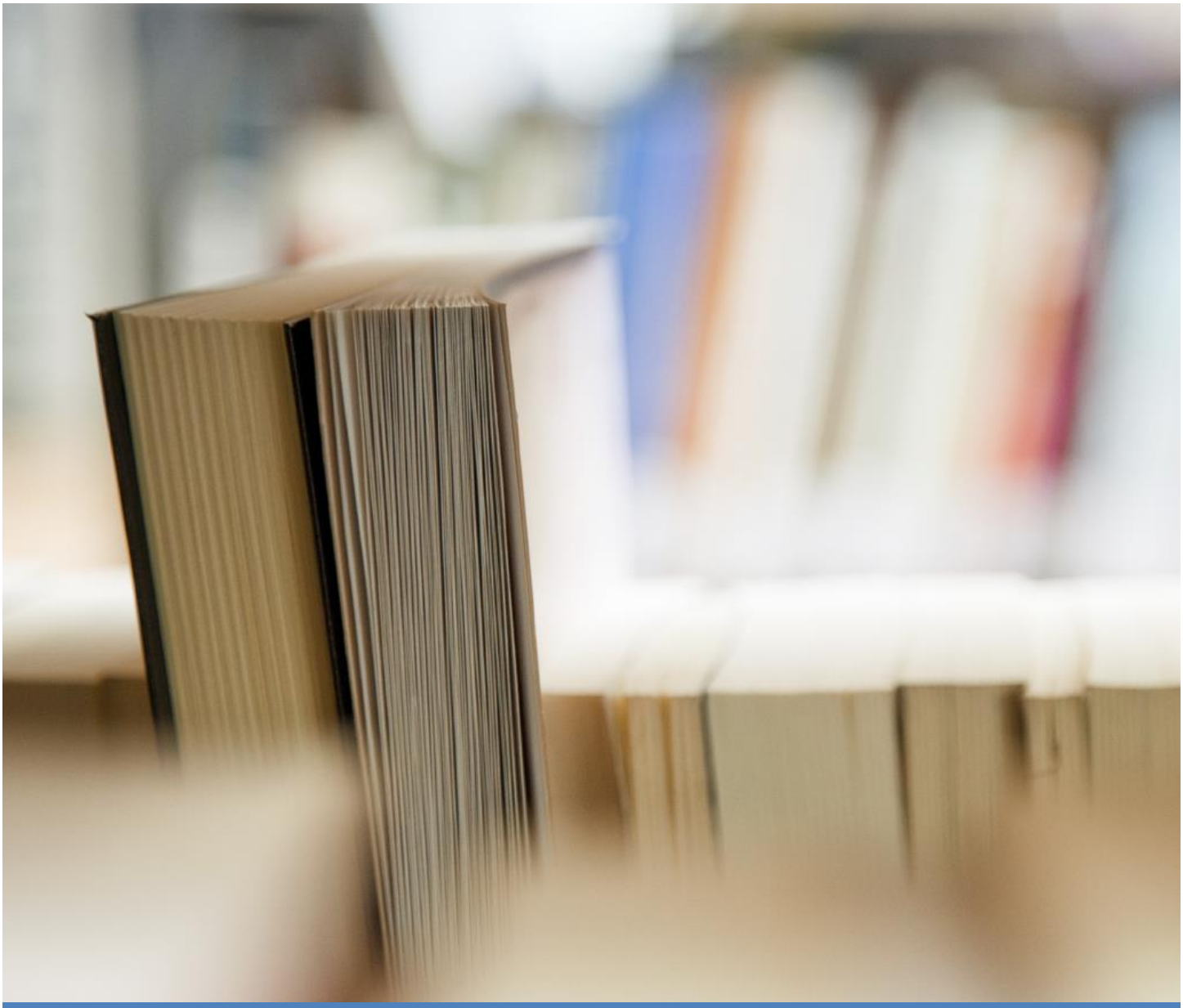


Dimodelo Architect User Guide

User Guide

Dimodelo Solutions 7/27/15

TABLE OF CONTENTS

DOCUMENT HISTORY

Version	Changes
1	Initial Version

INTRODUCTION

Dimodelo Architect is a Data Warehouse Automation tool.

Using Dimodelo Architect a developer can:

1. Capture the Data Warehouse and ETL **Design.**
2. **Generate** the Data Warehouse schema (and modifications) and ETL code.
3. Automatically **deploy** to multiple server environments (DEV,TEST,PROD etc).
4. **Execute** the ETL batch on the server and capture batch execution information.

Dimodelo Architect should be familiar to Microsoft developers because it is built as an addition to Visual Studio, the Microsoft Development Environment.

The best way to understand how Dimodelo Architect works is to:

1. Read the Overview below and the features page.
2. Watch the videos.
3. Download a trial edition.
4. Follow the tutorial.

OVERVIEW OF DEVELOPING A DATA WAREHOUSE USING DIMODELO ARCHITECT

To develop a Data Warehouse using Dimodelo Architect follow these steps:

1. Create a new Dimodelo Architect Project
2. Define one or more Connection Managers and Source Systems for the Data Warehouse. For each Connection Manager it is necessary to add additional information in a Source System record.
3. Design the Data Warehouse. A quick start method is to use the Star Schema Wizard to quickly identify Fact table, and define Dimensions for those Fact tables. Or, you may choose to just jump straight in, and start designing Staging tables, Dimensions, and Fact tables.
4. Once a set of staging tables, Dimension and Facts are defined, you can Generate the code. Generation creates both the DDL (T-SQL) to create/maintain the Staging and Data Warehouse databases, it also generates the SSIS packages that implement the ETL.
5. Generated code can then be deployed to a local or remote server.
6. Dimodelo Architect allows the user to Run an ETL batch, through either the Dimodelo Architect interface, or via Dimodelo Management Console on a server. You can find out more about executing a Batch in the Batch Management Guide.

Using this methods described above, you can iterate through a series of releases very quickly, continually refining the design. A Data Warehouse built through Dimodelo Architect is very flexible. It's just a matter of changing the design and re-generating. Once you are more familiar with Dimodelo Architect you will want to start working with more sophisticated functionality. This includes:

- Supporting multiple environments through Dimodelo Architect Project Configuration.
- Defining ETL Batch Workflows.
- Deploying to Servers and Scheduling ETL Batches.

- Advance design techniques including support for:
    - Conformed Dimensions and communication with other Data Warehouses.
    - Dimensions with Smart Keys.
    - Many to Many Bridge patterns.
    - Derived Staging tables.
    - Incorporating Custom DDL and SSIS packages into the project.

CREATE A NEW DIMODELO ARCHITECT PROJECT

This section describes how to create a new project.

- Open Dimodelo Architect.
- On the file menu Click **New>Project**.
- Select Dimodelo Project from the templates list.
- Give the project a **Name** and **Location**.
- Tick 'Create directory for solution'(recommended).
- Click Save.

CREATE A CONNECTION MANAGER

A Connection Manager defines the connection to a Source database, file etc.. Data is extracted from source system 'tables' into the Staging database. Dimodelo Architect supports many different types of Sources Systems including databases (Any database type that has an OLEDB driver, DB2, Oracle, SQL Server etc), Excel Files, Flat Files and XML files.

To define a new Database Connection manager do the following

1. Expand the **'Connection Managers'** Node in the Solution Explorer tool window.
2. Right click the **DB** folder and click **'Create New'**.
3. Enter a Name into the Connection Manager Name field. The Connection Manager names 'Staging' and 'Warehouse' are reserved and can't be used.
4. Click the **Configure** button.
5. Select a data source (e.g. **Microsoft SQL Server)** in the Data Source list, and then a Data Provider (e.g. '**.NET Framework Data Provider for OLE DB**') in the **Data Provider** dropdown. The dimodelo.desk.com site contains information about selecting the correct parameters for different types of Databases including Oracle.
6. Click **OK**.
7. Another dialog appears particular to the type of Data Provider selected. Fill in the details for you connection.
8. Click **Ok**.
9. For some providers (like Oracle) it is necessary to supply the **Owner/Schema** and **Password** details.
10. Click the **Save** icon in the Dimodelo Menu.
11. There should now be a *Name*.db node under Connection Managers>DB in the solution.

After you create a connection, you must also create a new Source System. The Source System contains a little more meta-data information about the Source System.

Important Note:

As you create, delete and change Connection Managers, DA will create, delete and change the equivalent connection strings in all DA Config files associated with the project via a Visual Studio project configuration. However if a Connection Manager is changed and it's connection string is modified, DA will only modify the equivalent connection string in the DA Config file associated with the currently active project configuration.

There is more specific information about Connections tips and tricks at our support site dimodelo.desk.com.

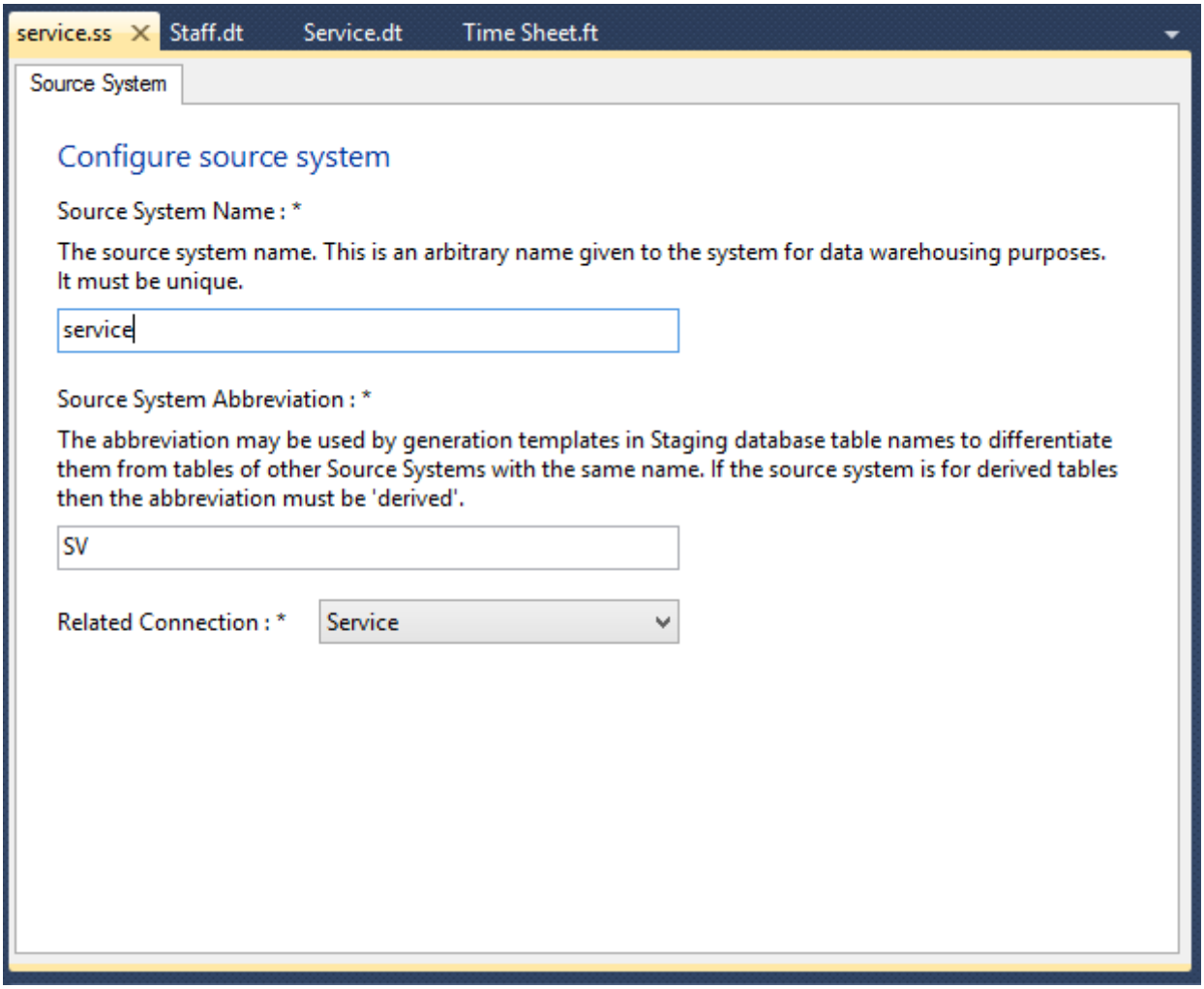
CREATE A SOURCE SYSTEM RECORD

In addition to a Connection Manager, a Source System record needs to be created. The Source System record contains additional information about the nature of the Source system.

The most important aspect of creating a Source System record is to define a 'Source System Abbreviation'. The 'Source System Abbreviation' is used to prefix table names in the Staging database. This ensures there is no naming conflicts in the Staging database between tables with the same name from different Source Systems.

Without a Source System, a Staging table can't be defined, and the Source System won't appear in the Sources tool window.

To Create a Source System, right click the Source Systems folder in the Solution explorer and click Create New. The following dialog appears.



- **Source System Name**. Give the source system a unique name. This is the name that appears in the Solution Explorer.

- **Source System Abbreviation**. Give the Source System a two or three letter abbreviation that is unique within the project. The 'Source System Abbreviation' is used to prefix table names in the Staging database. This ensures there is no naming conflict when staging tables with the same name from more than one Source System.

- **Connection**. Associate the Source System with one of the existing Connection Managers. This is mandatory.

In the future Source System records will be removed, and the information combined into the Connection information.

STAGING TABLE

You must define a Staging table for Source data before the data can be incorporated into the Data Warehouse. Dimodelo Architect allows you to import the metadata about columns and data types of a source table to quickly define a Staging table.

When defining a Staging table extract, you have the choice to either:

1. Map Staging columns to a Source table column.
2. Map Staging Columns to the columns of query against the Source System.

Create a Derived Staging table, by uses other Staging tables as the Source. For more information about Derived Staging tables see the Creating a derived Staging table section.

DA also implements two Patterns for Extracts. The two patterns are either Full Extract, or a Date Range Extract. More information can be found in the Patterns Guide.

To create a Staging table, you must first define a Connection Manager and Source System of the Source table(s) in the DA project. A Staging table can only stage data from a single Source System, but can stage data from multiple tables in that Source system, although this is not a recommended approach (according to Kimball et al) in most cases. If you do need to combine data from more than one Source System in the Staging database, to perhaps aggregate or allocate, then you can you a Derived Staging Table.

CREATING STAGING TABLES

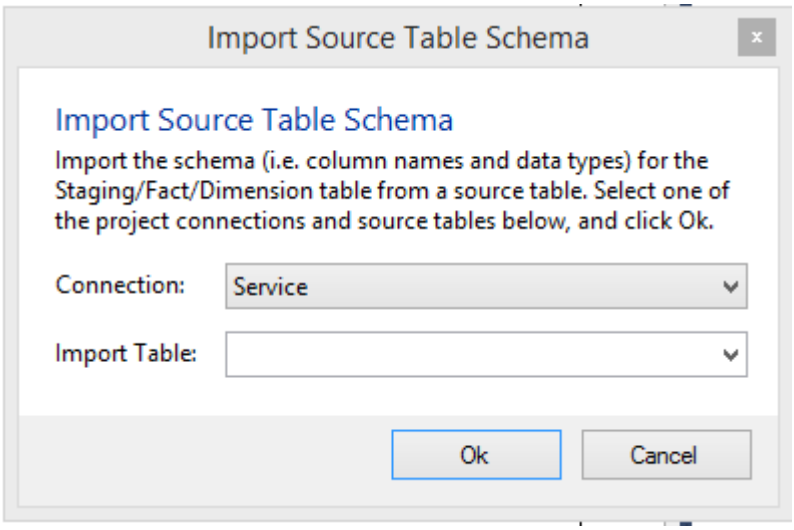
To create the staging table:

- Right click the **Staging** folder in the Solution Explorer tool window, and click **'Create New'**, a new Staging editor panel appears.
- Select the Source System of the Staging table in the Source System field. The Source System is used to define the name of the Staging table in the Staging database. The table name in the Staging database will be *Source SystemAbbreviation_Staging Table Name.*
- Enter a **Staging Table Name**. The name is used to define the name of the table in the Staging database. The Staging Table Name is used to define the name of the Staging table in the Staging database. The table name in the Staging database will be *Source System Abbreviation_Staging Table Name*.
- The **Derived** check box must be ticked for a derived staging table. This affects the Name of the table in the Staging database. The database table name is prefixed with *Derived* rather than the Source System abbreviation. For more information about Derived staging tables read the 'Creating a Derived Staging table' section.
- Enter a Description for the Staging table. This is optional. The Description is used as an overview for the table in the generated Documentation.

STAGING COLUMNS

The quickest way to define Staging table is to import its column definition from a Source table. If there isn't a matching Source table, then you can manually define the columns. To import a Source table do the following:

- Click on the Staging Columns tab.
- Click the **Import Schema** button.



- Select the desired Connection Manager in the Connection dropdown.
- Select the Source table or view in the **Import Table** field.
- Click **Ok**. The columns and data types of the source table are imported into the Staging table design. You can see the columns on the Staging Columns tab.

You can also:

- **Modify a Column Name**. Tip… Only modify the column names after you have defined an Extract. That way Dimodelo Architect will automatically match the columns in the Source table with columns in the Staging table based on name, when you edit the mapping in the Extract. After that, you can come back and change the name, DA recognises the name change. The Column Name becomes the name of the column in the Staging Database.
- **Modify a Column Data Type**. This may mean you need to convert the data type of the Source column in the Extract definition using an Expression.
- **Add a Description.** This is optional. The Description is used in the generated Documentation.
- **Delete a Column**. Just click the Delete link at the end of the row.
- **Add a Column**. You will need to add a mapping in the Extract if you add a column.

Dimodelo Architect will Sync the columns when the Data Warehouse is deployed.

ADVANCED TAB

- **Staging Tags**. Some Extract patterns require custom design data to operate correctly. See the patterns guide for more information on applicable custom design/meta data.

DEFINING EXTRACTS

To define an Extract select the **Add** button on the **Extract** tab. The Extract dialog appears. It's possible to define multiple Extract for a table, but not typical.

OVERVIEW TAB

- **Extract Name**. The Extract Name is important as it is used to uniquely identify the Extract and is used by the standard generation templates to name the Extract's SSIS Package. E.g. Extract_*Extract_Name*.dtsx. The name is defaulted to *source system abbreviation_Staging table name* Make sure the name matches a task in the Workflow. See more about Workflows in Batch Management Facility document.
- **Extract Pattern**. Select the pattern that will be used to determine the type of code that will be generated for this Extract. Additional Patterns can be defined in the Reference Data/Patterns.rf document with a category of 'Extract'. The standard Extract patterns include:
  - **Full.** The Staging table is truncated prior to extract, and all data from the Source (Table or Query) is extracted.
  - **Date Range.** Usually used for the source of high volume Transaction Fact tables, where only a range of data (based on date) is extracted from the source. See the next section for more details.
- **Extract Overview.** A description of the extract for documentation purposes.

EXTRACT PATTERNS

TRUNCATE

The Truncate Extract pattern is very simple. The Staging table is truncated prior to the Extract being executed. The data from the source table is extracted in full. There are no custom meta data tags required for this pattern.

DATE RANGE

The date range pattern is used to extract a subset of the source data based on a date range. The end of the date range is the current batch execution date. The start of the date range is a number of days prior to the current batch effective date. The number of days is specified by a custom meta data tag called 'DIL_Extract_Period_Days' added to the tags table on the advanced tab of the Extract dialog. Set the value of this tag to the number of days.

You must also tag one source column as the Date attribute that is used to determine if a source row is extracted based on the value of the 'DIL_Extract_Period_Days' tag of the Target Fact table. To identify this source column, edit it through the Mapping tab of the Extract dialog and add a tag named 'DIL_Date_Attribute'. Set the value to 'DIL_Date_Attribute' also. The column must be of type datetime or smalldatetime.

SOURCE TAB

Defines the source table/query of the Extract.

- **Connection**. First select the connection which contains the table(s) that is the source of the Extract.
- **Source Table View**. Select the name of the table/View that is the source of the Extract.
- **Source Query**. Select the Source Query option if the source of the Extract is a query rather than a single table. Enter your query in the source query text box. This query should be written as if it was executed against the Source database. Don't specify a database Server or database name in the Query. The connection manager handles this, and it may change depending on the configuration the SSIS package is generated with. A Query can't contain an ORDER BY, DECLARE or CTE clauses, as the query is wrapped as a sub query when generated in the Extract SSIS package. If the query is invalid, and you click the mapping tab, an error message is displayed.

MAPPING TAB

The Mapping tab defines the column mappings between the source columns and the staging table columns. The mappings may also be expressions.

- **Staging Column**. The Staging Column is the name of a Staging table column. The Staging Column is populated from the list of columns defined for the Staging table.
- **Source Column Mapping**. The Source Column Mapping is the name of the Source column that maps to the Staging table column. This field is auto matched based on name, after a source is selected in the Source tab.
- **Double click the column or select and click Edit mapping** to change the setting for the column, define an expression mapping, and add a description to the mapping for documentation purposes. Click the button and the Extract Mapping dialog appears.
    - **Source Column Mapping.** Select a Source Column mapping for the Staging column from the drop down.
    - **Expression.** Write an Expression for the mapping. Note the Expression can only reference source columns (not staging columns). The Expression is T-SQL, and must be a valid SQL expression when executed against the Source database. It should be written as if the expression will be part of a Select clause. E.g. SELECT *expression* FROM Source Table.
    - **Extract Mapping Description.** A description of the mapping for documentation purposes. Non Mandatory.
    - **Extract Mapping Tags.** Define custom meta Data for the mapping particular to the chosen Extract pattern.
    - **Data Quality Screening**. Future functionality.

ADVANCED TAB

- **Extract Tags**. Define custom meta Data for the Extract.

CREATING A DERIVED STAGING TABLE

What is a derived staging table?

It's a table created in your Staging database that derives its data from other already existing Staging tables. It's useful for creating aggregations, or allocations of data prior to the transform and load process into your Data Warehouse. It can also be useful for tables that don't have any other source other than an SQL Query which creates the data.

To create a Derived Staging table in Dimodelo Architect, do the following:

1. Create a Connection manager to your Staging Database. Name the Connection Manager anything other than 'Staging' which is reserved.

2. Create a Source System for your Staging Database. The Source System Abbreviation must be **Derived**. Again the Source system name can be anything but 'Staging'.

3. Create your Staging table. When defining the Staging table tick the Derived tick box on the Staging table tab of the Staging table editor. This ensures the Staging table is generated with the Derived prefix in the Staging database. Also the SSIS package is prefixed with Extract_Derived instead of Extract.

4. In the ETL Extract, set the source connection to the staging connection you just created, and define a query that selects data from one or more existing staging tables. The query contains the logic of aggregation, allocation etc. Tip. Give the Extract an Extract Name prefixed with *Derived*. When defining the ETL Batch Workflow later you can create a phase that executes after all the regular Staging table Extracts have been executed, and processes all extracts that start with the name Derived.

DIMENSIONS

CREATING A DIMENSION

To create a dimension, right click the 'Dimensions' folder (or one of its sub folders) and select Create New.

There are two ways to create a dimension. By far the quickest way is to import the Meta Data from a Source System, and then edit as required. If the Source Meta data is not available, then you can manually enter the information.

DIMENSION TAB

- **Dimension Name**. The Dimension Name is the name of your table in your data warehouse relational database. Spaces in the name will be converted to _ automatically when you save. The prefix for the Dimension table name can be set in the config file.
- **Dimension Friendly Name**. This is the friendly name the user sees in the Solution explorer. In the future, if DA was to generate a cube, it would be the name of the Dimension in the Cube. Spaces are not replaced in this name.
- **Dimension Description**. A description of the contents of the dimension. optional.
- **Reference.** This is a free form text you can use to reference the dimension design back to another design document, perhaps a business requirements document.

DIMENSION ATTRIBUTES TAB

You can add, delete and modify dimension attributes through the dimension's Attribute tab.

IMPORT

Use the **Import Schema** button on the Dimension Attributes tab to import the Meta Data for the Dimension from a Source System table.

To Import the Meta Data.

- On the **Dimension Attributes** tab, click the **Import Schema** button.
- Select the **Connection** where the source table resides.
- Select the **Table** and press OK.

The dimension attributes will be populate with the column names and data types from the source table. Some connection types don't return column data types. In this case the data types are defaulted to varchar(50).

TO ADD AN ATTRIBUTE

Click in the empty row at the end of the list.

- **Attribute Name**. Enter the attribute name. Spaces are replaced with _ when the dimension is saved. The Name must be unique within the dimension. Dimodelo Architect will validate the name. If a name is specified that violates SQL Server naming restrictions, a red ! will be displayed on the row.

- **Data Type**. Select a data type from the dropdown list, or just type the data type. The data type will be validated to ensure it is a valid SQL Server data type when the data type field loses focus. If the data type is invalid, a dialog box will display, and the field cleared.

- **Column Type**. Choose the column type for the attribute. If the attribute is part of the Natural/Business key of the dimension, then select 'Business Key'. The business key can be a composite key. That is, more than one attribute can have a type of Business Key. Normally the business key is what your ETL process will use to look up the dimension to identify if source rows already exist in the dimension. It is the primary key of the source table.

- **SCD Type.** SCD Type is an acronym for Slowly Changing Dimension Type. The list of types correspond to the SCD types set out by Ralph Kimball. When the Attribute type is 'Attribute' this field is enabled and is required. Select the appropriate SCD type for the attribute. The SCD type selected will affect how the DDL is generated for the attribute (using the supplied templates). Type 3 and 6 SCD Type attribute will have additional <attribute name>_Prev columns created for them. If you don't understand SCD types then read our 'Data Warehouse and Business Intelligence Concepts – Guide'.

- **Description**. Add a description of the attribute. Generally the description should make sense to an end user. Descriptions are used in the generated Data Dictionary.

- **Parent.** The parent indicator is used to indicate that the attribute is the natural key of the corresponding parent row in a parent-child hierarchy. That is, each row keeps its parent identifier in this attribute column. Parent identifiers can be composite keys, so more than 1 attribute can be marked as a Parent. The presence of a Parent indicator on any attribute will cause a Parent Surrogate Key to be generated in the DDL using the supplied DDL templates. Only a single Parent surrogate key is produced regardless of how many attributes are marked as Parent. Parent logic is added to the ETL Transformation if Parent is ticked.

- **Reference.** This is a free form text you can use to reference the attribute back to another design document, or perhaps a report the attribute appears on.

TO DELETE AN ATTRIBUTE

Select the attribute row and click the delete key, or click the Delete Attribute(s) button. You can select multiple rows.

TO CHANGE AN ATTRIBUTE

Simply find the attribute in the list, click in the field you want to change, and change as desired. Attribute name changes are generated in the DDL as non-destructive rename script. Changes are only made permanent after the document is saved.

ADVANCED TAB

- **Custom Surrogate Key Name**. This field allows you to specify a custom name for the surrogate key of this dimension. This is useful for in a conforming dimension scenario. See section **Error! eference source not found.** for more details. The standard DDL generation templates use the indicator and name to generate the correct surrogate key column name in the DDL.

- **Custom Surrogate Data Type**. This field allows you to specify a custom data type for the surrogate key of this dimension. This is useful for in a conforming dimension scenario. See section Conformed Dimensions section for more details. The standard DDL generation templates use the indicator and data type to generate the correct surrogate key column data type in the DDL.

- **Surrogate Key is Auto Generated**. Un-tick this option if the ETL manages surrogate keys, instead of the database auto generating surrogate keys. The standard DDL templates use the indicator to set IDENTITY on dimension keys. This should be used in conjunction with the Auto Generate Key toggle on the advanced tab of the ETL dialog.

- **Generate DDL**. Un-tick this option if you don't want Dimodelo Architect to generate the DDL for this table.

- **Conformed**. Meta data indicating the Dimension is a conformed Dimension from another Warehouse. This can be used in the Generation step to generate Conforming code. Currently this property is not used by the standard ETL generation templates.

- **Dimension Tags.** Custom Meta Data for the Dimension. Tag the dimension with a tag Name and Tag Value. Some Dimension patterns require custom metadata. See the Patterns guide for more information about the custom meta data required by each pattern.

CUSTOM MANAGEMENT COLUMNS

What are Management Columns? Management columns are columns that are added to your Fact and Dimension tables to help manage the data, during ETL. Management Columns include things like:

- Batch Identifier.

- Row Status (Current - Not Current).

- Row Effective Dates.

- Checksum (if you use checksums in your ETL)

Dimodelo Architect allows you to specify Custom management columns on an individual Fact/Dimension basis. A set of Standard Management Columns is specified for all Fact/Dimension tables. See section Standard Management Columns for an explanation of Standard Management Columns.

Custom Management columns can be used for Fact and Dimensions that have custom ETL that requires additional management columns for its operation.

Custom Management Columns can be added, changed and deleted through the Management Columns tab > Custom Management Columns table in the Fact and Dimension editors.

DEFINING DIMENSION TRANSFORMS

To define a Transform & Load process open the Dimension document of the Dimension table, and select the **Add** button on the **ETL** tab. The Transform & Load dialog appears.

OVERVIEW TAB

- **Transformation Name**. The Logical Data Map name will default to the name of the Dimension table. The Name is used to uniquely identify the Transform and is used by the standard generation templates to name the Transform stored procedure. The name must be unique across all Dimensions and Facts.
- **ETL Pattern**. Select the code pattern that will be used as the basis for the generated code for this Transform. Additional Patterns can be defined in the Reference Data/Patterns.rf document with a category of 'Transform'. The standard Transform patterns include:
    - **Type 1 and 2 Dimension.** This pattern caters for both Type 1 and Type 2 Slowly changing dimension attributes in the same dimension.
- **Transform Overview.** A description of the Transform for documentation purposes.

STAGING SOURCES

Defines the Staging tables and columns that are the source of the transform.

The staging source can be a single primary staging table or the combination of a primary staging table and multiple secondary staging tables. The primary Staging table should be the staging table that has the same grain as your destination Dimension table.

PRIMARY STAGING TABLE

- **Primary Staging Table.** Select the Staging table that will be the primary source for your transform. The primary source will have the same grain as the destination Dimension.
- **Staging Table Column List**. This list is automatically populated with all the fields from the Primary Staging table. Delete any columns that are not required.
- **Edit Button**. Click this button to specify further details for the Primary Staging Table. The Source Staging Table dialog appears.
    - Select the Primary Staging Table columns that are required for the transform.
    - **Filter**. In some cases, you may want to filter the data returned for the primary Staging table in the Transform process. Define the Filter, using an Expression. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.* See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns. The filter is applied in the Where clause of the generated Source Query.
    - **Tag Table**. Define custom meta data for the Primary Staging table.

SECONDARY STAGING TABLES

If data is required from another Staging table in the transform, define the table in the Secondary Staging Tables table, and how it joins to the primary (or other secondary).

- **Add Join Button**. Click this button to specify details for the Secondary Staging Table. The Source Staging Table dialog appears.
    - **Staging Table Name.** Select the Secondary Staging table from the drop down.
    - **Join On**. Write an expression describing how the Secondary table joins to either the Primary Staging table, or other Secondary Staging tables. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.*See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns. The Join on Expression is used in the LEFT JOIN ON clause of the generated Source Query. The Generated Source query is generated as a LEFT JOIN to the Secondary Staging table. If you require a JOIN, then use a '*Secondary Staging table key* IS NOT NULL' filter, to eliminate rows where there is no matching Secondary Staging table record.
    - **Filter**. In some cases, you may want to filter the data returned for the Secondary Staging table in the Transform process. Define the Filter, using an Expression. The filter is added to the Where clause of the generated Source query. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.* See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns.
    - **Tag Table**. Define custom meta data for the Primary Staging table.

MATCH ON

The Match On tab defines the columns of the Staging source tables that match to the business key(s) of the target Fact/Dimension for the purposes of identifying new and deleted source records. In the standard generated code, where a match is found, further comparison is made based on checksums, to determine if a change has been made to the target row.

There are 2 ways to define the match:

1. Use the Match button below the bottom right of the table to automatically match Staging to Target Business key based on Name.

2. Select the Match manually.

- **Staging Table**. Select the Staging table that contains the Staging column.

- **Staging Column**. Select the matching Staging Column.

- **… Button**. Click the … button if it is necessary to define an expression on columns of the Staging source to match to the target business key. The expression language is T-SQL. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.* See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns. The expression should be written as if it were in the SELECT list of a T-SQL Query against the Staging database. E.g. Select *expression* FROM Staging table.

- **Filter By**. In Rare cases, perhaps when you have defined multiple transforms on a Dimension, you may need to filter the Dimension table, to match the incoming Staging rows for that transformation. You can define the filter, by specifying a Where clause on the Dimension table.

The Filter is a T-SQL expression written as if it were part of a Where clause e.g. SELECT * FROM Dimension WHERE *Expression*. Reference Dimension table columns by Name.

TRANSFORMATION TAB

The transformation tab allows you to map Staging table columns to Attributes during the transformation.

There are 2 ways to define the mapping:

1. Use the **Auto Match** link below the bottom right of the table to automatically match Staging to Target Attributes based on Name.

2. Select the Mapping manually.

    - **Double click a row or click Edit Mapping button** to change the mapping for the column, define an expression mapping, and add a description to the mapping for documentation purposes. Click the button and the Transform Mapping dialog appears.

        - **Staging Column Mapping.** Select the Staging table and Staging Column mapping for the Target Attribute from the drop downs.

        - **Expression.** Write an expression for the mapping. Note the expression can only reference Staging columns. The expression language is T-SQL. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.* See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns. The expression should be written as if it were in the SELECT list of a T-SQL Query against the Staging database. E.g. SELECT *expression* FROM Staging_Table.

        - **Transform Mapping Description.** A description of the mapping for documentation purposes. Non Mandatory.

        - **Transform Mapping Tags.** Define custom meta Data for the mapping.

ADVANCED TAB

- **Transform Tags**. Define custom Design/Meta Data for the Transform.

Auto Generate Key. If the surrogate key for the Dimension is a 'smart' key (like 20100616 for a Calendar dimension), select false, and then specify the source Staging column that contains the 'smart' key. See the

- Defining a Dimension with Smart Keys section for more information.

DEFINING A DIMENSION WITH SMART KEYS

While conventional wisdom, and Kimball say that using meaningful identifiers for surrogate keys is a bad idea, there are some exceptions where it makes sense.

Typical examples include:

- Calendar. An example of a Calendar smart key is - year,month,day - 20110721. A datetime value can be converted to this date format using this statement convert(char(8),*date*, 112).
- Time (smart key example - hour, minute,second- 130159).

Dimensions which use 'smart' keys (i.e some kind of meaningful code as the key) are useful in your ETL because they do not require a dimension look up when loading a Fact table. They can also simplify SQL queries against your Data Warehouse.

In Dimodelo Architect you need to do 3 things to define the use of smart keys.

- On the advanced tab of the Dimension document, set the 'Custom Surrogate Key Name' to the name of the Smart Key in the Dimension.
- Un-tick the Surrogate Key is Auto Generated? check box.
- In the ETL dialog box for the Dimension, on the advanced tab, Set 'Auto Generate Key' to false, and select the column in the source Staging table that will be the source for your Smart Key. Only a single column can be used, so you may need to do a transform in your Staging table extract if there are multiple columns that make up the Smart Key.

A column cannot be both a business key and a surrogate key, so don't add the smart key column as an attribute of the Dimension. This will cause errors in code generation.

USING DIMODELO ARCHITECT WITH EXISTING DIMENSIONS FROM ANOTHER DATA WAREHOUSE

If you are a proponent of the Ralph Kimball school of Data Warehouse design, and the Data Warehouse Bus Architecture, you will subscribe to the concept of conformed dimensions.

Put simply, the concept of conformed dimensions states that Data Marts within an Organisation should share common dimensions so that cross Data Mart/Enterprise/Business process analysis can be delivered.

To support this concept Dimodelo Architect contains a number of advanced features.

To conform with an external dimension in another Data Warehouse/Mart, do the following.

- Define the external Dimension as an internal Dimension in the DA controlled Data Warehouse. This is so you can associate fact tables with it. When defining the Dimension, it is only necessary to define the Business Keys of the Dimension in the design.
- You will need to set the dimension to use a Smart Key (see the

- Defining a Dimension with Smart Keys section), and map the surrogate key of the external Dimension to the smart key of the internal dimension.
- If the external dimension uses a different standard for Surrogate key names and data types then on the advanced tab you will need to specify a custom surrogate key, and custom surrogate key data type.
- Set up a Connection Manager and Source system for the external Data Warehouse/Mart.
- Define a staging table for the external Dimension. Again only the business keys and surrogate key of the external dimension are required. Source the data from the external Data Warehouse/Mart.
- Define the ETL for the internal Dimension mapping only the business keys and Surrogate key as the smart key.

With this setup, a copy of the external Dimension is brought into the Data Warehouse. The ETL must be scheduled so the external Dimension is updated in the external Data Warehouse/Mart prior to the ETL that runs to stage the external Dimension. The Dimension in the Data Warehouse will have the same business keys and surrogate keys as the external Dimension, so when the Fact ETL is processed, Facts are associated with members in the Dimension with the same surrogate keys as in the External Dimension.

When defining an SSAS Cube or tabular model add the Fact from the Data Warehouse and the Dimension from the External Data Warehouse/Data Mart to the model. Associate them via the dimension surrogate key, which should now match.

FACT TABLES

CREATING A FACT TABLE

To create a Fact table, right click the 'Fact Tables' folder (or one of its sub folders) and select Create New.

There are two ways to create a Fact Table. By far the quickest way is to import the Meta Data from a Source System, and then edit as required. If the Source Meta data is not available, then you can manually enter the information.

FACT TABLE TAB

- **Fact Table Name**. The Fact Table Name is the name of your table in your data warehouse relational database. Spaces in the name will be converted to _ automatically when you save. The prefix for the table name can be set in the configuration utility.
- **Fact Friendly Name**. This is the Measure Group Name the user would see for the Fact Table through a Cube browser. Spaces are not replaced in this name.
- **Fact Table Description**. A description of the contents of the Fact Table. Optional. This is used when documentation is generated.
- **Reference.** This is a free form text you can use to reference the dimension design back to another design document, perhaps a business requirements document.

FACT TABLE MEASURES TAB

IMPORT SCHEMA

To Import the Meta Data.

1. On the Fact Table Measures tab, click the **Import Schema** button.
2. Select the **Connection** where the source table resides.
3. Select the **Table** and press OK.

The Fact Table Measures will be populated with the column names and data types from the source table. Some connection types don't return column data types.

DEFINE MEASURES

On this tab you need to define both the Business Keys (Natural Keys) and Measures of the Fact. All other columns that may have been imported from a Source table should be deleted. Columns used to lookup dimension don't need to be listed in this table.

- **Measure Name**. Enter the measure name. Spaces are replaced with _ when the Fact Table is saved. The Name must be unique within the set of Measures for the Fact.
- **Data Type**. Select a data type from the dropdown list, or just type the data type. The data type will be validated to ensure it is a valid SQL Server data type when the data type field loses focus. If the data type is invalid, a dialog box will display, and the field cleared.
- **Description**. Add a description of the measure. Generally the description should make sense to an end user. Descriptions are used in Data Dictionary generation.
- **Reference.** This is a free form text you can use to reference the measure back to another design document, or perhaps a report the measure appears on.
- **Business Key.** Tick the row(s) that represent the Fact Table's business/natural key. This is used in the definition of the Transform/Load process for the table.

TO DELETE A MEASURE

Click the delete link next to the measure in the Measures table. Changes are only made permanent after the document is saved.

TO CHANGE A MEASURE

Select the measure row and tap the delete key, or click the Delete Attribute(s) button. You can select multiple rows.

ASSOCIATING DIMENSIONS

In order to complete the star schema of the fact table you must associate it with one or more of the dimensions in the project. To associate a Fact Table to a dimension, use the **Dimensionality tab** of the Fact Table.

Select the dimension you want to associate to the fact table in the drop down list in the Dimension column. Click on another cell to ensure the association is recognized. If you intend to associate the dimension as a role play dimension, then give it a role play name. A dimension can be associated to a fact table multiple times, but only once without a role play name. That is, you can associate a dimension to a fact without a role play name once, and every other association must have a role play name and each role play name must be different.

Changes are not saved until the Fact table is saved.

CUSTOM MANAGEMENT COLUMNS

What are Management Columns? Management columns are columns that are added to your Fact and Dimension tables to help manage the data, during ETL. Management Columns include things like:

- Batch Identifier.
- Row Status (Current - Not Current).
- Row Effective Dates.
- Checksum (if you use checksums in your ETL)

Dimodelo Architect allows you to specify Custom management columns on an individual Fact/Dimension basis. A set of Standard Management Columns is specified for all Fact/Dimension tables. See the Standard Management Columns section for an explanation of Standard Management Columns.

Custom Management columns can be used for Fact and Dimensions that have custom ETL that requires additional management columns for its operation.

Custom Management Columns can be added, changed and deleted through the Management Columns tab > Custom Management Columns table in the Fact and Dimensions editor.

DEFINING FACT TRANSFORMS

To define a Transform & Load process open the Fact document of the Fact table, and select the **Add** button on the **ETL** tab. The Transform & Load dialog appears.

OVERVIEW TAB

- **Transformation Name**. The transformation name will default to the name of the Fact table. The Name is used to uniquely identify the Transform and is used by the standard generation templates to name the Transform stored procedure. The name must be unique across all Dimensions and Facts.
- **ETL Pattern**. Select the code pattern that will be used as the basis for the generated code for this Transform. Additional Patterns can be defined in the Reference Data/Patterns.rf document with a category of 'Transform'. See the Patterns guide for more information about the ETL Patterns.
- **Transform Overview.** A description of the Transform for documentation purposes.

STAGING SOURCES

Defines the Staging tables and columns that are the source of the transform.

The staging source can be a single primary staging table or the combination of a primary staging table and multiple secondary staging tables. The primary Staging table should be the staging table that has the same grain as your destination Dimension table.

PRIMARY STAGING TABLE

- **Primary Staging Table.** Select the Staging table that will be the primary source for your transform. The primary source will have the same grain as the destination Dimension.
- **Staging Table Column List**. This list is automatically populated with all the fields from the Primary Staging table. Delete any columns that are not required.
- **Edit Button**. Click this button to specify further details for the Primary Staging Table. The Source Staging Table dialog appears.
    - Select the Primary Staging Table columns that are required for the transform.
    - **Filter**. In some cases, you may want to filter the data returned for the primary Staging table in the Transform process. Define the Filter, using an Expression. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.* See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns. The filter is applied in the Where clause of the generated Source Query.
    - **Tag Table**. Define custom meta data for the Primary Staging table.

SECONDARY STAGING TABLES

If data is required from another Staging table in the transform, define the table in the Secondary Staging Tables table, and how it joins to the primary (or other secondary).

- **Add Join Button**. Click this button to specify details for the Secondary Staging Table. The Source Staging Table dialog appears.
    - **Staging Table Name.** Select the Secondary Staging table from the drop down.
    - **Join On**. Write an expression describing how the Secondary table joins to either the Primary Staging table, or other Secondary Staging tables. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.*See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns. The Join on Expression is used in the LEFT JOIN ON clause of the generated Source Query. The Generated Source query is generated as a LEFT JOIN to the Secondary Staging table. If you require a JOIN, then use a '*Secondary Staging table key* IS NOT NULL' filter, to eliminate rows where there is no matching Secondary Staging table record.
    - **Filter**. In some cases, you may want to filter the data returned for the Secondary Staging table in the Transform process. Define the Filter, using an Expression. The filter is added to the Where clause of the generated Source query. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.* See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns.
    - **Tag Table**. Define custom meta data for the Primary Staging table.

- **Tag Table**. Define custom meta data for the Primary Staging table.

MATCH ON

The Match On tab defines the columns of the Staging source tables that match to the business key(s) of the target Fact/Dimension for the purposes of identifying new and deleted source records. In the standard generated code, where a match is found, further comparison is made based on checksums, to determine if a change has been made to the target row.

There are 2 ways to define the match:

3. Use the Match button below the bottom right of the table to automatically match Staging to Target Business key based on Name.

4. Select the Match manually.

- **Staging Table**. Select the Staging table that contains the Staging column.
- **Staging Column**. Select the matching Staging Column.
- **… Button**. Click the … button if it is necessary to define an expression on columns of the Staging source to match to the target business key. The expression language is SSIS. An SSIS expression editor dialog appears. You can drag and drop column names and functions etc.
- **Filter By**. In Rare cases, perhaps when you have defined multiple transforms on a Dimension, you may need to filter the Dimension table, to match the incoming Staging rows for that transformation. You can define the filter, by specifying a Where clause on the Dimension table. The Filter is a T-SQL expression written as if it were part of a Where clause e.g. SELECT * FROM Dimension WHERE *Expression*. Reference Dimension table columns by Name.

DIMENSION LOOKUPS

Use this tab to define how the Transform should look-up the corresponding member in the associated Dimension based on Staging columns.

- **Double click a row or click the Edit Mapping Button** to open the Dimension Lookup.
    - **Where Table.**

      There are 2 ways to define the lookup using a where statement.

        - Use the match button at the bottom right of the table to automatically match the Staging to target Dimension business key columns.
        - Define the match manually.
            - **Business Key.** The Business Key column is populated from the list of business keys for the Dimension.
            - **Staging Table.** Select the Staging Table that contains the column that maps to the Dimension Business Key.
            - **Staging Column.** Select the Staging column that maps to the Dimension Business Key.
            - **Expression and … Button**. Define an expression using Staging columns that maps to the Dimension Business Key. The expression language is the SSIS Expression Language.

- **Join On**. In some cases it is necessary to define a Join On expression to join the Staging columns to the Dimension table columns. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.* See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns.

- **Advanced Tab**

  - **Filter**. In some cases you might want to filter the Dimension prior to applying the lookup. Define the where clause for filtering here. When referencing Staging columns in a T-SQL Expression, use the following convention *Source_System_Abbrev_ Table_Name.Column_Name.* See the Expression Language section for details of how to refer to Staging and Warehouse tables and columns.

  - **Dimension Lookup Tags**. Define custom Meta Data for the Dimension Lookup.

TRANSFORMATION TAB

The transformation tab allows you to map Staging table columns to Attributes during the transformation.

There are 2 ways to define the mapping:

3. Use the **Auto Match** link below the bottom right of the table to automatically match Staging to Target Attributes based on Name.

4. Select the Mapping manually.

- **Double click a row or click Edit Mapping button** to change the mapping for the column, define an expression mapping, and add a description to the mapping for documentation purposes. Click the button and the Transform Mapping dialog appears.

  - **Staging Column Mapping.** Select the Staging table and Staging Column mapping for the Target Attribute from the drop downs.

  - **Expression.** Write an expression for the mapping. Note the expression can only reference Staging columns. The expression language is SSIS Expression Language.

  - **Transform Mapping Description.** A description of the mapping for documentation purposes. Non Mandatory.

  - **Transform Mapping Tags.** Define custom meta Data for the mapping.

ADVANCED TAB

- **Execution Order**. Define an Execution order for this Transform in its phase. If this Transform should be run after other Transforms then elevate its Execution order. This property is not currently used by the standard ETL generation templates. Execution order is determined by the Batch workflow document.

- **Transform Tags**. Define custom Design/Meta Data for the Transform.

- **Auto Generate Key**. This should remain true for Facts. If there is a case for a non-auto generated key for a fact, this can be set to false.

ORGANIZING YOUR FACTS, DIMENSIONS AND STAGING TABLES WITH SUB FOLDERS

Within the Dimension, Fact and Staging folders in Solution Explorer you can create sub folders to organize your Dimensions and Fact Tables into logical groups, and make them easier to find. To create a Sub Folder, right click the Dimensions/Fact Tables folder and select **Add>New Folder**.

The application supports a full hierarchy of folders within folders.

EXPRESSIONS

Dimodelo Architect supports expressions. The standard generation templates require expressions in SQL syntax or SSIS Expression, however, if you use custom generation templates, you can define expressions in any language you choose. The expression dialog states whether the expression should be T-SQL or SSIS.

When referencing Staging columns in a T-SQL Expression, use the following convention:

Source_System_Abbrev_Table_Name.Column_Name.

Where

- **Source_System_Abbrev** is the abbreviation for the Source System the table belongs too. Must match the Source System Abbreviation field of the Source System document.
- **Table_Name** is the table name of the Staging Table that contains the Staging Column. Must match the Staging Table name field in the Staging document.
- **Column_Name** is the name of the Staging Column. Must match the Staging Columns tab 'Column Name' field in the Staging document.

Alternatively you can use the following custom syntax for Staging and Warehouse table column names. This syntax is interpreted into the correct format by the ETL generation templates.

Staging Columns: **stg(Source_System_Abbrev, Table_Name, Column_Name)**

Where

- **Source_System_Abbrev** is the abbreviation for the Source System the table belongs too. Must match the Source System Abbreviation field of the Source System document.
- **Table_Name** is the table name of the Staging Table that contains the Staging Column. Must match the Staging Table name field in the Staging document.
- **Column_Name** is the name of the Staging Column. Must match the Staging Columns tab 'Column Name' field in the Staging document.

Warehouse Columns: w**hs(dim/fact, Table_Name, Column_Name)**

Where

- **dim/fact**. If the table the column belongs to is a dimension, then 'dim', otherwise 'fact'.
- **Table_Name** is the table name of the Warehouse Table that contains the Column. Must match the Dimension/Fact Table name field in the Dimension/Fact document.
- **Column_Name** is the name of the Attribute/Measure. Must match the Attribute Name/Measure Name tab 'Column Name' field in the Dimension/Fact document.

STANDARD MANAGEMENT COLUMNS

What are Management Columns? Management columns are columns that are added to your Fact and Dimension tables to help manage the data, during ETL. Management Columns include things like:

- Batch Identifier.
- Row Status (Current - Not Current).
- Row Effective Dates.
- Checksum (if you use checksums in your ETL)

Dimodelo Architect has the concept of a set of standard management columns. These are managed through the 'Dimension Standard Management Columns' and 'Fact Standard Management Columns' documents in the Reference Data project folder. You can set up different standard management columns for Facts and Dimensions.

Any management columns you create in these documents will be applied to the Data Warehouse tables when DDL is generated. This includes any changes and deletions of standard management columns. Standard Management columns can be disassociated from individual Fact and Dimension tables through the Fact or Dimension's editor. Custom management columns can also be added to each Fact and Dimension.

To modify Standard management columns, open the appropriate Standard Management Columns document in the Reference Data folder. There you can add, change and delete Standard Management columns.

The Dimodelo Architect project template includes a set of standard management columns that are required by the code generated from the default generation templates to operate correctly. They include:

Dimension Standard Management Columns:

- Batch_Execution_Id. The Id of the Batch that last added or updated the row in the dimension table.
- Row_Effective_Date. The Date the row became effective.
- Row_End_Date. The Date the row was no longer effective. This is associated with a change in Latest/Current status.
- Row_Is_Latest. Indicates the row is the latest row (or not the latest row) in a series of changes to the underlying member in the dimension.
- Row_Is_Current. Indicates the row still appears (or doesn’t appear) in the source system.
- SCD_Type_1_Hash. The checksum used to identify Type 1 changes to the dimension member.
- SCD_Type_2_Hash. The checksum used to identify Type 2 changes to the dimension member.

Fact Tables Standard Management Columns

- Batch_Execution_Id. The Id of the Batch that last added or updated the row in the fact table.
- Row_Is_Current. Indicates the row still appears (or doesn’t appear) in the source system.
- Checksum. The checksum used to identify changes to the fact.

Staging Standard Management Columns:

- Reject Ind. Used to indicate row has been rejected based on certain business rule criteria.

REFERENCE DATA

The reference data folder holds simple reference data editors for reference data required by the application.

PROJECT CONFIGURATION

INTRODUCTION

With Dimodelo Architect new projects have a default development, test and production configuration file that targets the locally installed instance of Microsoft Server (if there is one), and generate code to the Generated_Code folder in the project directory. While this is suitable for demonstration and proof of concept purposes, a real project that must support deployment to multiple environments requires a different configuration. This chapter describes our recommended approach to project configuration to support multiple environments.

The first step is to define a Project folder structure to cater for multiple environments. Dimodelo Solutions has a recommended Project folder structure. Read the *Step 1 - Set up the Solution Folder* Structure section for more details.

Dimodelo Architect has flexible code generation and deployment capabilities allowing it to support:

- Multiple target environments (e.g. Development, Test, Prod)
- Generating code to different file locations
- Deploying to non-local servers.

Typically there would be a Development, Test and Production environments. Each of these environments may have different SQL Server Instance Names, Source System Connection strings and deployment paths. To support multiple environments it is necessary to create a Dimodelo Architect Config file for each environment. Each configuration file contains connection strings, and deployment paths specific to the target environment. To understand DA Config file set read the *Step 2 - set up Dimodelo Architect Configuration Files* section.

Each Dimodelo Architect Configuration file is associated with a different Visual Studio Project Configuration. The user must select a Visual Studio project configuration prior to executing a Generate, Deploy and Batch operation. DA uses the DA Config file associated with the currently selected Visual Studio project configuration to determine target servers etc for the action. To generate and deploy to a different target environment, you simply switch between different Visual Studio Project Configurations. Visual Studio Project Configurations are a standard Visual Studio feature that allow you to define different environmental properties for each target environment and build for those different environments. To learn more about using Visual Studio Project Configurations with DA read the *Step 3 –Project Configuration* section.

Finally… the output paths for the generated code artefacts must to be modified to match the Project folder structure setup in step 1. See the *Step 4 - Set the Output Paths for the Generated Code* section for more details.

STEP 1 - SET UP THE SOLUTION FOLDER STRUCTURE

Because Dimodelo Architect uses information in the DA Config file to generate SSIS configuration files and set configuration file paths in packages, it is necessary to generate a version of the SSIS ETL for each target environment. To facilitate this we recommend a particular folder structure and generated code output structure.

We recommend the following folder structure for your solution

- Solution *(Source Code)*
  - Dimodelo *(The Dimodelo Architect Project)*
  - Workflow. *(Contains the Batch Workflow file(s))*
  - Cube *(The SSAS Multidimensional Project if required)*
  - Tabular *(The SSAS Tabular Project if required)*
  - Custom *(The generated code)*
    - Custom Database *(Custom Stored Procedures (if required))*
      - DEV
      - TEST …
      - PROD …
    - Custom SSIS *(Custom SSIS Project if required)*
    - Custom Config *(Custom SSIS Config files for the Custom SSIS project)*
      - DEV
      - TEST …
      - PROD …
- Build. *(Build folder for full Solution automated Build)*
- Package *(Package folder for full Solution automated Build)*

STEP 2 - SET UP DIMODELO ARCHITECT CONFIGURATION FILES

As discussed in the introduction, typically you need a DA Config file for each of the environments you are going to deploy the solution too, i.e. Development, Test, Production etc. This section describes how to create a DA config file.

The following steps describe how to set up a DA config file for a TEST environment using our recommended Solution folder structure.

1. Right click the Config Folder in the Solutions Explorer, and select Create New.

2. A new DA Configuration File is added to the folder and opens in the content window.

3. You can **change the name** of the DA Configuration File by selecting the DA Configuration File in Solution Explorer and changes the File Name property in the Properties Tool Window. If you don't see the Properties Tool Window, the from the Visual Studio menu click View>Properties. In this example the name would be set to TEST.cuf. .cuf is the extension for a DA config file.

4. **Set the ETL code (SSIS) deploy path**, by setting the 'ETL Code Deploy Path/Server Connection String' property on the Connections tab. Without a deploy path, the deployment process for ETL will fail. For Development environment DA Config files, set the deployment path to a local folder path (e.g. F:\ DW\ SSIS). For external server deployments (i.e. TEST and PROD environments) set the property to the network path URI for server. For example, if our TEST server was TSTSR01 then the property might be set to \\TSTSR01\F$\DW\SSIS. F:\DW\SSIS on the TSTSR01 server. DA currently only supports the package file deployment model for SSIS 2012 projects. The path can also be relative e.g. ..\Deploy.

5. **Modify Data Warehouse and Staging Database Connections** to match the target server as required. On the Connections tab, use the configure buttons to create connection strings to your Data Warehouse and Staging Databases for the target environment. In the TEST example we would set the connection string of the Data Warehouse to Data Source= TESTSVR;Initial Catalog= DW_Warehouse;Integrated Security=True

6. **Modifying Source System Connection strings**.

   The 'Connection strings' table allows you to specify environment specific connections strings for the source system connections specified in your Dimodelo Architect project. This way you can define different source connections for each environment, i.e. development, test, production etc. The name corresponds to the Connection Manager name in the project. You can modify the connection strings for each DA Config File (targeting a different environment) using the Configure button on each row.

   To add or modify a Connection strings:

   - In the Name field, type the name of the connection from your project. Note this must 'Connection Manager Name' in the Connection Manager dialog in your Dimodelo Architect Project exactly.

   - Use the configure button to configure your connection string. Alternatively you can type or paste a connection string into the connection string field. To manually set the Server Name and Provider fields, view the code of your connection in the Dimodelo Architect project, and use the values of the ServerName and Provider elements.

   Note: DA will maintain all DA Config files, as you add, remove Connection Managers in the project. It only maintains DA Config files that are associated to a Visual Studio Project Configuration. While new connection managers are added to all DA Config files, modifying a Connection Manager only maintains the DA Config file associated with the currently active Project Configuration. For example if a connection manager connection string is modified, it will only modify the connection string in the DA Config file associated with the active project configuration.

7. Additional Configuration

   The configuration tab is used to set some defaults for Data Warehouse tables. In most cases it is not recommended to change anything on this tab.

   1. **Data Warehouse Database Schema**: Use this field to specify the schema name of Data Warehouse tables in the Data Warehouse database. The default is dbo. Note DA currently does not use this property when generated code.

   2. **Dimension Table Prefix**: The dimension table prefix is applied to the front of the table name of every dimension table in your schema. This allows you to keep your design free of implementation details. Note DA currently does not use this property when generated code.

   3. **Fact Table Prefix**: The dimension table prefix is applied to the front of the table name of every fact table in your schema. This allows you to keep your design free of implementation details. Note DA currently does not use this property when generated code.

   4. **Custom Meta Data**: The custom Meta Data table can be used to create custom meta data that can be used in the generation, deployment or batch execution processes.

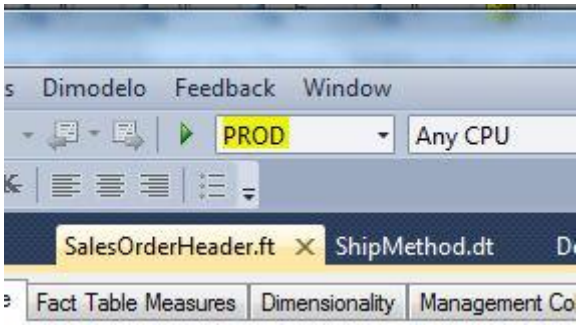
DELETING A DA CONFIG FILE

If you delete a Config file, be sure to also change any project configurations that access it, to access a DA Config file. If you don't you will get an error message saying the DA Config file could not be found, every time you attempt to add/change or delete any connection manager.

STEP 3 –PROJECT CONFIGURATION

Once a DA Config file has been created it needs to be associated with a Visual Studio Project Configuration, so that DA can use it.

1. **Create a new Visual Studio Project Configuration**. When the Generate, Deploy and Batch processes are executed, Dimodelo Architect uses the active Visual Studio Project Configuration to determine which Dimodelo Config file to use in the process. Before executing the Generate, Deploy or Batch processes, the user will select the desired project configuration.



Project Configurations are standard Visual Studio functionality. Dimodelo Architect has extended Visual Studio Project Configurations to include Dimodelo Architect specific project configuration. For more information about Visual Studio Project and Solution Configurations visit MSDN: Build Configurations. Each DA config file must be associated with a Project Configuration. To understand how to create a new Visual Studio Project Configuration visit How to: Create and Edit Configurations. We recommend you create a new project Configuration for each DA Config file that matches the name of the DA Config file.

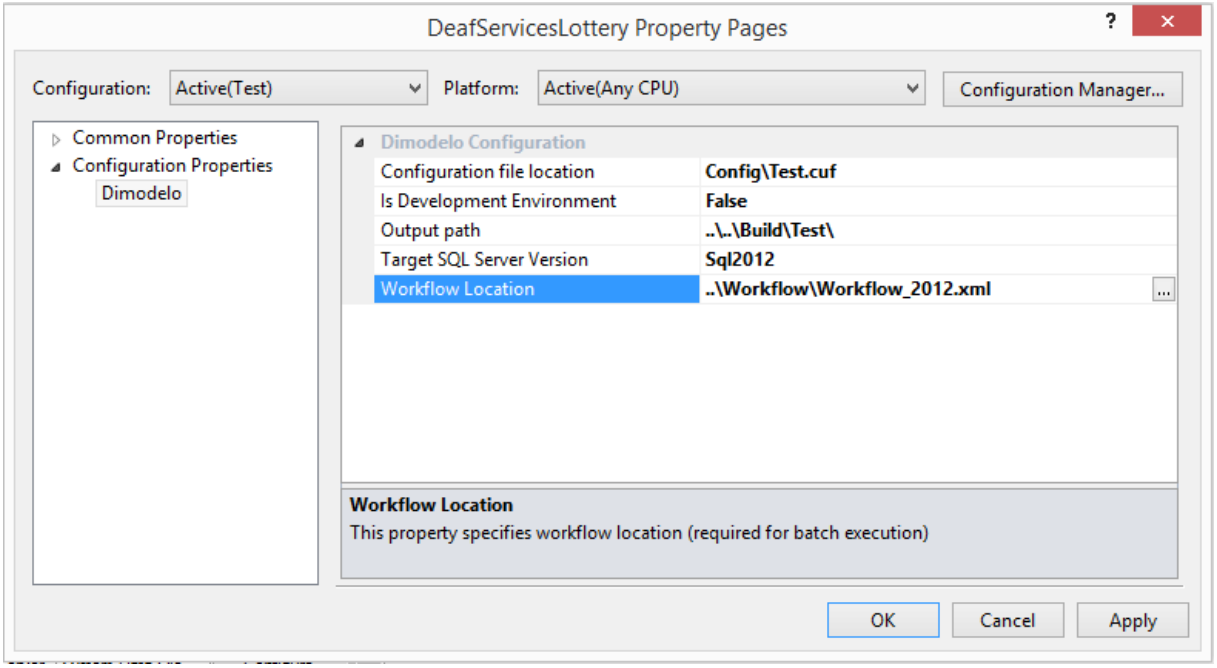


Figure 1 - Visual Studio Project Configuration dialog

2. **Modify the generation output path** to a path specific to the TEST DA configuration file. You should generate the output for each target environment to a different folder. To modify the generation output path, click the Output path row and use the … Browse button to select/create the path to the output folder. In our example the output path would be C:\Bulid\TEST, assuming the solution is saved

to C:. You can also use relative paths (relative to project file) which we highly recommend if there are multiple developer using Dimodelo e.g. ..\..\Build\Test. After generation the C:\Bulid\TEST folder will contain all the generated code for the TEST environment, as defined for this DA Config environment. When defining generation paths, make sure that the generation path in one configuration file is not the child of another path in another configuration file. The SSIS deployment action deploys all packages in the generated code path, including child folders. When deploying the code using the config file that contains the parent path, you will deploy the SSIS packages in that path, plus the packages in the child folders of that path, which includes the packages in the folder that contains the generated code from the other config file.This will causes deployment issues.

3. **Configuration File Location**. Once you have created the new project Configuration (e.g. TEST), set the 'Configuration File Location' property (see above) to the path of the DA Config file (e.g. C:\Solution\Dimodelo\Config\TEST.cuf). If you delete or rename a Config file, be sure to also change any project configurations that access it. If you don't, you will get an error message saying the DA Config file could not be found, every time you attempt to add/change or delete any connection manager.

4. **Is Development Environment**. Set this property too false for a development environment project configuration. Set it to True for all other environments. DA will use this in the future as a safeguard to warn users before deploying to a Production environment.

5. **Target SQL Server Version**. This property determine which version of SQL Server the Generate process generates SSIS packages for.

6. **Workflow Location**. This property determines what workflow file the Batch process uses, when running a batch from Dimodelo Architect with this VS project configuration.

STEP 4 - SET THE OUTPUT PATHS FOR THE GENERATED CODE

In the Generation/Templates folder of your DA project there are a number of .mnf manifest files that control the generation of different types of code artefacts. Each file contains xml, similar to that shown below:

```
<Generation_Template_Manifest>
  <Output_Relative_Location>SSIS/</Output_Relative_Location>
  <Template_Display_Name>SSIS 2008 Package Config for Staging</Template_Display_Name>
  <Generation_Engine_Name>SSIS 2008 Packages Generation Engine</Generation_Engine_Name>
  <Generation_Result_File_Name_Pattern>Staging.dtsConfig</Generation_Result_File_Name_Pattern>
  <Template_Relative_Location>SSIS_Staging_Configuration.xslt</Template_Relative_Location>
  <OperatesOn>Staging</OperatesOn>
  <Generates_For_Collection>
    <Generates_For_TargetType>ETL</Generates_For_TargetType>
  </Generates_For_Collection>
</Generation_Template_Manifest>
```

The Output_Relative_Location determines where the output for that code artefact type is written, relative to the 'Default Generation Output Path' of the currently active DA Config file.

In the example above, this is set to SSIS/. What this means is that for the artefact type represented by this manifest (which happens to be a Staging SSIS configuration file) the resulting generated artefact is written to the *'Default Generation Output Path'*/SSIS/ folder. In our example *'Default Generation Output Path'* = C:\Solution\Data Warehouse\TEST so the Staging SSIS configuration file is written to C:\Build\TEST\SSIS\. This works for all different DA Config files so, for example a DEV DA Config file might have a 'Default Generation Output Path' = C:\Solution\Data Warehouse\DEV so when generating with this DA Config file the Staging SSIS configuration file goes to C:\Build\DEV\SSIS.

To change the output path of all generated artefacts to the recommended solution folder structure we recommend you make the following changes.

1. Create a new Manifest folder under the Dimodelo Folder in the Solution and copy all the .mnf files from the generation folder into the Manifest folder. Why? From time to time Dimodelo Solutions issues updates. This may mean you need to overwrite the Generation folder of an existing project. When you overwrite the generation folder, you overwrite the manifests in that folder. By placing the manifests in their own folder it's possible to reapply these changes at a later date.

2. Change the manifests.

The following manifests apply to DDL generation. The Output_Relative_Location should be set to ./Database/*Folder Name* where folder name is the last folder name in the existing Output_Relative_Location.

- Clean_Up.mnf
- Control_Properties.mnf
- Create_Databases.mnf
- Staging_Tables.mnf
- Table_Synonyms_Dimensions.mnf
- Table_Synonyms_Facts.mnf
- Table_Synonyms_Staging.mnf
- Transform_Fact_Bridges_Pattern.mnf
- Warehouse_Tables.mnf
- Indexes_Staging.mnf
- Linked_Servers.mnf

The following manifests apply to SSIS package generation. The Output_Relative_Location should be set to ./SSIS/. You may only need to modify the 2008 or 2012 manifests depending on which version of SQL Server you are generating for.

- ExtractPattern2008_GenTemplate.mnf
- ExtractPattern2012_GenTemplate.mnf
- FactTransactionStandard2008_GenTemplate.m

- FactTransactionStandard2012_GenTemplate.m
- SSIS2008ProjectGenerationTemplate.mnf
- SSIS2012ProjectGenerationTemplate.mnf
- TranFactStdLedgerPattern2008_GenTemplate.mnf
- TranFactStdLedgerPattern2012_GenTemplate.mnf
- TransformDimensionPattern2008_GenTemplate.mnf
- TransformDimensionPattern2012_GenTemplate.mnf
- SSIS_Config_DB.mnf
- SSIS_Config_Excel.mnf
- SSIS_Config_Flat.mnf
- SSIS_Config_Staging.mnf
- SSIS_Config_Warehouse.mnf

The following manifests apply to Documentation generation. The Output_Relative_Location should be set to ../Documentation.

- Data_Dictionary_Dim.mnf
- Data_Dictionary_Fact.mnf
- DimodeloCss.mnf

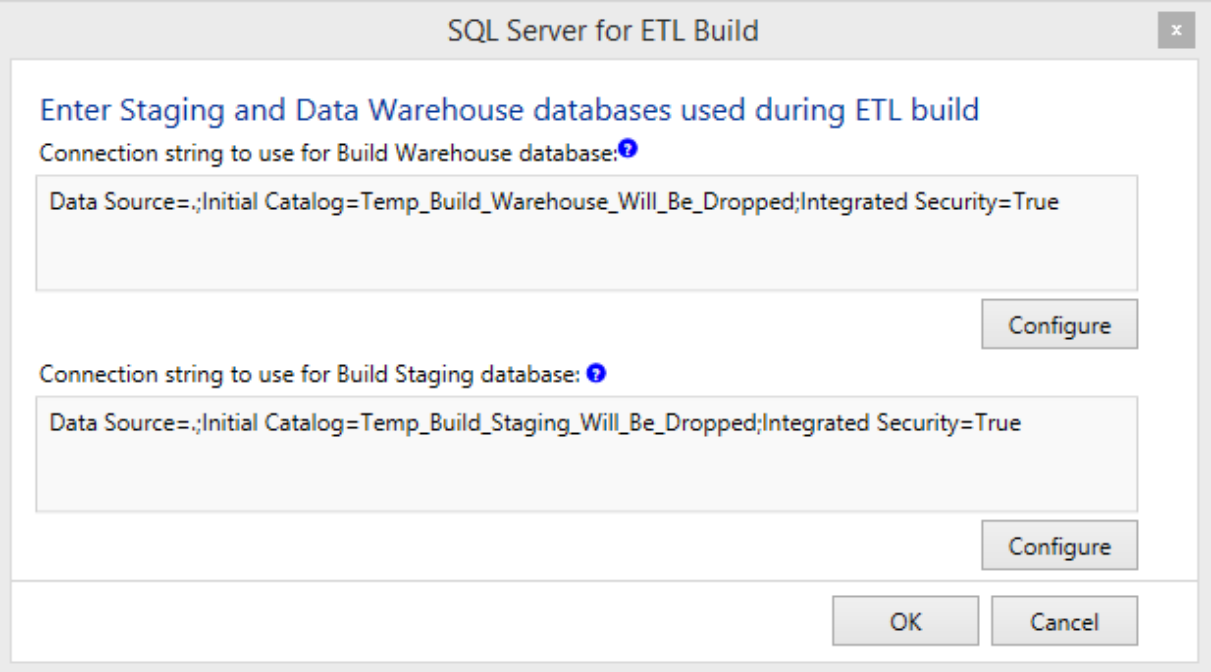
GENERATING AND DEPLOYING CODE

OVERVIEW

When you choose to Generate, the source code to create/maintain Data Warehouse and Staging database tables and the Extract Transform and Load (ETL) SSIS packages are generated from the design captured in DA.

When you choose to Deploy, the source code to create/maintain the Data Warehouse and Staging database tables is executed against the target database server, and the SSIS packages are moved to the Deployment folder path on a local or remote server.

When generating ETL code Dimodelo Architect first generates and deploys (and later deletes) a temporary Build version of the Staging and Data Warehouse databases to your build server (which can be your local PC). The Build server and database is configured in the Dimodelo> SQL Server for ETL Build menu option.



Code can be generated from within Dimodelo Architect through the Dimodelo menu. Dimodelo Architect has 9 Generate and Deploy options.

- **Generate> Generate Data Warehouse**. Generates the DDL for the Staging and Data Warehouse databases.
- **Generate > Generate ETL**. Generates the SSIS packages to implement the ETL.
- **Generate > Generate All.** Generates both the Data Warehouse and SSIS ETL.
- **Generate and Deploy > Generate and Deploy Data Warehouse**. Generates the DDL for the Staging and Data Warehouse databases and executes it against the target server.
- **Generate and Deploy > Generate and Deploy ETL**. Generates the ETL and deploys it to the Deployment path defined in the Configuration file.
- **Generate and Deploy > Generate and Deploy All**. Generates and deploys both the Data Warehouse and SSIS ETL.
- **Deploy > Deploy Data Warehouse**. Executes the previously generated DDL against the target Staging and Data Warehouse databases. The primary purpose being to update the Staging and Data Warehouse databases to match the latest version of the design.
- **Deploy > Deploy ETL**. Copies the previously generated SSIS Packages to the Deployment path defined in the Configuration file.

- **Deploy > Deploy All**. Does both deploy Data Warehouse and Deploy ETL.

When in Development we recommend that you use the **Generate and Deploy ETL** option. This option refreshes everything, so no change is missed. As you become more familiar with the process, you may want to just Generate and Deploy the ETL, if your changes have only affected the ETL and not the structure of any tables. You can also generate the all the code related to just a single Staging/Dimension or Fact table.

When generating and deploying to either a Test or Production environment, we recommend **Caution**, and following the recommendations in the Production Deployment Section.

Dimodelo Architect supports generating code for multiple environments. Code must be generated for each environment, as each environment will have different source connections and different target Staging and Data Warehouse Servers. More information about configuring DA to support multiple environments can be found in the *Project Configuration* section.

HOW TO GENERATE CODE

PREREQUISITES

It is important that configuration is set up correctly before generating the code. This includes

- Setting up connections strings for the Staging, Data Warehouse and Source databases for the target environment.
- Setting the Default Generation Output Path in the Dimodelo Architect Configuration file.
- Setting the ETL Code Deployment Path.

The configuration for Dimodelo Architect projects, supporting multiple Environments is discussed in detail in the *Project Configuration* section.

1. Before Generating ensure you have applied the ***Error! Reference source not found.*** and *Project Configuration*.

2. If you have added a new source system, check that the connection string in the associated DA Config file is correct. You may need to change the connection string for different environments in different DA Config files.

GENERATING ALL CODE

1. Make sure that the correct Project Configuration is selected for the environment you want to generate for. The Project Configuration is associated with a DA Config file and this determines connection strings etc. that are used in the generated SSIS packages.

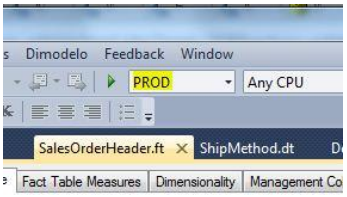
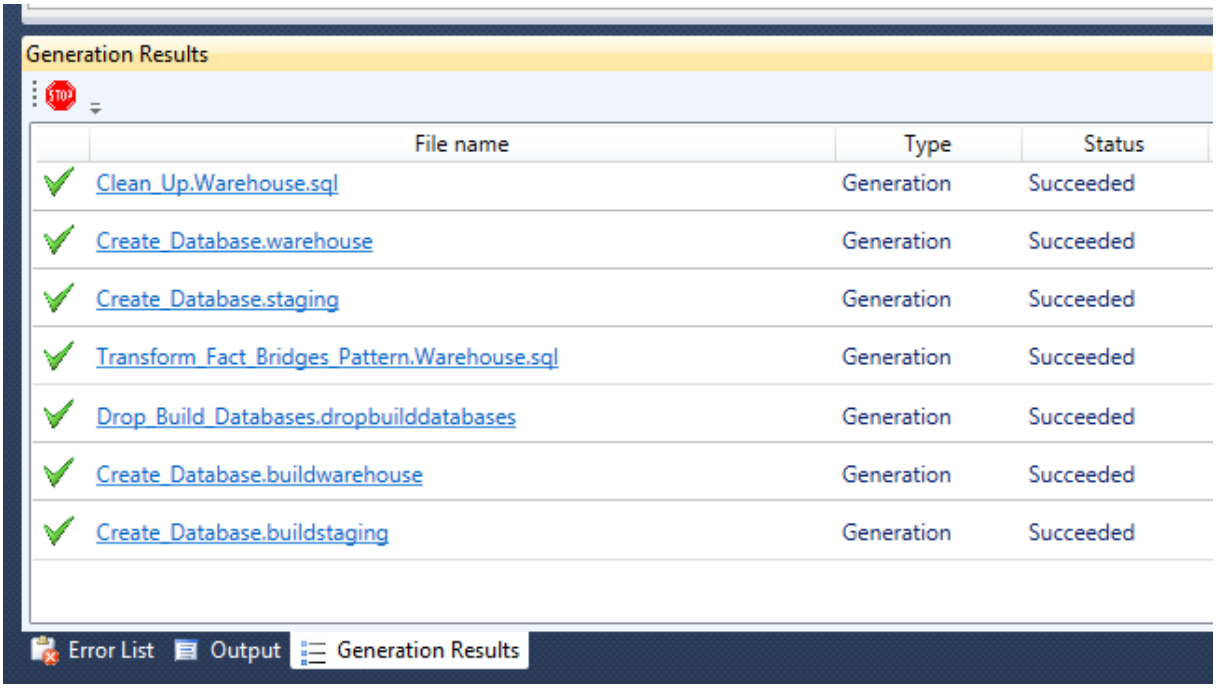


Figure 2 - Project Configuration

2. Click the desired menu option. E.g. **Dimodelo>Generate and Deploy> Generate and Deploy All**. This option generates and deploys all the code to create/modify your Data Warehouse and Staging databases and SSIS ETL.

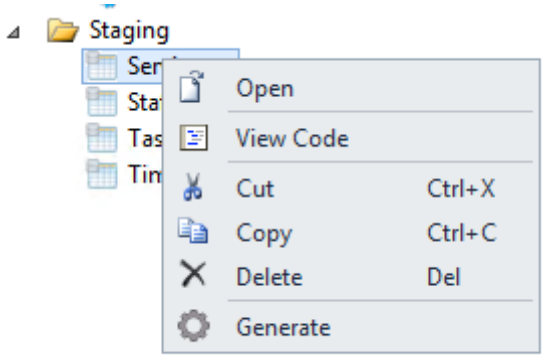
3. During code Generation **the Generation Results Window** will show a list of the files that have been generated. If the **Generation Results Window** is not visible, click the Generation Results tab at the bottom of the Dimodelo application. If there are errors, both the Generation Results Window and the Error List Tool window will display the errors.



4. View the generated code. You can click on any file link in the **Generation Results Window** to open the windows default application for that file type. Otherwise you can navigate to the output folder to view the output files.

GENERATING FOR JUST A SINGLE STAGING/DIMENSION OR FACT TABLE

Right click any Staging, Dimension or Fact and click Generate to Generate all the code associated with that table. This includes code to update the Staging/Data Warehouse databases and SSIS packages.



This method is only recommended for generating for development environments, unless you really understand the impacts of changes to your design. For Test and Production, it's recommended you do full generations prior to deployment.

Using this generation method Dimodelo Architect first generates and deploys all the code required to update all tables in the build Staging and Data Warehouse databases. This is important because of the dependencies between Staging, Dimension and Fact tables. If the entire latest version of the Data Warehouse was not available at the time the code artefacts are generated for the selected Staging, Dimension or Fact table, then generation may fail. This is why you see a lot of activity when you select to generate a single item. Fortunately generating the changes to the databases and deploying is quick.

Generating a single items requires an understanding of the impact and dependencies of changes to that item. For example, if you change a column in a Staging table, not only do you need to re-generate that Staging table you also need to generate all Dimensions and Facts that use that Staging table.

If you fail to re-generate dependant items then you may see errors in the batch execution.

Below is a table of common changes, and the generation action required.

Change	**Generation Action Required**
Change Staging table Columns.	Regenerate Staging table and any dependant Dimensions and Facts.
Change a Staging table Extract.	Only Regenerate Staging table.
Change Dimension Attributes.	Only Regenerate Dimension.
Change Dimension Business Keys.	Regenerate Dimension and associated Fact. (Actually it's likely you need to make changes to the Dimension Lookup in the Fact).
Change Dimension Transformation.	Only Regenerate Dimension.
Change Fact.	Only Regenerate Fact.

TROUBLE SHOOTING

Sometime code generation will fail. **This is usually due to an issue with the design or configuration Meta data**. A generation error is indicated by an ✖ on the generation results row.

Do the following to try to fix the issue:

- Review the error message to get a better understanding of where the error message occurred. Click the … at the end of the generation results row reporting an error.

The output will look something like this:

```
Name of SSIS Component/Executable that caused this error
---------------------------------------------------------
Employee Ole Db Source Component

Properties
----------
Connection = HumanResources

Failed property
---------------
SqlCommand = SELECT
     [EmployeeCode]
                              ,[FirstNames]
                              ,[LastNames]
                              ,[Gender]
                              ,[BirthDate]
                              ,[Title]
```

```
                                ,[LicenceClass]
                                ,[HomePhone]
                                ,[WorkPhone]
                                ,[Email]
                                ,[StartDate]
                                ,[TerminationDate]
                                ,[TerminationReason]
                                ,[DepartmentCode]
                                ,[UserId]

    FROM
    [dbo].[Employee] with (noLock)


Inner exception
---------------
Error source: {B2FC301E-DC7F-480A-856E-338A3C120A8D}, sub component: Connection
manager "HumanResources", message: SSIS Error Code DTS_E_OLEDBERROR.  An OLE DB error
has occurred. Error code: 0x80004005.
An OLE DB record is available.  Source: "Microsoft OLE DB Provider for SQL Server"
Hresult: 0x80004005  Description: "[DBNETLIB][ConnectionOpen (Connect()).]SQL Server
does not exist or access denied.".


Inner exception
---------------
Exception from HRESULT: 0xC020801C
Source query
---------------
SELECT
     [EmployeeCode]
                                ,[FirstNames]
                                ,[LastNames]
                                ,[Gender]
                                ,[BirthDate]
                                ,[Title]
                                ,[LicenceClass]
                                ,[HomePhone]
                                ,[WorkPhone]
                                ,[Email]
                                ,[StartDate]
                                ,[TerminationDate]
                                ,[TerminationReason]
                                ,[DepartmentCode]
                                ,[UserId]

    FROM
    [dbo].[Employee] with (noLock)

Inner exception
---------------
Exception from HRESULT: 0xC020801C
```

The output tells you:

- The name of the SSIS component that was being generated when the error occurred.
- The properties being set on the component.
- The value of the property that caused the error (in this case the SQLCommand was set to the generated SQL Query)
- The exception message .In this case the source database was not accessible.
- The generated source query. This can sometimes be the cause of the issue.
- The inner exception to the exception being generated.

Actions we suggest you take:

- Check the design meta data of the object being generated thoroughly for inconsistencies or omissions. This will usually be the cause of an issue.
- Copy and paste the generated SQL query to SQL Server Management Studio (SSMS) and run it against the source database. This can sometimes identify issues with the design meta data.
- Click on the SSIS package or SQL Script file in the generation results. This should open the code in the appropriate editor. Review the code, this can help you understand where the issue is.
- Sometimes the SSIS exception messages can be cryptic. Usually Googling on the exception code can give you clues on the issue.
- Visit dimodelo.desk.com for more information on known issues.
- Raise a ticket with our support.

UNDERSTANDING THE GENERATION PROCESS

Dimodelo Architect supports an open API for developers to create custom Generation Engines and Generation Templates. More information can be found in the API guide.

An explanation of the generation process can be found in the Dimodelo Architect Code Generation Process blog posts:

CUSTOM GENERATION TEMPLATES

Dimodelo Architect ships with an EzAPI and XSLT Generation Engine. The XSLT engine takes XSLT generation templates as input. There are several EzAPI templates that are used to generate SSIS Packages for ETL. More information about EzAPI can be found on the Dimodelo blog.

Developers can develop their own XSLT or EzAPI templates to generate custom code that better suits their target environment. They can choose to use the standard templates, a combination of standard and custom templates or only custom templates.

More information about creating your own SSIS generation templates can be found at:

- http://www.dimodelo.com/blog/2013/getting-started-with-ezapi/
- http://www.dimodelo.com/blog/2013/generating-ssis-packages-using-ezapi-hello-world/

CUSTOM GENERATION ENGINES

A custom generation engine can be created, if XSLT or EzAPI is not an appropriate way of generating code for a target platform.

CUSTOM DEPLOYMENT ENGINES

Dimodelo Architect supports an open API for developers to create custom Deployment Engines. More information can be found in the API guide.

A custom deployment engine can be created, if a deployment engine doesn't exist for your target **environment**. The standard deployment engine will deploy SQL and DDL to a Relational DBMS. If you have custom code for another platform, you can create a Deployment Provider to deploy that code to that platform.

GENERATED DOCUMENTATION

DA generates a data dictionary in two html files:

- Dim_Data_Dictionary.html which contains the Dimension documentation.
- Fact_Data_Dictionary.html which contains the Fact documentation. Fact_Data_Dictionary.html references Dim_Data_Dictionary.html but it must be in the same folder.

The style of the output can be modified by editing the Dimodelo.css file.

Example Documentation:

Time Sheet

The Time Sheet Fact records the weekly time entries of staff against services and tasks.

Reference: 23C
Physical Table Name: fact_Time_Sheet
Load Frequency: Daily

Measures

Name	Description	Data Type	Column Reference
TimeRecordId	Unique Identifier of the time entry.	int	
Hours	The number of hours recorded for the time entry.	decimal(18,2)	

Dimensionality

Dimension	Role Play	Link
Service		Service
Staff		Staff
Task		Task
Task	Initial_Task	Task

SOURCE CONTROL

Dimodelo Architect eases the issues commonly experienced by Data Warehouse projects when attempting to migrate Data Warehouse projects up through development environments.

Because Dimodelo Architect projects can be placed under source control just like any other Visual Studio project it is possible manage a Dimodelo Architect project just like any other project. Various trunk and branches can be created with different versions of the project, as required for major or minor releases, patches etc.

PRODUCTION DEPLOYMENT

PREREQUISITES

1. Install and Register Dimodelo Management Console on the Data Warehouse Server. We recommend using the 64 bit version for 64 bit operating systems. See the Installation Guide on the Dimodelo Solutions web site www.dimodelo.com for instructions.

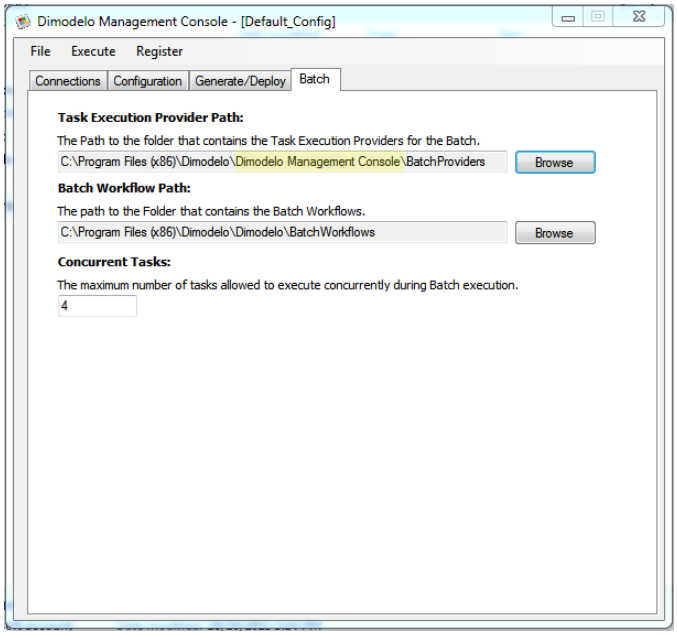
2. Create a folder on the server where SSIS packages can be deployed too.
3. Create a folder called Batch on your target server in a sensible place, perhaps where SSIS packages are deployed to.

4. Create and modify the DA Config file and DA Visual Studio Project configurations for the target environment. For more information see the ***Error! Reference source not found.*** and *Project Configuration* sections.

DEPLOYMENT PROCESS

This deployment process assumes that the DA project exists in the recommended in the *Step 1 - Set up the Solution Folder Structure* section.

1. Check in the DA project to Source Control. There may be other steps you need to do to tag/branch/merge the repository at this stage, but that's just standard source control.
2. **Backup the Data Warehouse database (If it isn't done automatically) in the target environment!!!!!**
3. In Dimodelo Architect
    - Select the project configuration for the environment you want to deploy to (i.e. TEST, PROD)
    - First generate the Data Warehouse code only through the Generate > Generate Data Warehouse option.
    - If you have made any changes to custom scripts or SSIS projects/packages in your Custom folder then manually copy from Custom/../DEV/ to Custom/…/*Env*/ where *Env* is the target deployment environment, and make any changes for the new environment as required.
    - Deploy the changes to the Data Warehouse and Staging Databases to your target environment servers using the Deploy > Deploy Data Warehouse option. **Note – This will update the databases in your target environment, so be sure you have a backup**!
    - Review all facts and dimensions for soft deleted columns (Z_DIL_Dropped) in the target Data Warehouse database– and remove these columns once satisfied.
    - Generate the ETL for the target environment. Use the Generate > Generate ETL option. **Note A warning dialog should appear**.
4. Deploy the ETL to the target environment. . Use the Deploy > Deploy ETL option.
5. Copy the workflow file (Data Warehouse/Workflow) to the Batch folder on the target Server.
6. Copy the Dimodelo/Config/Env.cuf file to the Batch folder on the target Server, where *Env*.cuf is the DA Config for file for target environment. Edit the *Env*.cuf on the server using Dimodelo Management Console. Change Task Execution Provider Path directory from "Dimodelo\Dimodelo" to "Dimodelo\Dimodelo Management Console" as shown below. When only Dimodelo Management Console is installed, the BatchProviders path used by Dimodelo Architect is invalid.



7. Deploy the Cube/Tabular model using your preferred method.

SCHEDULE THE ETL BATCH JOB

Read the scheduling the Batch section of the Batch Management Facility document for more information on scheduling the ETL Batch job.

ADVANCED DESIGN TECHNIQUES

CREATING A MANY TO MANY RELATIONSHIP

To create a Many to Many relationship between a Fact Table and Dimension you need to create a Group Dimension and an Intermediary Bridge Fact Table.

For more information about this technique you should refer to the 'Data Warehouse Toolkit' book by Ralph Kimball or visit the following TechNet article which explains the same technique in Microsoft terms. http://technet.microsoft.com/en-us/library/ms345139.aspx.

1. First create your Group Dimension, without any dimension attributes.
2. Next create a intermediary Bridge Fact Table, this time without any measures. Associate the Fact table with both the target (of the many to many join) Dimension and the Group Dimension. Use the **Standard** pattern in the ETL.
3. Open your target (of the many to many join) Fact Table, and associate it only with the Group Dimension.

In the Microsoft example in the above link,

- The Transaction Fact Table is the target Fact Table.
- The Customer Dimension is the Target Dimension.
- The CustomerAccount Fact Table is the Intermediary Bridge Fact Table.
- The Account Dimension is the Group Dimension.

INCORPORATING CUSTOM DDL AND SSIS INTO THE DIMODELO ARCHITECT PROJECT

Dimodelo Architect can manage deploy and execute custom DDL (Data Definition Language T-SQL for creating and managing the Data Warehouse and Staging databases) and SSIS packages.

In the ‘Step 1 - Set up the Solution Folder Structure’ section we recommended a folder structure that included a Custom folder. When DA deploys the DDL and SSIS packages it will also deploy code contained in this folder or sub folders of this folder. Place any custom DDL (with a file extension of .sql) or SSIS packages (with a file extension of .dtsx) into this folder.

The name of the folder containing the custom DDL is important. You will notice that in the Database folder, each folder is prefixed with a number. The number determines the order in which the contained .sql files are executed. To have the custom DDL deploy (be executed} in a particular place in the order of execution, name the folder with the appropriate prefix number.

The batch workflow is determined by the workflow file. As long as the custom SSIS packages conforms to a naming standard that the workflow file understands as a file it should execute, it will be executed as part of the ETL batch.

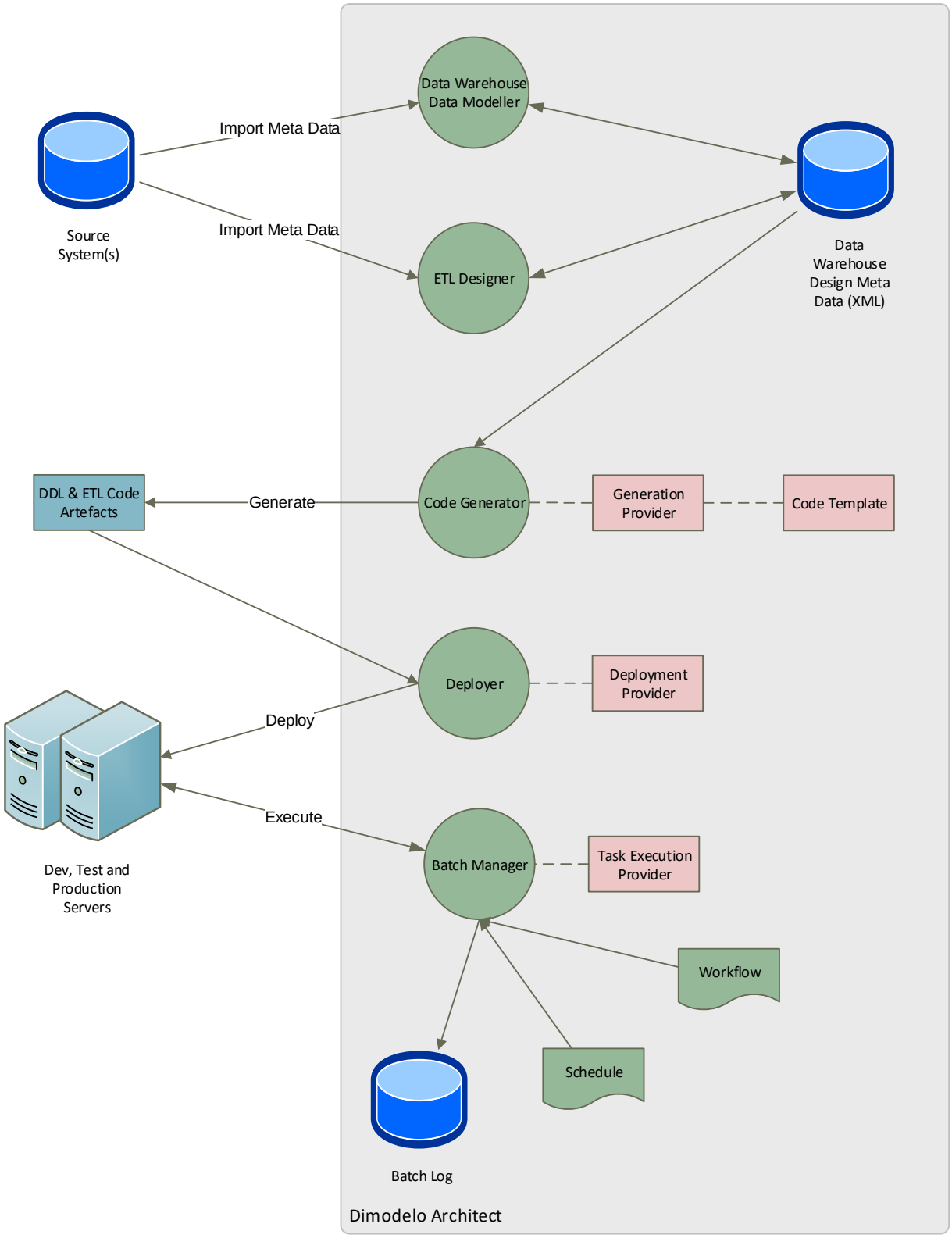
It is possible to have Dimodelo Architect manage a Staging, Dimension or Fact table, and have a custom SSIS package to implement its ETL. Simply define the Staging, Dimension or Fact table in DA, but don’t define an Extract or Transform for it.

PRODUCT ARCHITECTURE

'Dimodelo Architect' is designed to simplify building a Data Warehouse and its ETL. Out of the box, it comes with all the necessary patterns and code generators to create a Data Warehouse on the Microsoft SQL Server Platform.

'Dimodelo Architect' is also extensible and customisable, allowing Data Warehousing professionals to configure the tool to generate code using their own patterns and practices.

The various logical components of Dimodelo Architect are shown in the diagram on the next page.



SUPPORT

For support please email support@dimodelo.com or visit our website at our support web page.