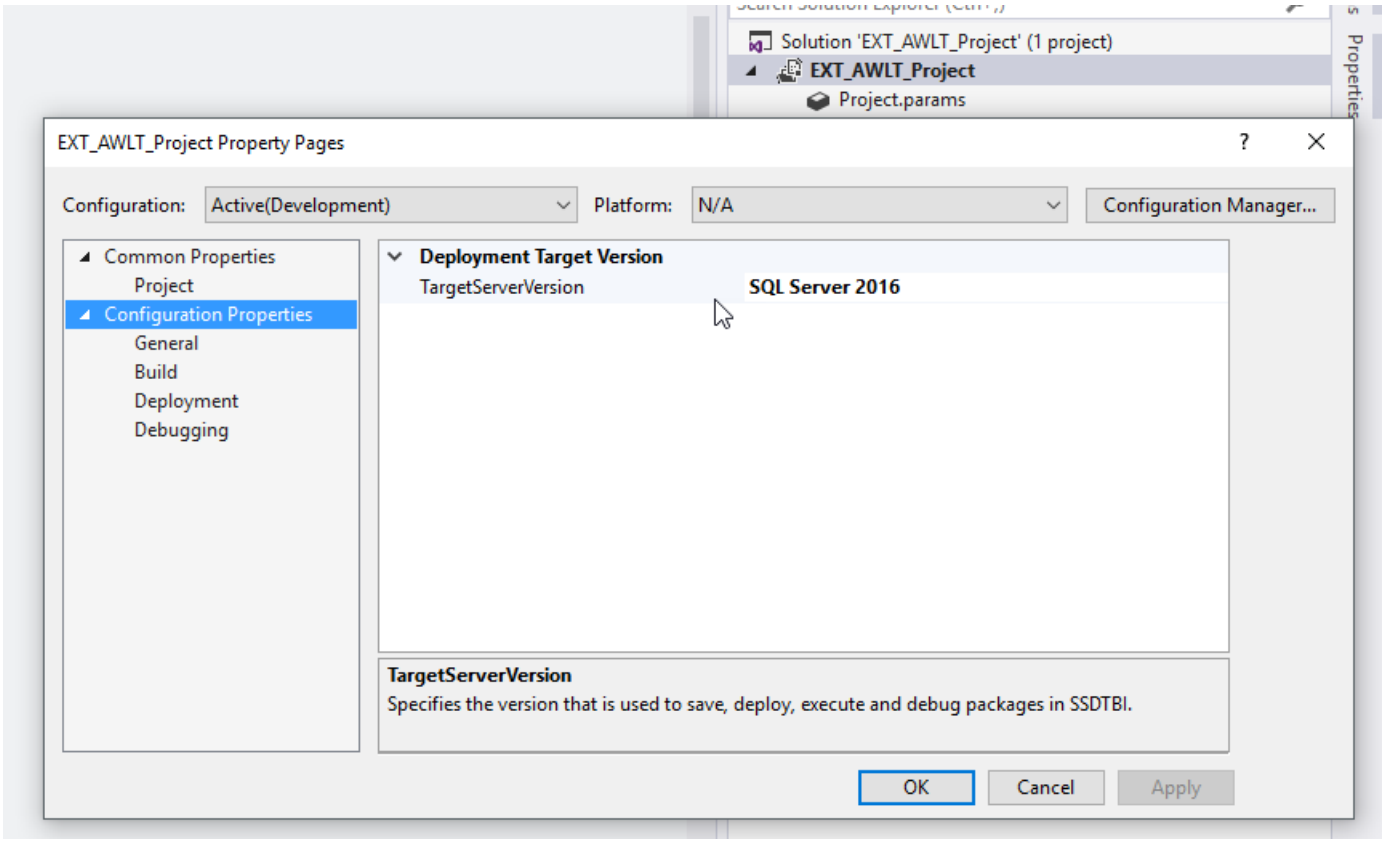
An option to consider is to create an empty Visual Studio Solution in the output folder and add all projects to this solution for easy access to all artifacts from one location.

Once the project is opened in Visual Studio the packages can be reviewed and tested.

For Visual Studio, it is important to match the Visual Studio version to the destination SQL Server. Visual Studio 2010 and 2012 can work with and deploy to SQL Server 2012, Visual Studio 2013 works with SQL Server 2014 and Visual Studio 2015 can be configured to work with SQL Server 2012-2016.

Since the default behavior in Visual Studio 2015 is to work with SQL Server 2016 it is common for issues to occur when deploying to earlier versions of SQL Server.

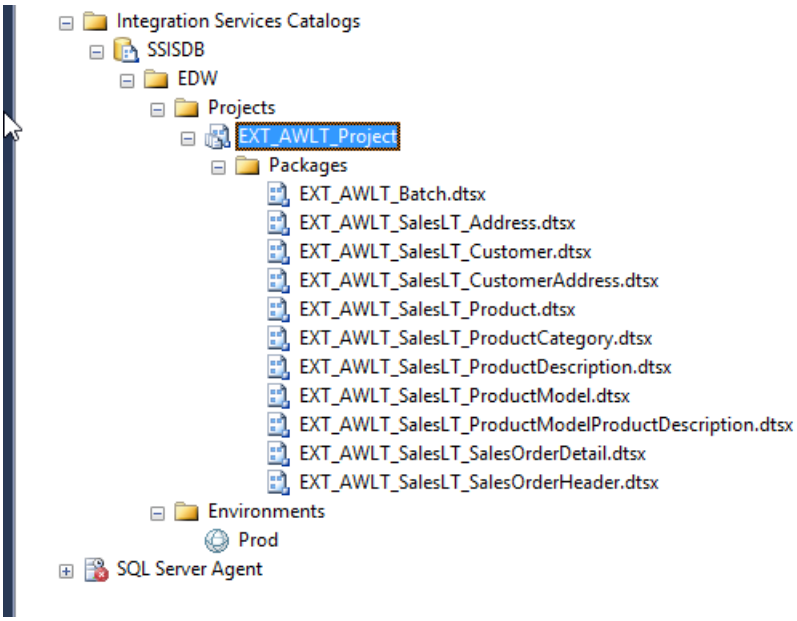




Once the project is ready to be deployed to the SSIS catalog either use the Deploy option from the Project menu can be chosen, or right-click on the project in Solution explorer and Deploy.

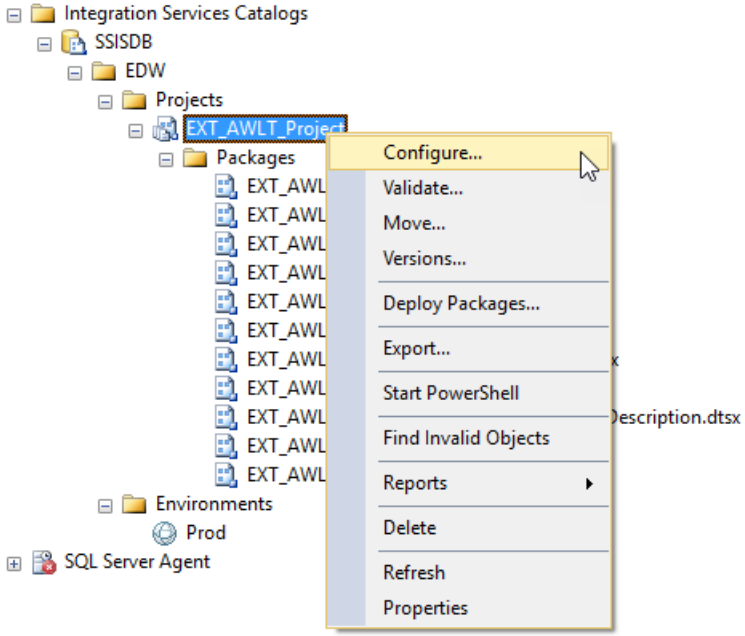
Reviewing the project in the Catalog

Once the project is deployed to the target server and the chosen folder it is available for review through SQL Server Management Studio



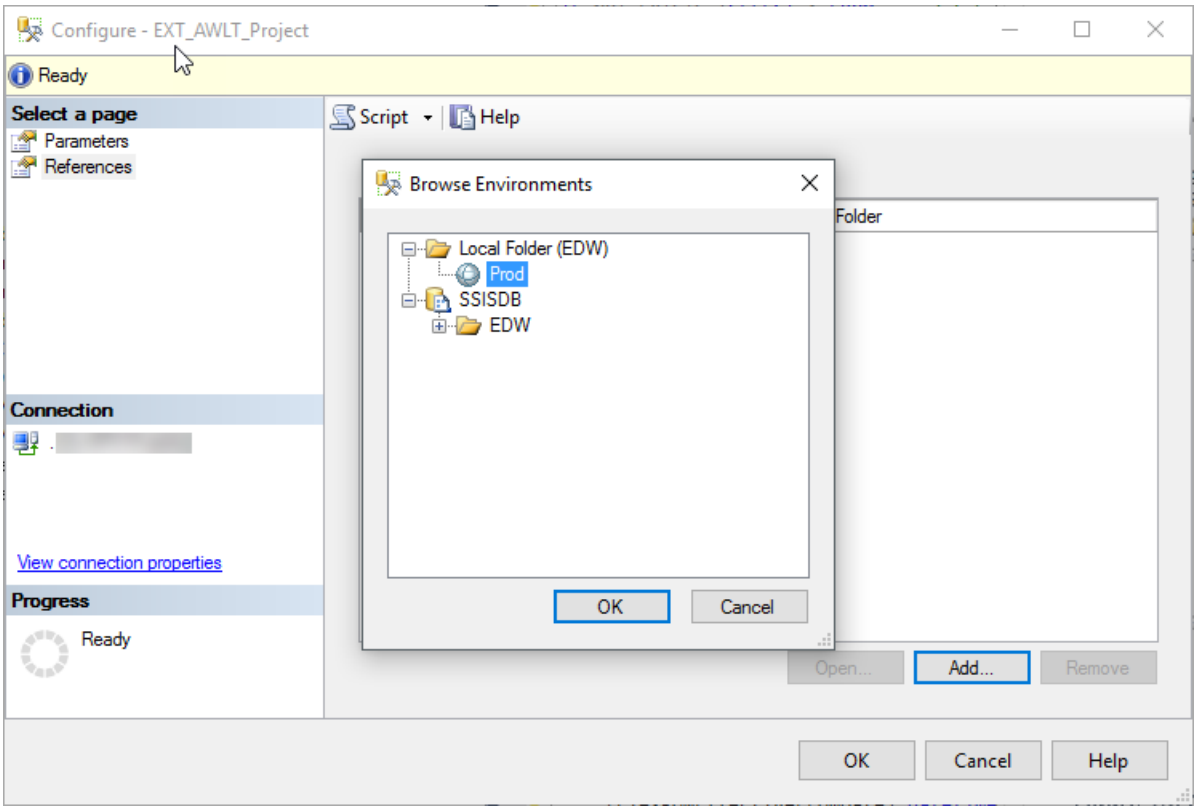
Configuring the Environment and connecting the project to it

Once the project is in the Catalog it can be configured.



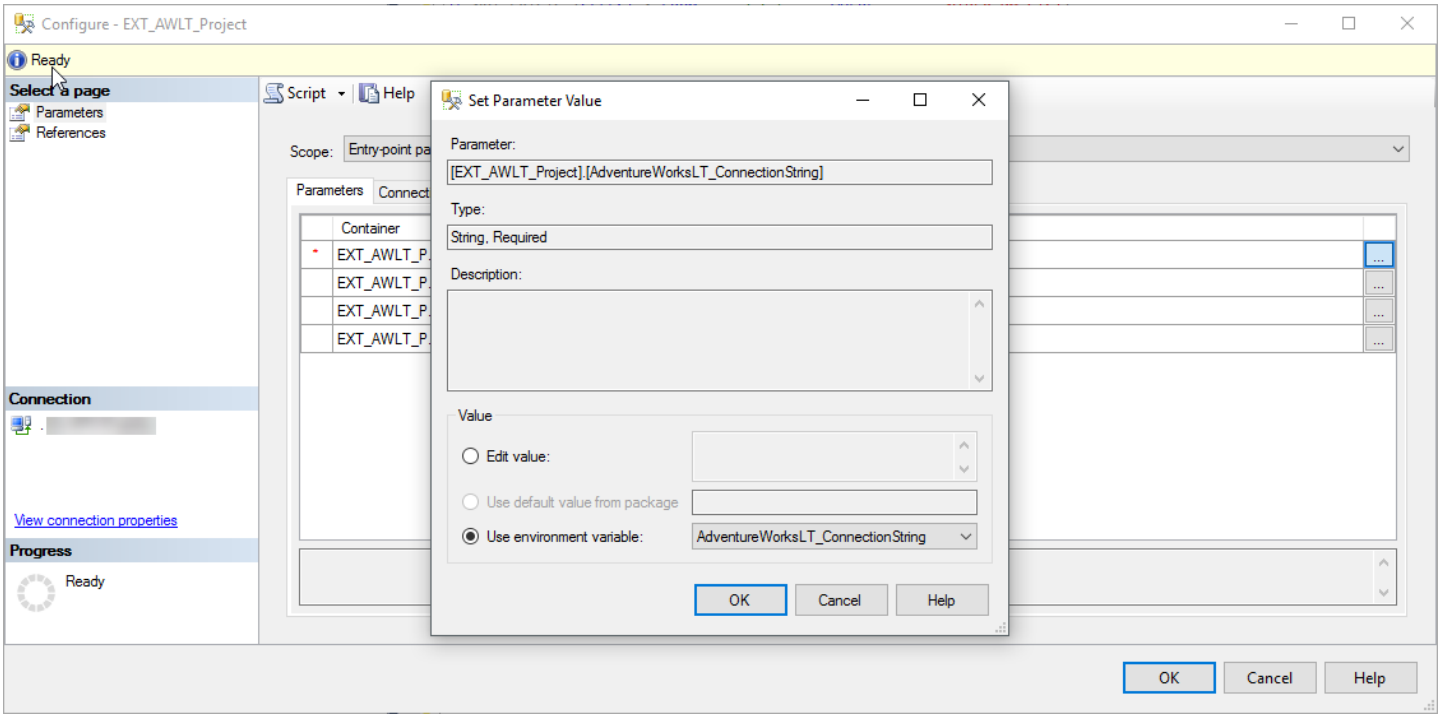
For the environment to be available to the project it needs to be associated

From the configuration dialog, the project can be configured directly and associated with an environment. Enable the project to be configured from the created environment.

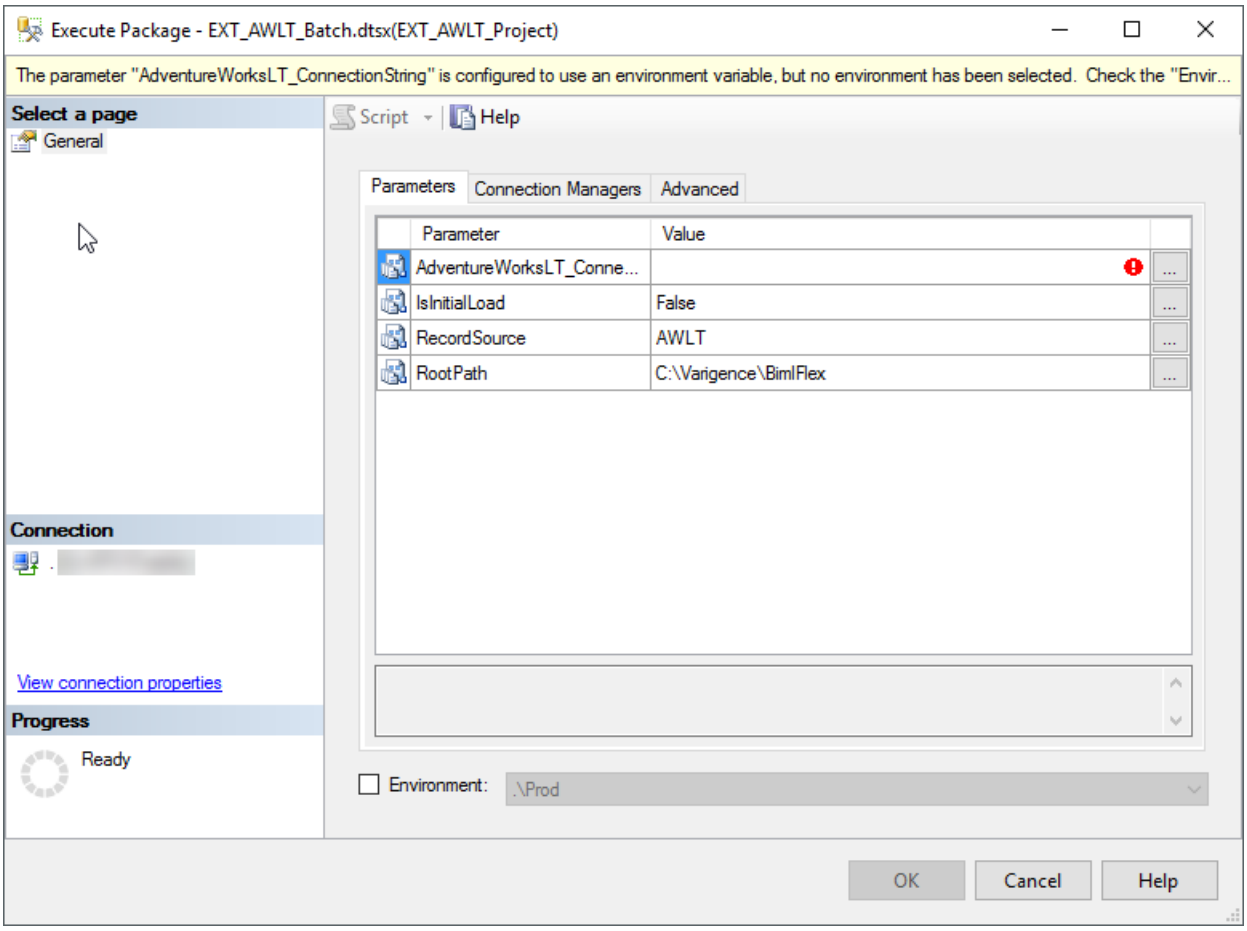


From the references tab, add a reference to the Local Folder Prod Environment.

Once the reference is added, configure the parameters to be overridden from the environment.



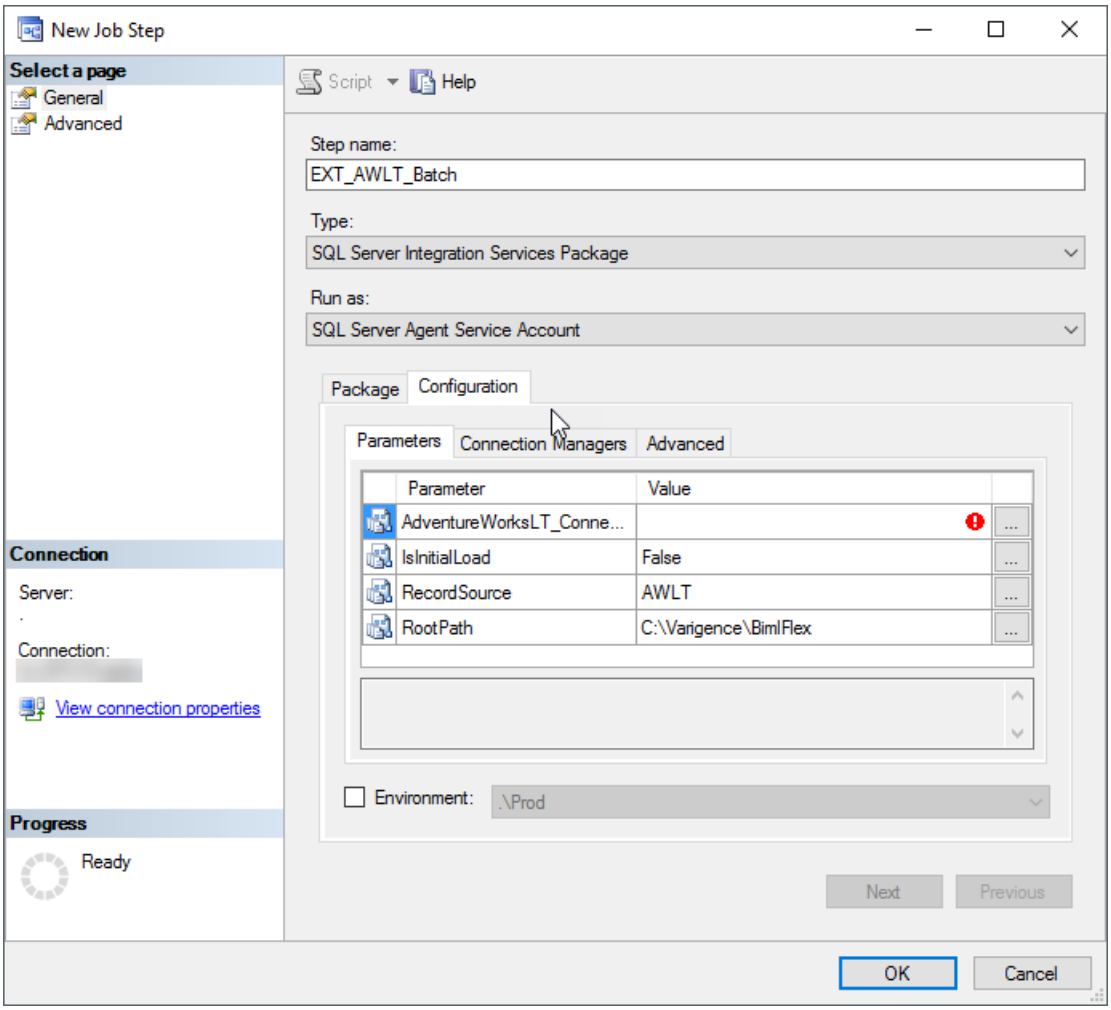
Once the project is configured to use an environment it will always require an environment specification to run, both through SQL Server Agent and the Catalog



Creating an Agent job and configure it to run the batch with a configured environment

Once the project is configured and ready to be scheduled a SQL Server Agent Job can be created to run the projects batch package.

When configuring the job, specify the environment it will use when running.



Specifying the environment through the Agent job also means different scheduled jobs can be run using different environment configurations.

As an example, a scheduled job every 2 minutes for incremental loads can use one environment and another agent job running once a day for full loads can use another environment with different variable values configured.

Closing remarks

There are many ways to configure and run projects for SSIS using package or project configurations, the Catalog or other storage mechanisms, using environments or other configuration mechanisms etc.

For a new SQL Server project, it is recommended to use the project deployment methodology and the SSIS catalog with environments as described here.

This document used the Management Studio UI to detail the steps, there are ways to script this that works better in an automated deploy pipeline.

Once the environment is set up and project configured it is not necessary to reconfigure when new deploys are made. So, this setup is normally a one-time exercise.

There are many other considerations with regards to SSIS deployments and Catalog configurations, such as security definitions for environments, that have been considered out of scope for this guide.

Continuous Integration and Continuous Delivery

One core feature in using the BimlFlex Data Warehouse Automation solution is that it can be used in a CI/CD pipeline.

There are numerous approaches used for CI/CD and Data Warehouses have their special considerations compared to the more common web based or software development based CI/CD processes.

In this document the following approaches are discussed:

- Manual automation of build and deployment steps using command line scripts
- Automatic build and processing of changes through a build server pipeline

The manual scripted approach is a low cost entry into automation in that it doesn't rely on setting up and configuring build servers. It allows developers and testers to script and use the same process over and over again to deploy a solution.

The automated, build server-based approach takes a slightly different approach in that it builds an automatic deployment out of the contents of a source repository/source control project.

Special considerations for Data Warehousing Automation

For a data warehouse it is not enough to make sure the structures and processes are in place for the builds and deployments, it also needs to synchronise dependencies as well as take both existing database structures and data into account. For a classic SQL Server Data Warehouse built with SSIS the build process first needs to create all tables so that the SSIS packages can be built later. The SSIS build process also requires access to both the destination DW tables as well as the source itself. This means utilising online build servers and readily available automation processes for other solution types sometimes needs to be adjusted to fit specific Data Warehousing and ETL process requirements

Special considerations for SSIS

A special note is warranted for solution using SSIS. SSIS can only create packages when the referenced tables, both source and destination are available. As such, a 2-step approach is needed.

1. An initial build step that creates all required tables
2. A second step, running once the tables are available, that builds the SSIS packages. Note that this step also requires access to the sources.

Once the packages are built they can be deployed to the SQL Server SSIS Catalog for testing and running.

Scripted approach

Manual creation of build steps

BimlFlex can locally build out the SQL Server based SSDT project that contains the SQL artefacts used for the project.

Build server based approach

Microsoft's normal build command MsBuild.exe does currently not include the functionality to build out ispaq files from *.dtproj files directly. There are several options for building using Visual Studio through devenv.exe as well as creating custom tasks for msbuild to allow it to build the ispac files. As the Biml compiler builds the ispaq file as part of the normal project build it is possible to skip this entire step and directly use the generated ispaq file. This allows the build and integration process to work smoothly even in environments that doesn't allow custom installations and configurations.

Note that this build command will create the ispac file in the `bin\Development` folder instead of the bin location used by the BimlStudio build described here.

Automate Building

Once a CI server is running it can either build files from sources or deploy compiled project files that have been checked in. Some questions for considering automating the Build:

- can build tools (such as BimlFlex) be installed and licensed on the build server/agent
- can the build server access the BimlFlex metadata repository database

There are pros and cons to using a completely automated build

- Users can forget to compile the code or to check-in the compiled output which could mean that the code is up to date, but the output (ispac file) is not
- Having a central location where build errors are visible is key to responding rapidly to issues

Regardless of if the Builds are automated and triggered on push or PR, scheduled or manually triggered, or if they are manually run without a CI server the following steps are normally required. For a build server process these steps would be performed after the repository has been loaded from the source.

Sample Automation process

Sample Automation process and sample scripts

1. Build the SQL Server SSDT database projects

This step can either use the Biml Compiler or MsBuild to build the artefacts. It connects to the metadata instance and ejects the SSDT Projects. Note that there is an SSDT Project per database.

The build process uses a separate settings file as well as separate build configuration files to only build the required SSDT Project. This allows the developer to have the original settings files for normal development and makes sure the automated process builds the expected result. The Biml Compiler uses a different build settings file to the MSBuild approach.

1. Compile the SSDT Projects

This step uses the generated SSDT Projects and compiles a dacpac file. This dacpac file can then be deployed to a database server.

Before the compilation it is sometimes necessary to consider changes and migrations. This example uses dacpacs for database deployments. These are state-based and thus, only state aware. It is up to the developer to manage the journey from the last state to the new state, both for the schema and the existing data. The dacpac deployment process provides some automated conflict resolution approaches but for complex changes it might require developer consideration.

The dacpac build process uses the MSBuild.exe file to build the dacpac file from the .sqlproj project file

Note

Note that the destination folder under the `output` folder is `SSDT\<CustomerUID>\<VersionName>\<DatabaseName>`. The build script needs to reference the correct rproject file for each database.

This sample script loops through all databases in the defined list and builds them all.

Note

Note that the destination folder for the created dacpac is different for this process compared to the Visual Studio based one. By default Visual Studio ejects the dacpac to a subfolder called `Development`

Sample Scripts

Sample Script using the Biml compiler

location: project root folder filename: `_1.build_sql_bimlc.bat`

```
@echo off
rem (c) Varigence 2018
rem https://varigence.com/BimlFlex

pushd %~dp0

rem call the Biml Compiler with the specific resp file, use the path to the installed version of BimlStudio
rem 64 bit: %programfiles%
rem 32 bit: %programfiles(x86)%

"%programfiles%\Varigence\BimlStudio\5.0\bimlc.exe" @"output\SqlOnly.mst.bimlc.resp"
```

Sample Script using MSBuild

location: project root folder filename: `_1.build_sql_msbuild.bat`

```
@echo off
rem (c) Varigence 2018
rem https://varigence.com/BimlFlex

pushd %~dp0

rem call msbuild.exe with the specific resp file, use the path to a compatible, installed version of
msbuild.exe

C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe @"output/SqlOnly.mst.resp"
```

1. Deploy the Dacpac to SQL Server

Once the dacpac is created (including any custom, bespoke, migration logic) it can be deployed to the destination SQL Server. The deployment step uses SqlPackage.exe to deploy the dacpac to the specified destination SQL Server.

Sample Script for deployment

location: project root folder filename: `_2.build_sql_dacpac.bat`

```
@echo off
rem (c) Varigence 2018
rem https://varigence.com/BimlFlex

pushd %~dp0

rem set these to match your environment

SET "MsBuildVersion=14.0"
SET "CustomerUID=5528d0c8-2ead-42e3-9c0e-7826f25527b7"
SET "VersionName=Version 1"
SET "DatabaseList=(CI_BFX_STG,CI_BFX_ODS)"

rem call MSBuild to build the dacpac for the ssdt projects.
rem this loops through all databases specified in the DatabaseList variable array
rem note that the path to msbuild needs to be specified and match machine.

for %%i in %DatabaseList% do "%programfiles(x86)%\MSBuild\%MsBuildVersion%\Bin\MSBuild.exe"
"output\\SSDT\\%CustomerUID%\\%VersionName%\\%%i\\%%i.sqlproj"
```

Sample Script to deploy dacpac

location: project root folder filename: `_3.deploy_sql_dacpac.bat`

```
@echo off
rem (c) Varigence 2018
rem https://varigence.com/BimlFlex

pushd %~dp0

rem set these to match your environment

SET "SqlServerVersionPath=140"
SET "CustomerUID=5528d0c8-2ead-42e3-9c0e-7826f25527b7"
SET "VersionName=Version 1"
SET "DatabaseList=(CI_BFX_STG,CI_BFX_ODS)"
SET "ServerName=localhost\SQL2016"

rem call SqlPackage.exe to deploy the dacpacs to the SQL Server instance for the ssdt projects.
rem this loops through all databases specified in the DatabaseList variable array

for %%i in %DatabaseList% do "%programfiles(x86)%\Microsoft SQL
Server\%SqlServerVersionPath%\DAC\bin\SqlPackage.exe" /Action:Publish
/SourceFile:"output\\SSDT\\%CustomerUID%\\%VersionName%\\%%i\\bin\\Output\\%%i.dacpac"
/TargetDatabaseName:%%i /TargetServerName:%ServerName%
```

Sample Script to build SSIS Project with Biml Compiler

location: project root folder filename: `_4.build_ssis_bimlc.bat`

```
@echo off
rem (c) Varigence 2018
rem https://varigence.com/BimlFlex

pushd %~dp0

rem call the Biml Compiler with the specific resp file to build the SSIS projects

"%programfiles%\Varigence\BimlStudio\5.0\bimlc.exe" @"output\SsisOnly.mst.bimlc.resp"
```

Sample File to build SSIS Packages with MSBuild

location: project root folder filename: `_4.build_ssis_msbuild.bat`

```
@echo off
rem (c) Varigence 2018
rem https://varigence.com/BimlFlex

pushd %~dp0

rem call msbuild.exe with the specific resp file, use the path to a compatible, installed version of
msbuild.exe

C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe @"output/SsisOnly.mst.resp"
```

Optional Step to build ispac files

location: project root folder filename: `_5.build_ssis_ispaq.bat`

Note

Note that the normal BimlStudio build creates the ispac file in the SSIS build step. This optional, separate, step requires a compatible version of Visual Studio capable of building ispac files.

```
@echo off
rem (c) Varigence 2018
rem https://varigence.com/BimlFlex

pushd %~dp0

rem set these to match your environment

SET "ProjectList=(EXT_AWLT)"
SET "VisualStudioPath=Microsoft Visual Studio 14.0"

rem build the ispac files from the SSIS Projects
rem this loops through all projects specified in the ProjectList variable array

for %%i in %ProjectList% do "%programfiles(x86)%\%VisualStudioPath%\Common7\IDE\devenv.com"
"output\\%%i\\%%i_Project.dtproj" /rebuild
```

Deploy ispac file with SSIS Packages to SSIS Catalog

location: project root folder filename: `_6.deploy_ssis_ispac.bat`

```
@echo off
rem (c) Varigence 2018
rem https://varigence.com/BimlFlex

pushd %~dp0

rem set these to match your environment

SET "SqlServerVersionPath=130"
SET "ProjectList=(EXT_AWLT)"
SET "ServerName=localhost\SQL2016"
SET "SsisDbName=SSISDB"
SET "FolderName=4-ci-demo"

rem create the folder in the SSISDB catalog if needed

sqlcmd -S %servername%  -d %ssisdbname% -Q "EXEC catalog.create_folder @folder_name = '%FolderName%';"

rem call isdeploymentwizard.exe to deploy the ispacs to the SQL Server SSIS Catalog instance for the SSIS
projects.
rem this loops through all Projects specified in the ProjectList variable array

for %%i in %ProjectList% do "%programfiles(x86)%\Microsoft SQL
Server\%SqlServerVersionPath%\DTS\Binn\isdeploymentwizard.exe" /S /SP:"output\%%i\bin\%%i_Project.ispac"
/DS:%ServerName% /DP:"/%SsisDbName%/%FolderName%/%%i/"
```

Sample settings file for SQL Only builds

location: project root folder filename: `SqlOnly.bimlb.settings`

Update this file to reflect project settings and configurations

```
[
    {
        "Namespace": "",
        "Name": "CustomerUID",
        "Value": "5528d0c8-2ead-42e3-9c0e-7826f25527b7"
    },
    {
        "Namespace": "",
        "Name": "Server",
        "Value": ".\\sql2016"
    },
    {
        "Namespace": "",
        "Name": "Database",
        "Value": "BimlFlex_Testing"
    },
    {
        "Namespace": "",
        "Name": "Version",
        "Value": "Version 1"
    },
    {
        "Namespace": "",
        "Name": "Provider",
        "Value": "SQLNCLI11"
    },
    {
        "Namespace": "",
        "Name": "UseWindowsAuthentication",
        "Value": true
    },
    {
        "Namespace": "",
        "Name": "UserId",
        "Value": ""
    },
    {
        "Namespace": "",
        "Name": "Password",
        "Value": ""
    },
    {
        "Namespace": "",
        "Name": "RememberPassword",
        "Value": false
    },
    {
        "Namespace": "",
        "Name": "IsUserMode",
        "Value": false
    },
    {
        "Namespace": "",
        "Name": "ToggledOffFiles",
        "Value": "BimlScripts\\1.00.1-flx-src-stg-main.biml|BimlScripts\\1.20.1-flx-psa-stg-
main.biml|BimlScripts\\1.70.1-flx-src-to-file-main.biml|BimlScripts\\1.80.1-flx-src-to-file-main-
express.biml|BimlScripts\\2.20.1-flx-dv-source-main.biml|BimlScripts\\3.10.1-flx-dwh-
main.biml|BimlScripts\\3.20.1-flx-mds-main.biml|BimlScripts\\3.50.1-flx-dwh-sql-
main.biml|BimlScripts\\3.50.2-flx-dwh-source-sql-
main.biml|BimlScripts\\_OutputFlatBiml.biml|BimlScripts\\_OutputFlatDDL.biml"
    }
]
```

Sample settings file for SSIS Only builds

location: project root folder filename: `SsisOnly.bimlb.settings`

Update this file to reflect project settings and configurations


```
[
    {
        "Namespace": "",
        "Name": "CustomerUID",
        "Value": "5528d0c8-2ead-42e3-9c0e-7826f25527b7"
    },
    {
        "Namespace": "",
        "Name": "Server",
        "Value": ".\\sql2016"
    },
    {
        "Namespace": "",
        "Name": "Database",
        "Value": "BimlFlex_Testing"
    },
    {
        "Namespace": "",
        "Name": "Version",
        "Value": "Version 1"
    },
    {
        "Namespace": "",
        "Name": "Provider",
        "Value": "SQLNCLI11"
    },
    {
        "Namespace": "",
        "Name": "UseWindowsAuthentication",
        "Value": true
    },
    {
        "Namespace": "",
        "Name": "UserId",
        "Value": ""
    },
    {
        "Namespace": "",
        "Name": "Password",
        "Value": ""
    },
    {
        "Namespace": "",
        "Name": "RememberPassword",
        "Value": false
    },
    {
        "Namespace": "",
        "Name": "IsUserMode",
        "Value": false
    },
    {
        "Namespace": "",
        "Name": "ToggledOffFiles",
        "Value": "BimlScripts\\0.01.0-flx-import-stg-tables.biml|BimlScripts\\0.02.0-flx-import-rdv-
tables.biml|BimlScripts\\0.03.0-flx-import-dwh-tables.biml|BimlScripts\\0.04.0-flx-import-psa-
tables.biml|BimlScripts\\1.20.1-flx-psa-stg-main.biml|BimlScripts\\1.70.1-flx-src-to-file-
main.biml|BimlScripts\\1.80.1-flx-src-to-file-main-express.biml|BimlScripts\\2.20.1-flx-dv-source-
```

```
main.biml|BimlScripts\\3.10.1-flx-dwh-main.biml|BimlScripts\\3.20.1-flx-mds-main.biml|BimlScripts\\3.50.1-
flx-dwh-sql-main.biml|BimlScripts\\3.50.2-flx-dwh-source-sql-
main.biml|BimlScripts\\_OutputFlatBiml.biml|BimlScripts\\_OutputFlatDDL.biml|BimlScripts\\_OutputSsdtDDL.biml
"
    }
]
```

Sample SqlOnly.mst.bimlc.resp settings file

location: output folder filename: `SqlOnly.mst.bimlc.resp`

```
--targetPath="output" --buildDocumentation=False --docOutputPath="documentation" --bundle="BimlFlex.bimlb" --
bundleSetting="SqlOnly.bimlb.settings" --version=SqlServer2016 --version=Ssas2014 --version=SsasTabular2016 -
-version=Ssis2016 --ssisDeploymentModel=Project --ddlBuildMode=None --warnAsError=False --warn=4 --
cleanOutputFolder=True
```

Sample SqlOnly.mst.ProjectView.bimlproj settings file

location: output folder filename: `SqlOnly.mst.ProjectView.bimlproj`

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <BuildDocumentation>False</BuildDocumentation>
    <DocumentationOutputPath>documentation</DocumentationOutputPath>
  </PropertyGroup>
  <ItemGroup Condition="true">
    <Bundles Include="../BimlFlex.bimlb" />
    <BundleSettings Include="../BimlFlex-sqlonly.bimlb.settings" />
  </ItemGroup>
  <Import Project="$(AssemblyPath)\Varigence.Biml.targets" Condition="true" />
  <!--
  Any changes outside of the auto-generated ItemGroup and PropertyGroup tags
  will be ignored by the Configuration designer.
  Any content within the auto-generated ItemGroup and PropertyGroup tags can be overwritten by the
Configuration designer.
  -->
</Project>
```

Sample SqlOnly.mst.resp settings file

location: output folder filename: `SqlOnly.mst.resp`

```
"output\4-ci-demo-sqlonly.mst.ProjectView.bimlproj" /p:OutputPath="output" /p:SqlVersion=SqlServer2016
/p:SsasVersion=Ssas2014 /p:SsasTabularVersion=SsasTabular2016 /p:SsisVersion=Ssis2016
/p:SsisDeploymentModel=Project /p:DdlBuildMode="None" /p:WarnAsError=False /p:Warn=4
/p:CleanOutputFolder=False /p:TaskName=Varigence.Biml.Engine.MSBuild.BimlCompilerTask
/p:AssemblyFile="C:\Program Files (x86)\Varigence\BimlStudio\5.0\BimlEngine.dll" /p:AssemblyPath="C:\Program
Files (x86)\Varigence\BimlStudio\5.0"
```

Sample SsisOnly.mst.bimlc.resp settings file

location: output folder filename: `SsisOnly.mst.bimlc.resp`

```
--targetPath="output" --buildDocumentation=False --docOutputPath="documentation" --bundle="BimlFlex.bimlb" --
bundleSetting="BimlFlex-SsisOnly.bimlb.settings" --version=SqlServer2016 --version=Ssas2014 --
version=SsasTabular2016 --version=Ssis2016 --ssisDeploymentModel=Project --ddlBuildMode="None" --
warnAsError=False --warn=4 --cleanOutputFolder=True
```

Sample SsisOnly.mst.ProjectView.bimlproj settings file

location: output folder filename: `SsisOnly.mst.ProjectView.bimlproj`

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <BuildDocumentation>False</BuildDocumentation>
    <DocumentationOutputPath>documentation</DocumentationOutputPath>
  </PropertyGroup>
  <ItemGroup Condition="true">
    <Bundles Include="../BimlFlex.bimlb" />
    <BundleSettings Include="../BimlFlex-sqlonly.bimlb.settings" />
  </ItemGroup>
  <Import Project="$(AssemblyPath)\Varigence.Biml.targets" Condition="true" />
  <!--
  Any changes outside of the auto-generated ItemGroup and PropertyGroup tags
  will be ignored by the Configuration designer.
  Any content within the auto-generated ItemGroup and PropertyGroup tags can be overwritten by the
Configuration designer.
  -->
</Project>
```

Sample SsisOnly.mst.resp settings file

location: output folder filename: `SsisOnly.mst.resp`

```
"output\4-ci-demo-sqlonly.mst.ProjectView.bimlproj" /p:OutputPath="output" /p:SqlVersion=SqlServer2016
/p:SsasVersion=Ssas2014 /p:SsasTabularVersion=SsasTabular2016 /p:SsisVersion=Ssis2016
/p:SsisDeploymentModel=Project /p:DdlBuildMode="None" /p:WarnAsError=False /p:Warn=4
/p:CleanOutputFolder=False /p:TaskName=Varigence.Biml.Engine.MSBuild.BimlCompilerTask
/p:AssemblyFile="C:\Program Files (x86)\Varigence\BimlStudio\5.0\BimlEngine.dll" /p:AssemblyPath="C:\Program
Files (x86)\Varigence\BimlStudio\5.0"
```

Installing BimlStudio

Supporting Videos

Supporting BimlFlex Documentation

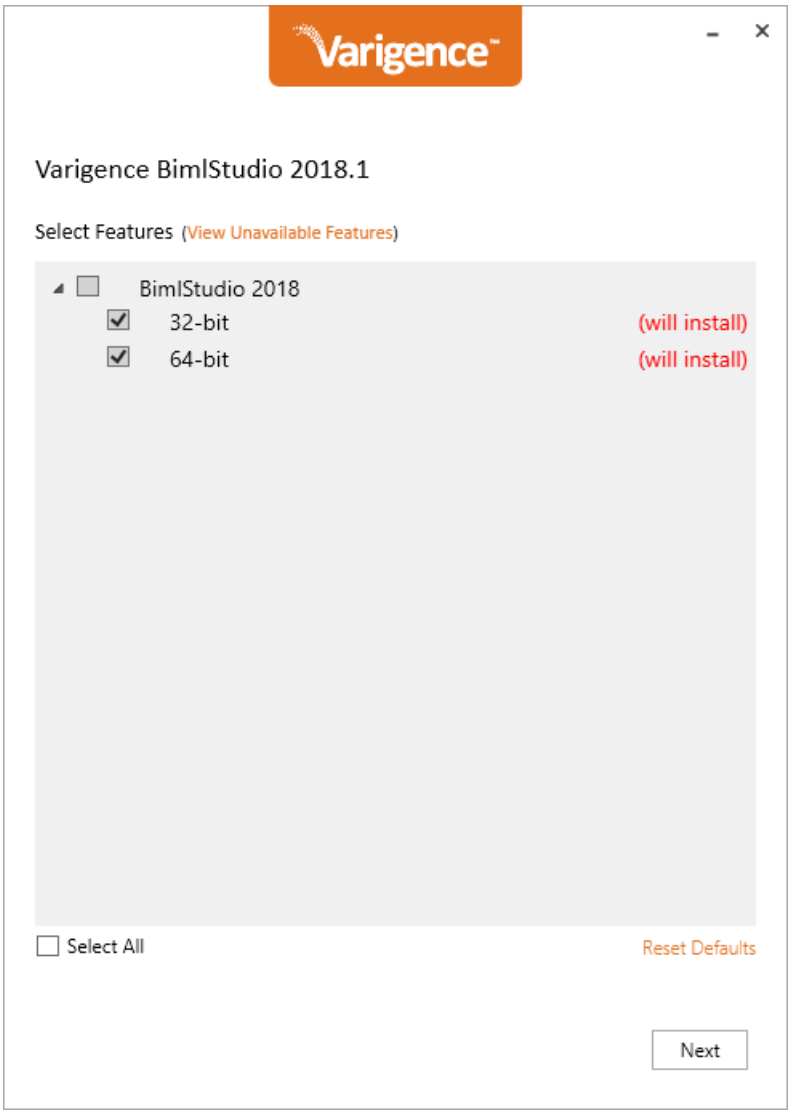
- Developer Installation
- BimlStudio User Guide
- Analyst Installation
- @bimlflex-server-installation

BimlStudio

BimlStudio is the development environment provided for BimlFlex. The installation provides the platform for creating and working with BimlFlex projects. The building of SSIS packages and SSIS projects also require that the matching Visual Studio and SQL Server Data Tools with SSIS support is available on the machine. For machines where there is a local installation of a 64 bit SQL server there will be 64 bit build components available. For an installation without SQL Server but with Visual Studio and SSDT there will be 32 bit build components available.

BimlStudio is available in 2 bitness flavors, 32 and 64 bit. For the trial it is recommended that both versions are installed.

For scenarios where only 32 bit SSIS build components are available locally it is still possible to run the 64 bit version of BimlStudio and build using the 32 bit components by configuring the build process to target 32 bit.



Once the installation is completed it is possible to start BimlStudio.

The first time it is opened it will ask for a license key. The trial license key is provided in the trial welcome email together with the download locations and documentation. Adding the license key enables the use of BimlStudio and the BimlFlex functions.

BimlStudio supports both classical BimlStudio projects and BimlFlex projects. For the BimlFlex projects to work for the trial it is important that the databases are created and that the BimlFlex Bundle file is updated through the BimlFlex Utility Application.

Detailed Steps

The following detailed steps walks through the installation of BimlStudio

Download BimlStudio

Navigate to the BimlStudio page or use the link in the Trial email to download the installer.

Install BimlStudio

Run the installer to install the application. When presented with the choice of 32 and/or 64 bit installation, choose to install both versions.

Enter product key

Once BimlStudio is started for the first time, enter your Trial key from the trial welcome email to activate the product.

Installing BimlFlex

Supporting Videos

Supporting BimlFlex Documentation

- Developer Installation
- Analyst Installation
- @bimlflex-server-installation

BimlFlex

The BimlFlex installation installs the BimlFlex Excel-based metadata editor. This is a plugin in to Excel and will be visible as a separate tab in the Excel ribbon when the BimlFlex-enabled Excel workbook is opened from a BimlFlex project.

When installing it is important to match the Excel bitness version. Verify the Excel bitness version before starting the installation and only install the matching version of BimlFlex.

Detailed Steps

The following detailed steps walks through the installation of BimlFlex

Download BimlFlex

Use the link in the Trial email to download the installer

Install BimlFlex

Run the installer to install the application. When presented with the choice of 32 and/or 64 bit installation, choose the version that matches your local Excel installation and install only that version.

Enter product key

Once the BimlFlex Excel based metadata editor is started for the first time, enter your Trial key to activate the product.

Installing Custom Ssis Components

Supporting Videos

Supporting BimlFlex Documentation

- Server Installation

Installing Custom SSIS Components

BimlFlex uses a set of custom SSIS components in the data processing to enable additional functionality. The components are part of the open source BimlCatalog project available on GitHub.

Direct link for downloading the SQL Server 2016 custom components

The components are installed through the supplied `install.bat` file

Detailed Steps

The following detailed steps walks through the installation of the custom SSIS components required for BimlFlex SSIS packages

Downloading the components

Download the components using the link in the Trial email or through the BimlCatalog repository link and store the file locally.

Direct link for downloading the Sql Server 2016 custom components

Unpack the file

Once downloaded, unpack the zip file contents to a convenient location

Running the installation batch file

Run the `install.bat` file with administrator privileges. This will install the custom components into the required locations. As the installation copies the components to the global assembly cache it needs to run using admin privileges.

Creating and configuring a BimlFlex project

Supporting Videos

Supporting BimlFlex Documentation

- Initial Setup and Configuration

Create and configure the BimlFlex project

The BimlFlex project is comprised of the project files on disk that BimlStudio interacts with and the metadata stored in the BimlFlex metadata repository database.

The project has a connection defined to the Metadata Repository to use. The default database name is BimlFlex and that database was created using the utility application in the previous step. The first time the project is opened it needs to be configured for the SQL Server version used and SSIS deployment method. Once the project is configured it is ready to be used for data modelling and Data Warehouse generation.

Detailed Steps

The following detailed steps walks through the creation of a BimlFlex project.

Prerequisites

The previous steps in the trial process needs to have been completed. This includes:

- The latest version of BimlStudio and BimlFlex are installed.
- The BimlFlex Utility Application has been used to create the BimlFlex and BimlCatalog Databases.
- The BimlFlex Utility Application has been used to update the Bundle File in the installation folders.

Start BimlStudio

When BimlStudio is started there is an option to create a new BimlFlex project.

Create BimlFlex Project

Click the Create BimlFlex project button to create a new project. Enter a valid location for the project and name it `BimlFlex Trial Project`.

Define Metadata Connection

Define the connection and login information to the BimlFlex metadata database to use. This database was created using the Utility

Application.

Create Metadata Customer

In the BimlFlex configuration, create a new Customer and name it `BimlFlex Trial Project`

Configure BimlStudio to build for Sal Server 2016 SSIS

Right-click the project name `BimlFlex Trial Project.mst` in the logical view and choose properties to display the BimlStudio project properties. Update the SQL Server version setting to build out to SQL Server 2016.

Click the `Save All` button or menu option to save the metadata and project settings.

Restart BimlStudio

BimlStudio needs to be restarted once the customer metadata settings and SSIS configuration has been updated. When restarted the configurations are active in BimlStudio.

Open and configure BimlFlex Excel Metadata Editor

Once BimlStudio has restarted, navigate to the BimlFlex Tab and click the `Excel Metadata Editor` button to start the Excel Metadata Editor.

If this is the first time it is opened, enter the license key in the activation dialog.

Once opened, configure the metadata connection to point to the Trial Project.

Click the `Get All Entities` button in the BimlFlex ribbon to query the repository for the latest metadata set. The first time a request is sent to an empty customer BimlFlex allows creation of sample metadata. Click `Create Sample` to create the sample metadata.

Save the Excel file to save the connection information for next time.

Creating the scaffolding metadata

Supporting Videos

Supporting BimlFlex Documentation

- Initial Setup and Configuration

Opening the Excel metadata editor

The first time the Excel metadata editor requests data from a new customer it will ask if sample metadata should be created. Creating the sample metadata will create the scaffolding for the trial process.

Scaffolding metadata was created as part of Creating and Configuring a BimlFlex project. This documentation describes how to recreate it by creating a new customer.

It is also possible to reload metadata from a specific point in the trial process by using the prepared trial metadata

Detailed Steps

The following detailed steps walks through the creation of the BimlFlex scaffolding metadata

Generating the scaffolding metadata

For a new customer, click the `Get All Entities` button in the BimlFlex tab in the Excel Metadata editor. Create the sample metadata by clicking the button in the dialog.

The trial process assumes the connections and database names are all kept as the default names. The connection strings to the databases on the connections sheet should be updated to match the SQL Server used for the data warehouse. It is recommended to use Integrated Security.

Setting up AdventureWorks LT source database

Supporting Videos

Supporting Documentation

- Microsoft Sql Server Samples

Setting up AdventureWorks LT

The trial process uses the AdventureWorks 2012 LT database as source. This Microsoft sample database is available for download from Codeplex and its new location on GitHub. AdventureWorks LT on GitHub

Download the data file and attach it to the Sql Server engine through Management Studio to make it available for the trial process.

Detailed Steps

1. Download the .mdf Sql Server database file.
2. Copy it to the database file location used by the Sql Server engine.
3. Use Management Studio to attach the database file to the instance.

Importing Source Metadata

Supporting Videos

Supporting BimlFlex Documentation

- Importing Metadata

Import Trial Source Metadata

Importing existing metadata from database connections is a fast and easy way to start modelling data from a source to a Data Warehouse. By reading existing metadata instead of manually entering it or manually creating tables for the Data Warehouse it is possible to increase development speed and quality.

The Excel Metadata Editor provides a Ribbon Tab for working with the BimlFlex metadata.

To import metadata, the connection information to the source has to be valid. Validate the connection strings in the connections sheet.

It also needs a properly defined Batch. A Batch is the orchestration container used for load processes.

It also needs a properly defined Project. A Project corresponds to an SSIS project.

Once the connections, batches and projects are validated the metadata can be imported.

BimlFlex will read all specified metadata from the chosen connection to the Excel sheets.

The metadata import dialog provides a set of options for managing

- What to import
- Naming conventions
- Inferred metadata
- Options
- how to treat already imported Metadata

Once all settings have been reviewed the import function will read the metadata from the source into the Excel sheets. Tables are added to the Objects sheet. Columns are added to the Columns sheet.

Once validated the metadata can be written to the metadata repository.

Detailed Steps

1. Open the Excel Metadata Editor from BimlStudio.

   The Excel Metadata Editor provides a Ribbon Tab for working with the BimlFlex metadata.

2. Validate the metadata connection information.

3. To get the latest metadata, click the top part of the Get All Entities Button to load all sheets with the data from the repository.

4. Validate the connection strings in the connections sheet.

   The metadata import will use the connection to the AdventureWorksLT source database. This connection has been defined as a source and with the `AWLT` Record Source code.

5. Validate Connections, Batches and Projects sheets.

6. Click the Import Metadata button to display the dialog.

7. Choose `AdventureWorks LT` as the source and `EXT_AWLT` as the destination project.

   BimlFlex will read all available metadata. Use the filtering options for large sources.

8. Deselect the `dbo` schema. Deselect the `import views` option to only import metadata from the tables in the `SalesLT` schema.

9. Click Import to read the metadata from the source into the Excel sheets.

   Tables are added to the `Objects` sheet. Columns are added to the `Columns` sheet.

10. To write the imported metadata to the repository, click the `Set Current Sheet` on the `Objects` sheet and on the `Columns` sheet.

11. To validate that all metadata is available, click get all entities to reload everything from the repository.

    The `Objects` sheet now shows all imported tables and the `Columns` sheet shows all columns for all tables.

Building Databases, Tables and SSIS packages for source to staging

Supporting Videos

Supporting BimlFlex Documentation

- Source to Staging Templates

Building Databases, Tables and SSIS packages for source to staging in BimlFlex

In BimlStudio, the BimlFlex project is able to create SQL DDL scripts to create databases and tables for the Data Warehouse. These can be run on the database server to create all artefacts required to run the trial process.

BimlFlex can also build out a Visual Studio SQL Server Data Tools (SSDT) Project with all database artefacts. This feature is enabled by default and this project can be used to deploy the databases and tables.

Once the tables are available it is possible to use the build process in BimlStudio to build the SSIS project and packages for the source to staging and persistent staging loads.

The creation and building of the initial source to staging will validate that all local configurations and build tools are in place and working as expected. it will also validate access to the database server used for the data warehouse.

The default build process assumes that Windows authentication is used to access the databases. Should SQL logins be needed the project needs to be configured with project parameters for connection strings. This is outside the scope of the Trial but reference documentation is available here

Detailed Steps

The following detailed steps walks through the creation of database scripts and building of the SSIS project:

Refresh metadata

To start, click the `Refresh Metadata` button to make sure BimlFlex has the latest metadata

Creating SQL DDL scripts to create databases and Tables

In BimlStudio, navigate to the `BimlFlex` tab. In the `Generate Scripts` dropdown, choose `Create Table Script`. BimlFlex will generate the database, schema and table create scripts. Copy the script or open the file in Microsoft SQL Server Management Studio

and execute them on the target SQL Server.

Building SSIS packages in BimlStudio

Once the tables are available it is possible to use the build process in BimlStudio to create the SSIS project and packages for the load process.

In BimlStudio, navigate to the `Build & Deploy` tab, choose the `32-bit` build option in the dropdown unless there is a local installation of 64 bit SQL Server and click the `Build` anvil and hammer icon.

Opening and reviewing the SSIS project in Visual BimlStudio

The output folder for generated artefacts is configurable, the default location is a folder called `output` in the BimlFlex project location.

In the output folder there are multiple folders for the created artefacts. BimlFlex creates default SSIS projects to create databases, schemas and tables as well as expanded code for both Biml and SQL. The load project is located in a folder with the same name as the project name defined in the Metadata project sheet.

For the Trial this will be a folder called `EXT_AWLT`. Open the `EXT_AWLT_Project.dtproj` project file in Visual Studio 2015 to review the generated packages. To test the load, run the `EXT_AWLT_Batch.dtsx` SSIS package. This package will call the individual Table load packages and load the data from the source database to the staging and persistent staging databases in the Data Warehouse.

Configuration of BimlFlex

Supporting Videos

Supporting BimlFlex Documentation

Configurations

Configurations and Settings in BimlFlex

Once the project and metadata connection is defined it is possible to work with the metadata in the Excel-based metadata management application. One of the key features in the metadata is the configuration of the BimlFlex framework. Most aspects can be configured, including naming conventions, metadata added to layers, hashing approaches and base accounts and locations to use.

Detailed Steps

The following detailed steps walks through the recommended configuration of the BimlFlex project for the trial.

Open and Refresh Metadata

Open the Excel Metadata Editor and click the `Get All Settings` to read all configurations and settings from the repository.

Review Configurations

The Configurations sheet contains the BimlFlex configurations for core metadata. Here metadata columns and data types are defined as well as if they are included in different layers.

Review Settings

The Settings Sheet contains configurable settings such as naming conventions used.

For the Trial it is recommended to update the Hash Algorithm to match the `HASHBYTES()` function in SQL.

Update the `UseSqlCompatibleHash` setting in the settings sheet to `Y`

Once the value is updated, click `Set Current Sheet` on the BimlFlex Ribbon Tab to set the value in the database.

Review Data Type Mappings

The Data Type Mappings sheet contains the BimlFlex Data Type Mappings that can be used to expand data types for incoming data.

Review Versions

The Versions sheet contains the current Customer Versions. Enabled Versions are visible in the drop down when configuring the metadata connection.

Modelling of source Metadata

Supporting Videos

TODO: Coming Soon

Supporting BimlFlex Documentation

- Implementation Guides

Modelling of source Metadata

Once the source system metadata has been imported in to BimlFlex it is possible to model it in the Excel-based metadata management application.

Modelling through metadata management is the way to tweak the process to match business requirements and allows BimlFlex to automatically generate the required Sql structures and SSIS packages for the Etl process.

Detailed Steps

The following detailed steps walks through the options for modelling of source Metadata as recommended for the trial

Reviewing keys

BimlFlex has 3 types of keys managed through metadata:

- Primary Key
- Business Key
- Alternate Business Key

A Primary Key is used to define the grain of the table, same way as a primary key normally works. For a staging and persistent staging load the primary key is used to differentiate between a new row and an update to an already loaded row. When generating table scripts from BimlFlex the defined primary key columns will be used to build the staging and persistent staging primary keys. The default behavior is to generate the Primary Key together with the `RowEffectiveFrom` column.

A source table can have multiple columns in the Primary Key definition.

A Business Key is the key describing the business entity stored in the table. It is used in Data Vault modelling to describe the Core Business Concept or Enterprise Wide Business Key candidate that will be used to model and load Hubs. The Business Key can be the same as the Primary Key or it can be derived from a combination of attributes. The Business Key is always a string representation so the default creation behavior in BimlFlex is to apply the `FlexToBk()` expression to convert the source column data types to a string. The import metadata function can derive a Business Key from either the Primary Keys or the Unique Constraints from the source.

Sample constraints from a Staging table create script to illustrate Primary Key and Business Key definitions:

```
-- Constraints
,CONSTRAINT [PK_AWLT_Address] PRIMARY KEY CLUSTERED
(
  [AddressID] Asc, [FlexRowEffectiveFromDate] ASC) WITH(PAD_INDEX = OFF,IGNORE_DUP_KEY = OFF) ON "default"
,CONSTRAINT [UK_AWLT_Address]  UNIQUE NONCLUSTERED
(
  [Address_BK] Asc, [FlexRowEffectiveFromDate] ASC) WITH(PAD_INDEX = OFF,IGNORE_DUP_KEY = OFF) ON "default"
)
```

An Alternate Business Key is used as a backup of the source Primary Key when the primary key definition needs to be changed in the modelling process. If there are Primary Key columns defined as derived columns that aren't persisted into the Persistent Staging layer

BimlFlex will use the Alternate Business Key instead.

Applying Data Type Mappings, expansion

Source system data types can be expanded to be more accommodating. This enhances resiliency in the load process when the source system data types change. The Data Type Mapping feature is described in a separate document and video

For the trial process, add the default Data Type Mappings to the `AWLT` record source

Defining Business Keys

Defining Business Keys to use for Data Vault modelling is a Business Analysis phase in the implementation. finding the correct Business Key definition relies on business process and source system knowledge as well as source system data profiling. For cross system key matching in Data Vault extensive analysis across all candidate sources. Defining the Business Key is a more straightforward exercise. For a given source table there can be only one Business Key column. For scenarios with multiple source keys the columns are concatenated with a separator character to build a single Business Key. This maps to the Business Key column in the Data Vault Hub tables and allows the Data Vault model to adhere to the pattern.

BimlFlex provides an expression to concatenate and separate columns into the Business Key using the `FlexToBk(Column1, Column2, Column 3)` function. This can be applied in SSIS for SSIS load patterns and in Sql for Sql patterns. The trial uses the SSIS pattern and will implement the expression in a derived column in the generated SSIS packages.

The concatenation character used with multiple columns is defined in the configuration using the `ConcatenatorBusinessKey` key. The default concatenation character is `~`.

BimlFlex also provides a set of shortcut codes for accessing specific data when building the Business Key. An example is the `@@rs` record source shortcut that injects the current record source code in to the Business Key. This is commonly used when there is key overlap between source system with different meaning. This can be when the same codes mean different things and, more commonly, when synthetic keys are used that are commonly reused, such as number sequences.

For the trial process, add the `@@rs` shortcut to all Business Keys that were created by the import metadata process. For the Address table that would be changing the `FlexToBk(AddressID)` to `FlexToBk(@@rs, AddressID)`

Defining Relationships

Source system relationships describe the metadata so that the correct load patterns can be used. For Data Vault, relationships help the Accelerator build out Link constructs between entities. When a metadata import is performed, BimlFlex will add any constraints defined in the source. This information is maintained in the `ReferenceTable` and `ReferenceColumnName` columns in the Columns sheet in the metadata. The chosen reference column must be a Primary Key in the related table. Relationships are also present in the Data Vault and Dimensional model to describe relationships between Data Vault entities such as Hubs and Satellites as well as Facts and Dimensions to define load patterns.

For the trial process, maintain the relationships between the source FK's and PK's. Review the relationship definitions in the columns tab.

Defining object related Metadata

The objects tab contains the tables from the `AdventureWorks LT` source system as well as any Data Warehouse tables created and accelerated. The metadata can be modelled and tweaked in several ways on the object and table level.

The main approaches are:

- Renaming columns and providing naming guidelines to the accelerator
- Tweaking the load pattern used by filtering, joining and grouping the data queried from the source.

Naming conventions

Using the naming override to define names in the Data Vault layer

`ObjectAlias` `ModelOverrideName` `ModelOverrideShortName` `ModelGrouping` `ModelObjectType`

Naming guidelines

Grouping entities TODO: Coming soon

Load Pattern Tweaks

`CreateSql`

Overrides the object creation script. Can be used to completely override the object creation. This is useful for views that are used in the Data Warehouse where the definition of these views are required to be maintained in the metadata. Since the automatic object creation is dynamic and accommodates any changes to the metadata it is recommended to only use the CreateSql functionality when needed.

In the trial process there are abstraction layer views for the dimensional model defined in the CreateSql metadata. This is used later in the process.

`OverrideSql`

Overrides the source Sql Select statement for the object. Can be used to completely override the select statement. This is useful for scenarios where the required select statement is too complex to be built automatically from the rest of the metadata. Since the automatic Sql creation is dynamic and accommodates any changes to the metadata it is recommended to only use the OverrideSql functionality when needed.

The OverrideSql functionality is not used in the trial process.

`JoinSql`

the `JoinSql` function allows the addition of join statements to the generated source query. This is useful for gathering additional data into the query, either for adding columns or for allowing additional filter conditions. A common use case is for Business Keys to be joined in when modelling a Data Vault on Business Keys rather than technical keys.

In the trial process there are 2 derived tables that use the JoinSql function to derive different types of addresses and maintaining them as separate entities in the staging layer and Data Vault.

`WhereSql`

The `WhereSql` function allows the addition of where statements to the generated source query. This is useful for filtering. Note that load parameters are added automatically to the Where clause when using the parameters function so there is no need to manually inject them in to the Where clause through the WhereSql functionality.

In the trial process there are filtering added to 2 derived tables to filter only certain types of addresses.

`GroupBySql`

The `GroupBySql` function allows the addition of Group By statements to the generated source query.

In the trial process the GroupBySql functionality is not used.

Defining columns related metadata

TODO: Coming Soon

Applying load parameters

TODO: Coming Soon

Applying Data Type Mappings

Supporting Videos

Supporting BimlFlex Documentation

BimlFlex User Guide

Data Type Mappings

Data Type Mappings is a BimlFlex feature that can expand the Data Types of the source to a larger data type that is more accommodating. This is done to accommodate changes in the source system without the need to update the Data Warehouse or the load process. The most common expansions are for short string representations that might be updated in the source. A Name field of 20 characters might be updated to 250 to accommodate longer customer names. An Integer might be updated to a Big Int when the source counter nears its max. By expanding incoming data it is possible to cater for these updates before they become a load issue.

Note that larger data types might require more database resources.

BimlFlex provides a rules engine for applying the expansions based on the Business Requirements. Based on the configuration BimlFlex can apply the expansions across all columns for a source and all tables and load patterns will take the new expanded type into consideration.

Detailed Steps

The following detailed steps walks through the application of Data Type Mappings for the trial source data.

Open and Refresh Metadata

Open the Excel Metadata Editor and click `Get All Entities` and `Get All Settings` to read all metadata from the repository.

Review Data Type Mappings

The `Data Type Mappings` sheet contains the BimlFlex Data Type Mappings that can be used to expand data types for incoming data. Should a specific mapping approach be required, tweak the mappings sheet to match and publish the updated information to the repository.

Apply Data Type Mappings

The Data Type Mappings are added to all columns for a specific `Record Source`. Choose the record source to apply to and click apply. The mappings will be added to all mappable columns. Once the mapping has been applied the mapped data type will be defined in the Data Type mappings column in the columns sheet. The original data types will be maintained in the column data type definition.

Creating table scripts from BimlStudio will use the applied data type mappings for the column definitions.

Building Business Keys for Data Vault

Supporting Videos

TODO: Coming Soon

Supporting BimlFlex Documentation

BimlFlex User Guide

Business Keys in Data Vault

TODO: Coming Soon

Detailed Steps

The following detailed steps walks through how to tweak Business Keys for the trial.

Open and Refresh Metadata

Open the Excel Metadata Editor and click `Get All Entities` and `Get All Settings` to read all metadata from the repository.

Review already generated Business Keys

The Import Metadata wizard created Business Keys from the Primary Keys of the source.

Update the Business Keys

Apply some updates to the keys

Accelerating the Raw Data Vault Layer

Supporting Videos

TODO: Coming Soon

Supporting BimlFlex Documentation

- Data Vault Accelerator

Data Vault Acceleration

Once the source and corresponding Data Vault modelling is done, creating a Raw Data Vault layer involves mapping source data to Data Vault constructs. BimlFlex provides acceleration of this process by technical analysis of the source data and its relationships to automatically create the required Data Vault structures and mapping the source data to the destination.

The accelerator creates a preview of the Data Vault artefacts so that the model can be reviewed and refined through the normal modelling and development process. Once the preview matches expectations it can be published to the metadata repository.

Once the metadata has been refreshed in BimlStudio and in the Excel based metadata management solution it can be built and implemented or further refined.

Detailed Steps

The following detailed steps walks through the acceleration of the Raw Data Vault Layer:

Configuring the Accelerator

To start the acceleration process, configure the accelerator to the record source and project used. For the trial this is the `AWLT` record source from the AdventureWorks LT database and the `LOAD_RDV` project.

Previewing the Acceleration result

Click the preview button to generate a preview of the Data Vault. In the BimlStudio logical view the corresponding tables will be visible in the preview schema. Using the schema visualization tool in the documentation tab the data vault schema can be visualized and reviewed.

Publishing the Acceleration result

TODO: Coming Soon

Refreshing the metadata in Excel to review the Data Vault

TODO: Coming Soon

Building the new solution

TODO: Coming Soon

Adding Business Data Vault helper and performance constructs

Supporting Videos

TODO: Coming Soon

Supporting BimlFlex Documentation

- Data Mart Templates

Adding Business Data Vault performance constructs

The Point in Time and Bridge table structures are used in Data Vault to make the Data Vault easier to query and to improve query performance.

- Point in Time, PIT, tables are used to create timelines for all changes in all or some Satellites attached to a business entity in a Hub
- Bridge tables are used to link business entities in Hubs through their link tables into easy to query constructs

Detailed Steps

The following detailed steps walks through adding Business Data Vault performance constructs

Adding Point in Time table metadata

TODO: Coming Soon

Adding Bridge table Metadata

TODO: Coming Soon

Building PIT and BRG Tables

TODO: Coming Soon

Creating PIT and BRG Stored Procedures

TODO: Coming Soon

Building the PIT and BRG load packages

TODO: Coming Soon

Dimensional Model from Data Vault

Supporting Videos

TODO: Coming Soon

Supporting BimlFlex Documentation

BimlFlex User Guide

Implementing the Dimensional Model from Data Vault

Based on the PIT and Bridge tables in the Data Vault layer a set of Fact and Dimensional views are created that are used to populate a dimensional model in a data mart in the presentation layer.

Detailed Steps

The following detailed steps walks through the creation of the Dimensional Model from Data Vault

Creating the source views for the dimensional model

TODO: Coming Soon

Creating the dimensional model metadata

TODO: Coming Soon

Mapping the model metadata

TODO: Coming Soon

Building the dimensional model SQL artefacts

TODO: Coming Soon

Building the dimensional model SSIS load project

TODO: Coming Soon

Using prepared trial metadata

Supporting Videos

TODO: Coming Soon

Supporting BimlFlex Documentation

TODO: Coming Soon

Using prepared trial metadata

The BimlFlex database contains preloaded trial metadata in the archive tables. It is possible to restore the trial metadata from any point in the trial process and start from there.

Detailed Steps

The following detailed steps walks through how to use the prepared trial metadata

running stored procedures to restore trial metadata from archive

The trial metadata is stored in the archive tables and uses the standard BimlFlex archive and snapshot functionality to restore metadata. The stored procedure that restores metadata is called

`[archive].[SetRollbackSnapshot]`

and is called through the following syntax

```
DECLARE @RC int
DECLARE @CustomerUID nvarchar(40)
DECLARE @VersionName nvarchar(50)
DECLARE @SnapshotName nvarchar(128)
DECLARE @UserName nvarchar(100)

-- TODO: Set parameter values here.

EXECUTE @RC = [archive].[SetRollbackSnapshot]
   @CustomerUID
  ,@VersionName
  ,@SnapshotName
  ,@UserName
GO
```

The following snapshots are available to restore:

NO	NAME	DESCRIPTION
1	Snapshot 1	Snapshot of metadata for source to staging load for AdventureWorks LT
2	Snapshot 2	Snapshot of metadata for source to staging to persistent staging load for AdventureWorks LT
3	Snapshot 3	Snapshot of modelled metadata before acceleration of Data Vault objects
4	Snapshot 4	Snapshot of metadata after acceleration of Data Vault objects

NO	NAME	DESCRIPTION
5	Snapshot 5	Snapshot of metadata after adding Bridge and PIT Data Vault objects
6	Snapshot 6	tba
7	Snapshot 7	tba
8	Snapshot 8	tba
9	Snapshot 9	tba
1o	Snapshot 10	tba