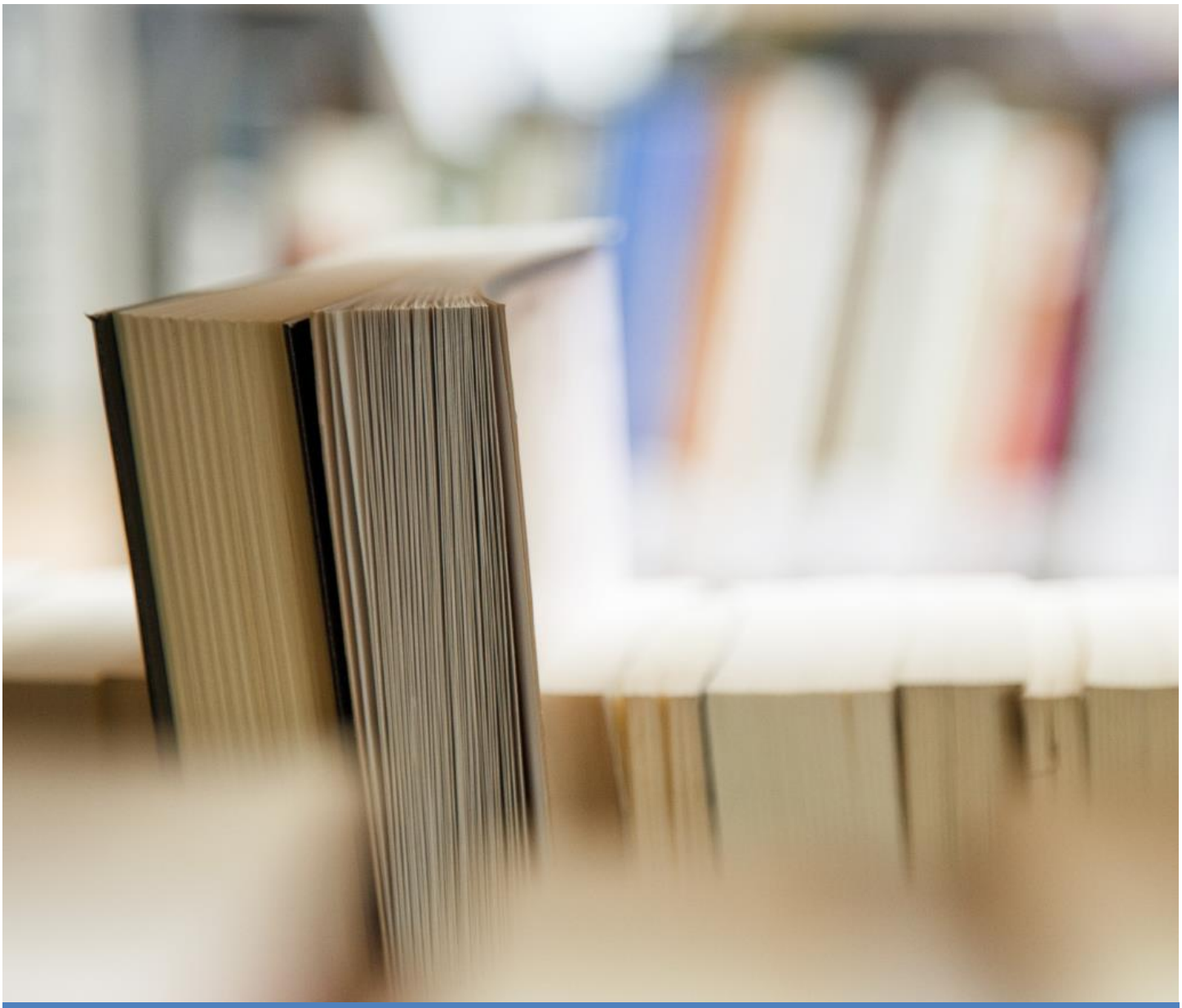


Managing and Monitoring ETL Batches

Batch Management Facility

CONTENTS

DOCUMENT HISTORY

Version	Changes
2.1.1	Initial Version.
2.2	Added Scheduling the Batch Section.
2.8	Updates for Batch Management database.
2.15	Updates and clarifications.

OVERVIEW

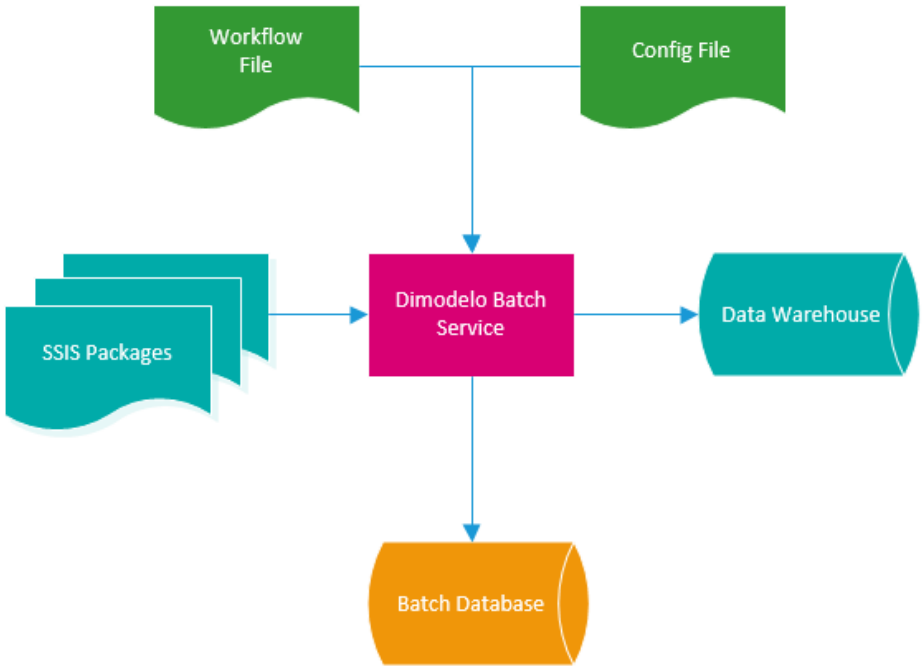
Dimodelo Solutions provides an ETL Batch Management facility through the Dimodelo Management Console. The Dimodelo Management Console is a small .NET utility that is designed to be installed on a server and used to generate code, deploy code and (mainly) run ETL batches. An ETL Batch is the execution of the set of SSIS Packages that extract data from Source Systems, and transform and load the data into the Data Warehouse. The Batch Management facility of the Dimodelo Management Console provides the following functions:

- **ETL Batch Execution**. Define the workflow of the Batch and Dimodelo Architect will execute the Batch Tasks in order.
- **ETL Batch Execution Logging**. Batch task progress is logged to a Batch Database.
- **Batch Analysis**. The Batch Database can be used to analyze Batch performance over time.
- **Batch Scheduling**. Batches can be scheduled using standard scheduling tools like Windows scheduler.
- **Batch Notifications**. Batch Success and Failure Notifications can be configured.
- **Batch Utilization Management**. The Batch can run multiple parallel tasks. Over time, the Batch Manager can determine the longest running tasks, and execute them first, so that the overall Batch finishes sooner.

The Batch can be run directly from Dimodelo Architect while developing a Data Warehouse Solution, and via the command line using Dimodelo Management Console on Test and Production servers.

THE BATCH PROCESS OVERVIEW

There are a number of Components used in the Batch process.



- **The Batch Service**. This executes the batch. You can invoke the Batch Service via Dimodelo Architect, or on the command line via Dimodelo Management Console.
- **Workflow file**. This specifies the order of tasks to be executed in a batch. You can have multiple workflow files for different batches (e.g. Daily, Weekly, and Monthly). The Batch workflow file defines a set of Phases and Tasks (equivalent to a set of SSIS packages/ stored procedures). The tasks can use wildcards, so that adding tasks does not necessarily mean having to update the batch workflow. The batch workflow file is passed to the Batch Service when a Batch is run.
- **Config file**. This is passed to the Batch Service, along with the Workflow. It specifies the location of the Batch Database to use with the Batch run, and the number of concurrent Tasks that can be executed within the Batch.
- **A Batch Database**. Used to log Batch progress.
- **Control Properties Table**. A table that lives in the Data Warehouse and is used by ETL tasks (like SSIS Packages and Stored Procedures) to get the current Batch Id, Batch Execution Id, and Batch Effective Date.

When a Batch Service is invoked (either from Dimodelo Architect/Visual Studio or via the Server side Dimodelo Management Console), the config file and work flow files are passed to the Batch Service as parameters. When a Batch starts the following sequence of events occurs.

1. A new Batch record is created and the start of the Batch is logged in the Batch database. The target Batch database is defined in the config file. The current Batch Id, Batch Execution Id, and Batch Effective Date are written to the control properties table.
2. The Batch Service reads the workflow file.
3. For each Phase in the workflow file, the Batch Service logs the start of the Phase to the Batch database. The current Phase Id is written to the control properties table.

    a. For each Task in the Phase, the Batch Service retrieves and executes the specified task execution provider, passing the Task name. There are existing Task Execution Providers for stored procedures and SSIS packages. The Task Execution Provider finds all Tasks (e.g. SSIS Packages, Stored Procedures) that match the Task name (including wild-carded names) and executes then in-turn, with longer running tasks (determined from execution history) executing first. The Task execution provider logs the start and the end of the Task in the Batch database, and writes the current task id to the control properties table. All Tasks within a Phase can be run in parallel. The Configuration file specifies how many tasks can be run in parallel.
4. At the end of the Phase, the Batch Service logs the end of the Phase.
5. At the end of the Batch, the Batch Service logs the end of the Batch.

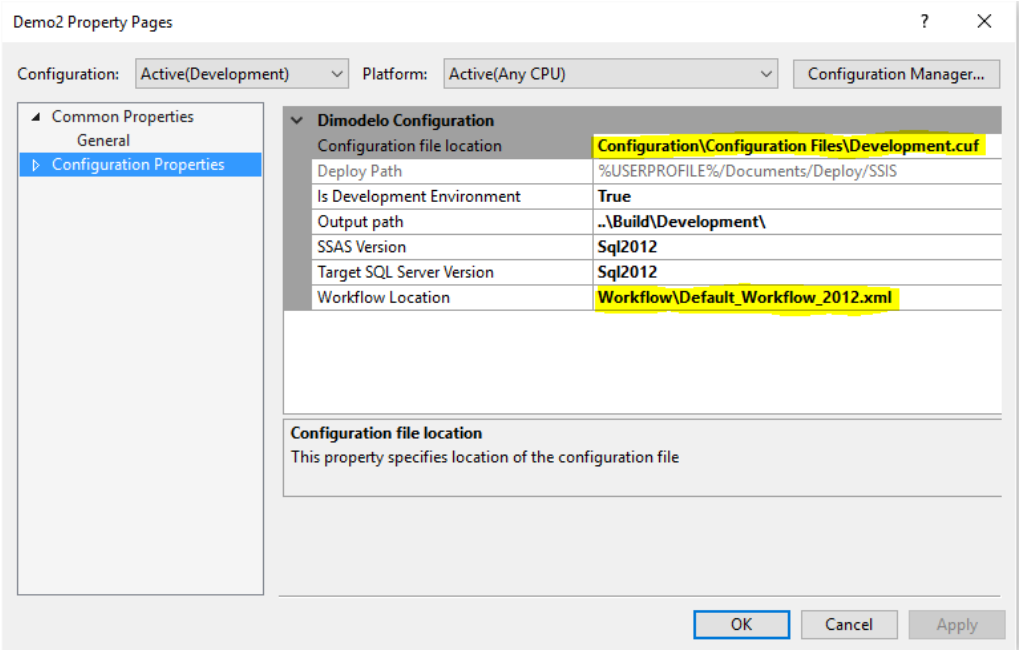
If there are any errors during task execution then Task, Phase and Batch errors are logged, and the Batch stops executing. A Batch can be run in Recover mode. In this mode, a new Batch Id, and Batch Effective Date are not generated. Only a new Batch Execution Id is generated. A Batch can have one or more Batch Executions. All Dimodelo Architect ETL patterns have been written to be self-healing.

RUNNING THE BATCH

RUN THE BATCH THROUGH VISUAL STUDIO/DIMODELO ARCHITECT

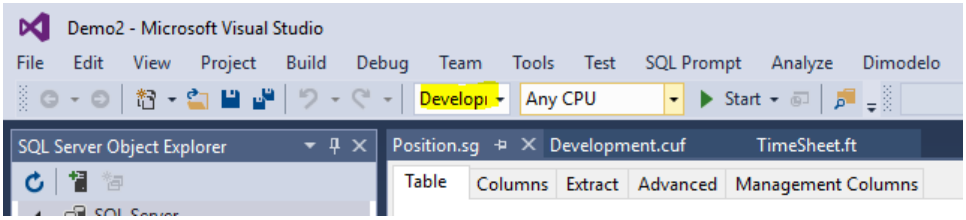
The ability to run a Batch through Dimodelo Architect is available to assist in Data Warehouse Development. To run the Batch simply select the Dimodelo>Batch menu option. The batch progress is reported in the Visual Studio Output tool window.

The Batch menu option invokes the Batch Service, passing the currently selected Dimodelo Architect Config File and the work flow file identified by the Workflow Location property in the Visual Studio Project properties.



The default is called Default_2012_Workflow.xml in the Batch folder of your project directory. To modify the workflow that the Batch runs when invoked from Dimodelo Architect, change the Workflow Location property.

The Current Dimodelo Architect Config file is the Config file associated with the currently selected Visual Studio Project Configuration.



RUN THE BATCH THROUGH DIMODELO MANAGEMENT CONSOLE

To run batches on a Test or Production Server, where Dimodelo Architect is not installed, you can use the Dimodelo Management Console. This is a small .NET application that can execute the same Generate, Deploy and Batch execution tasks as Dimodelo Architect.

To run a Batch on a server you need 4 things:

1. Dimodelo Architect Management Console installed with a Full (Server), Trial or Community license. The Trial is time restricted, and the Community edition is restricted to only 20 tasks per day. Enough to run a daily Batch for a small Data Warehouse.
2. A configuration file, created through Dimodelo Architect.
3. A Batch Workflow file.
4. Deployed Code.

To execute a batch you must run a command line statement, either directly in a command window, or through a .bat command file. The syntax of the command is:

```
"Dimodelo Management Console" -cmd [Command] --cfg [config file] -workflow [workflow file] -runtype [New|Recover] -output [output text file]
```

An example command is shown below:

```
"C:\Program Files\Dimodelo\Dimodelo Management Console 2.15\Dimodelo Management Console" --cmd Batch --cfg "..\Configuration\Configuration Files\Development.cuf" --workflow Default_Workflow_2014.xml --runtype New --output out.txt
```

When you run the Batch with the 'New' instruction, a new Batch is started, with a new Batch Id, Batch Effective Date and Batch Execution Id. When you run the Batch with the 'Recover' instruction, a new Batch is started, but only a new Batch Execution Id is generated.

The progress of the batch to the Batch database and is written to the console, or to the output text file, if one is specified.

A Batch can be scheduled using Windows scheduler (or equivalent).

For each of your production and Test environments you will want to install a server version of the Dimodelo Management Console so you can execute ETL Batches in these environments.

WORKFLOW

A Batch workflow consists of an XML file with custom schema. An example of the schema is shown below. The workflow below consists of three phases Extract, Transform Dimensions and Transform Facts.

```
<Batch_Workflow>
  <Batch_Type>Nightly SQL Server 2012</Batch_Type>
  <Workflow>
    <Phase>
      <Phase_Name>Extract</Phase_Name>
      <Task>
        <Task_Name>Extract_</Task_Name>
        <Task_Provider>SSIS 2012 Package Task Execution Provider</Task_Provider>
      </Task>
    </Phase>
```

```
    <Phase>
      <Phase_Name>Extract Derived</Phase_Name>
      <Task>
        <Task_Name>Extract_Derived</Task_Name>
        <Task_Provider>SSIS 2012 Package Task Execution Provider</Task_Provider>
      </Task>
      <Task>
        <Task_Name>Extract_Derived%</Task_Name>
        <Task_Provider>StoredProcExecutionProvider</Task_Provider>
      </Task>
    </Phase>
    <Phase>
      <Phase_Name>Transform_Dim</Phase_Name>
      <Task>
        <Task_Name>Transform_Dimension_</Task_Name>
        <Task_Provider>SSIS 2012 Package Task Execution Provider</Task_Provider>
      </Task>
    </Phase>
    <Phase>
      <Phase_Name>Transform_Facts</Phase_Name>
      <Task>
        <Task_Name>Transform_Fact</Task_Name>
        <Task_Provider>SSIS 2012 Package Task Execution Provider</Task_Provider>
      </Task>
    </Phase>
    <Phase>
      <Phase_Name>Audit</Phase_Name>
      <Task>
        <Task_Name>Meta_Audit%</Task_Name>
        <Task_Provider>StoredProcExecutionProvider</Task_Provider>
      </Task>
    </Phase>
  </Workflow>

</Batch_Workflow>
```

The Workflow contains a set of Phases which in turn contain a set of tasks. Each Phase executes in order and each task in order within the Phase. The task name can be wild carded. Each task execution provider should be written to interpret wild card task names into a set of tasks to execute, as is the case with the StoredProcExecutionProvider and SSIS2008PackageTaskExecutionProvider.

- **<Batch_Type>**. The Batch Type is simply a name for the Workflow file, it has no execution significance. It's recorded in the batch database for each batch execution.
- **<Workflow>**. Wraps Phases and Tasks within the Workflow. Only a single Workflow element is allowed.
- **<Phase>**. A Phase represents a Phase of a ETL batch. For example the Extract Phase, the Transform Dimensions Phase and the transform Facts Phase. All task within a Phase must be able to be executed in parallel, but the Batch Execution Service will complete all tasks within a Phase before moving on to the next Phase.
    - **<Phase_Name>.** A logical name for the Phase. The Phase name is used in the Batch Logging.
    - **<Phase_Provider>.** The default task provider for the Phase. Not currently used by the Batch Execution Service.
- **<Task>.** The task element represents either a single task or group of similar tasks within the workflow. The task name element defines the name of the task, and the task name can have a wild card value that is matched to multiple physical tasks. Physically a task is a stored procedure or SSIS package. Other types of tasks can be executed through custom build task execution providers.
    - **<Task_Provider>.** This is the name of the Task Execution Provider that executes the task. The Name must match the <Provider_Name> Element of the Task Execution Provider Manifest.

Read the Task Execution Providers section for more information on Task Execution providers, and Manifests. The standard choices for the value of Task_Provider are SSIS2008PackageTaskExecutionProvider to execute 2008 SSIS packages and StoredProcExecutionProvider to execute stored procedures. Custom Task Execution providers can be created.

- **<Task_Name>.** The name of the task (stored procedure or SSIS package) that will be executed. The Name can contain a wild card, so that multiple physical stored procedures or SSIS packages are executed from the one task declaration. For the StoredProcExecutionProvider the wildcard is a %. For the SSIS2008PackageTaskExecutionProvider, no wild card is required, the Task Execution provider will attempt to match all SSIS packages that start with the task name. The current generation templates generate SSIS packages with the prefix Extract_*Table_Name* for Staging extracts, Transform_Dimension_*Table_Name* for Dimension transforms and Transform_Fact_*Table_Name* for Fact transformations.

To modify the work flow, simply add new Phases and/or new Tasks.

EXECUTING CUSTOM CODE DURING THE WORKFLOW

Dimodelo Architect can execute custom stored procedures and SSIS packages in the batch workflow. The lesson on custom code explains adding custom code to the project.

Dimodelo Architect will deploy custom code to the server if it's included in the Custom code project and named appropriately. (i.e. stored procedures .Staging.sql or .Warehouse.sql suffix)

By adding a task to the workflow that matches the name of the custom stored procedure or custom SSIS package, Dimodelo will attempt to execute it. Ensure that the correct task provider is specified.

Note: only stored procedures deployed to the data warehouse database are available to the batch service. The file that contains the stored procedure in the Custom Code Project must have a .Warehouse.sql suffix.

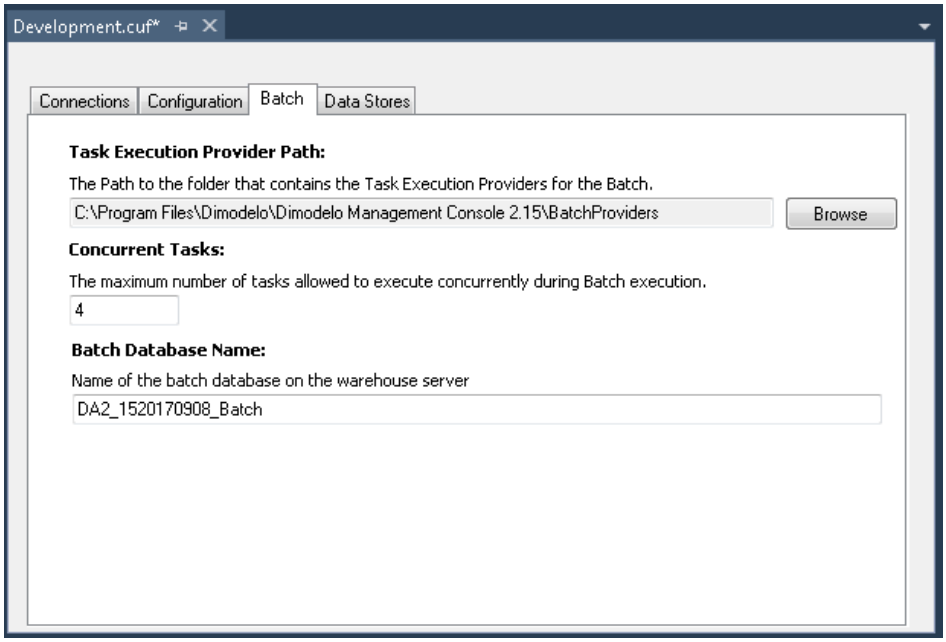
CONFIGURATION

There are 2 additional configuration setting that need to be set prior to execution the batch.

- **Concurrent Tasks**. Concurrent Tasks defines the number of batch tasks that can execute concurrently within a Phase of the Batch. The value of Concurrent Tasks configuration setting on the Batch tab determines the number of threads that are started to execute tasks in the batch. The batch processor manages these threads, and which tasks are executed on them at which time. Only tasks within a Phase can be executed concurrently. i.e. All Extracts are executed, then all Dimension Transforms, then all Fact Transforms etc. Over time the batch processor will dynamically adjust. It will determine which tasks are the longest running within a Phase, and will start these first in order to shorten the overall length of the batch.

- **Task Execution Provider Path**. The path to the folder that contains the Task Execution Providers for the Batch Execution.

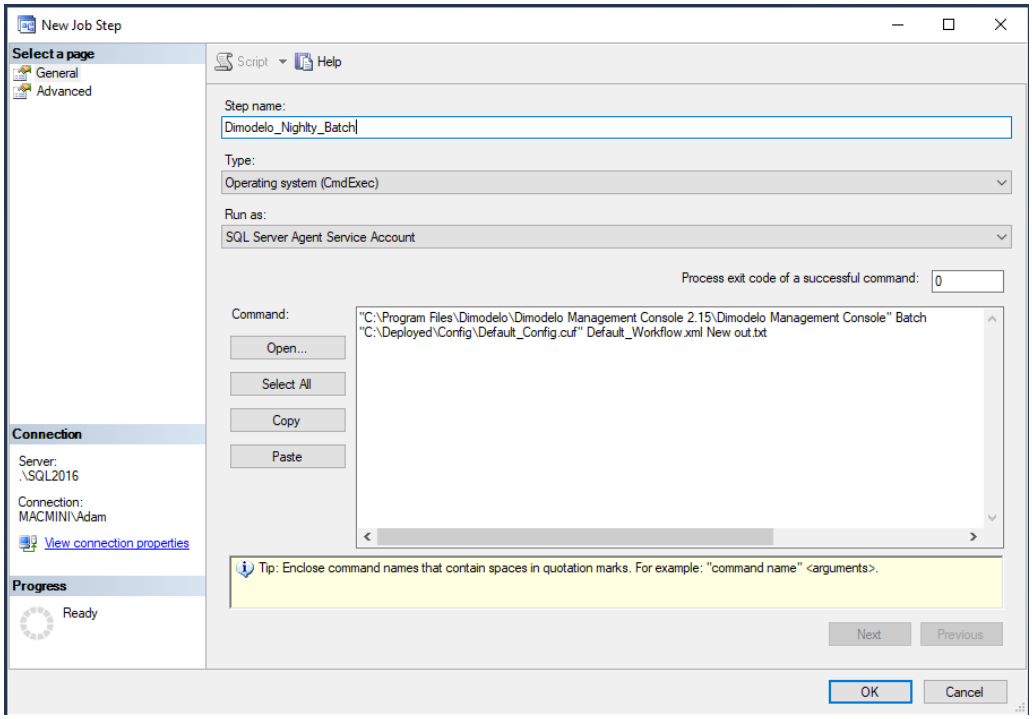
- **Batch Database Name**. The name of the database that captures batch execution information. This database will be deployed to the same server as your Data Warehouse database.


Development.cuf*
Connections
Configuration
Batch
Data Stores
Task Execution Provider Path:
The Path to the folder that contains the Task Execution Providers for the Batch.
C:\Program Files\Dimodelo\Dimodelo Management Console 2.15\BatchProviders
Browse
Concurrent Tasks:
The maximum number of tasks allowed to execute concurrently during Batch execution.
4
Batch Database Name:
Name of the batch database on the warehouse server
DA2_1520170908_Batch

SCHEDULING THE BATCH

Any scheduler capable of running a command line process can run the Dimodelo Architect batch.

The recommended approach is to use SQL Server job agent. Create a job and step that executes an Operating system command.



You may need to add a Credential, Operator and Proxy to run the step 'As' - https://docs.microsoft.com/en-us/sql/ssms/agent/create-a-sql-server-agent-proxy.

Through the job, you can also set up notifications to operators on job completion. The command line returns -1 on failure, so it's possible to see success or failure in the email notification.

It's also possible to schedule the job through the Windows Task Scheduler. Use the Task Scheduler to execute a Dimodelo Management Console batch command line, as described in the 'Run the Batch through Dimodelo Management Console' section.

CONTROL PROPERTIES TABLE

The Control Properties table lives in the Data Warehouse and is used by ETL tasks (like SSIS Packages and Stored Procedures) to get the current Batch Id, Batch Execution Id, and Batch Effective Date. These columns are used in the insertion and update of data in the Data Warehouse. It also helps to drive the Batch Execution process, and can assist failure diagnostics. The Control Properties are updated by the Batch Execution Service during the course of Batch Execution.

current_batch_effective_date	The process effective date of the current batch being executed. Note. This is not necessarily the same as today's date. If a batch failed, and the Batch is restarted in Recover mode a day later, the current_batch_effective_date would be yesterday's date, its process effective date. The current_batch_effective_date is used as the effective date of new records in the data warehouse.
current_batch_execution_id	The unique id of the current execution of the current batch. A single batch can be executed more than once (due to failure). The details of each execution are recorded in the Batch_Execution table in the Batch Database. New records in dimension and fact tables have the current_batch_execution_id written to the batch_execution_Id field.
current_batch_id	The batch number that is currently being executed. A batch can be executed multiple times (due to failure). Each execution of the batch gets a new id, while the batch id remains static until an execution is successful.
Current_phase_execution_id	The unique Id of the currently executing phase.
Current_task_execution_id	The unique Id of the currently executing task.
Status columns	There is a status column for Batch, Batch Execution, Phase and Task. The values are 'completed' for completed items, 'in progress' for items in progress and 'failed' for failed items. A failed task causes a cascading update of the phase, Batch Execution and Batch status fields.

BATCH DATABASE

The batch database will exist on the same server as your Data Warehouse database. The name of the database is configured in the **Batch Database Name** property on the Batch tab of the Dimodelo Architect configuration file.

The batch database captures ETL batch execution information including task durations and messages. You can query and report from this database to monitor and analyze your ETL batch processes.

BATCH TABLE

Capture a row per Batch initiation.

Column Name	Description
Batch_Id	Unique Id of the Batch. Generated by the Batch Service for each new Batch.
Batch_Effective_Date	The Effective Date of Data loaded into the Data Warehouse with the Batch Id.
Batch_Type	Always 'full'
Correlation_Id	Deprecated
Batch_Status	'completed' for completed Batches, 'in progress' for Batches in progress and 'failed' for failed Batches.
Batch_Error_Message	Error Message for failed batches.
Batch_Error_Batch_Execution_Id	The id of the Batch Execution that produced the Batch failure.

BATCH_EXECUTION TABLE

Captures a row per execution of a batch. A batch could have multiple execution, in the case of failure.

Column Name	Description
Batch_Execution_Id	Unique Id of the Batch Execution. Generated by the Batch Service for each new Batch Execution. A single batch can be executed more than once (due to failure). The details of each execution are recorded in the Batch_Execution table. New records in dimension and fact tables have the current_batch_execution_id written to the batch_execution_Id field.
Batch_Id	The Batch Id Of the parent Batch the Batch Execution belongs too.
Batch_Execution_Status	'completed' for completed Batch Executions, 'in progress' for Batch Executions in progress and 'failed' for failed Batch Executions.

Batch_Execution_Start_Date	The Start Date and Time of the Batch Execution run.
Batch_Execution_End_Date	The End Date and Time of the Batch Execution run.
Batch_Execution_Error_Message	Error Message for failed Batch Executions.
Batch_Execution_Error_Phase_Id	The id of the Phase that produced the Batch Execution failure.
Batch_Execution_Duration	The duration in seconds of the batch Execution.

PHASE EXECUTION TABLE

Captures a row per phase execution.

Column Name	**Description**
Phase_Execution_Id	The Unique Id of the Phase Execution.
Batch_Execution_Id	The Id of the parent Batch Execution of the Phase Execution.
Phase_Name	The Name of the Phase, from the WorkFlow file.
Phase_Start_Time	The Start Date and Time of the Phase run.
Phase_End_Time	The End Date and Time of the Phase run.
Phase_Duration	The duration in seconds of the Phase execution.
Phase_Status	'completed' for completed Phases, 'in progress' for Phases in progress and 'failed' for failed Phases.
Phase_Error_Message	Error Message for failed Phase Execution.
Phase_Error_Task_Id	The id of the Phase that produced the Phase Execution failure.

TASK EXECUTION TABLE

Capture a row per every task executed.

Column Name	Description
Task_Execution_Id	The Unique Id of the Task Execution.
Phase_Execution_Id	The Id of the parent Phase Execution of the Task Execution.
Task_Name	The Name of the Physical Task Executed. This is will be the Stored Procedure or SSIS Package name. Other task execution providers will set the name as required.
Task_Start_Time	The Start Date and Time of the Task execution.
Task_Status	'completed' for completed Tasks, 'in progress' for Tasks in progress and 'failed' for failed Tasks.
Task_Duration_Seconds	The duration in seconds of the Task execution.
Task_Error_Num	A task dependent error code returned by the task execution provider. Only written if available.
Task_Error_Severity	A task dependent error severity returned by the task execution provider. Only written if available.
Task_Error_Proc_Line_Num	A task dependent line of the procedure that caused the error. Returned by the task execution provider. Only written if available.
Task_Error_Row_Num	The Data row number that caused the error. Returned by the task execution provider. Only written if available.
Task_Error_Message	A task dependent error message returned by the task execution provider. Only written if available.
Task_End_Time	The End Date and Time of the Task execution.

TABLE STATISTICS TABLE

As of version 2.13, Dimodelo Architect captures the number of rows inserted, updated and deleted by each task during a batch execution.

TASK PROGRESS

As of version 2.15 Dimodelo Architect captures the steps each task executes, if supported by the generation template. The generated code can write to this table, as it executes significant pieces of the code. For example, 'get row count', 'update', 'insert', 'reorder'. The persistent staging table is a good example. A large load is broken into segments, and progress for each segment is written to the task progress table.

The step start date/time, duration, affected entity, and (if available) row count, file name, partition/segment number is captured.

EXTENSIBILITY

Dimodelo Architect supports an open API for developers to create custom Task Execution Providers. More information can be found in the Task Execution Providers section.

A custom Task Execution Provider can be created, if a Task Execution Provider doesn't exist for your target environment. The following standard Task Execution Providers are implemented:

- StoredProcExecutionProvider will execute Stored Procedures in a Relational DBMS.
- SSIS2008PackageTaskExecutionProvider to execute 2008 SSIS packages.
- SSIS2012PackageTaskExecutionProvider to execute 2012 SSIS packages.
- SSIS2014PackageTaskExecutionProvider to execute 2014 SSIS packages.
- SSIS2016PackageTaskExecutionProvider to execute 2016 SSIS packages.

TASK EXECUTION PROVIDERS

Tasks are executed by Task execution Providers. A Task Execution providers is required for each Server Type/Platform that batch tasks execute on. The StoredProcExecutionProvider shipped with 'Dimodelo Architect' will execute stored procedures on Microsoft SQL Server.

When the Batch Service executes a Batch, it uses the 'Task Execution Provider Path' in the Dimodelo Architect config file to find the details of available Task Execution Providers. The Batch Service searches the directory for files with the extension '.mnf'. These files represent Task Execution provider manifests that describe available task execution providers. An example Manifest is shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<Task_Execution_Provider_Manifest>
    <Provider_Name>StoredProcExecutionProvider</Provider_Name>
    <Provider_Class>com.dimodelo.batch.StoredProcExecutionProvider</Provider_Class>
    <Provider_Assembly>StoredProcExecutionProvider.dll</Provider_Assembly>
</Task_Execution_Provider_Manifest>
```

- **<Provider_Name>.** The Name of the Task Execution Provider. This should match the name of the <Task_Provider> in the workflow.
- **<Provider_Class>.** The name of the concrete class that implements the Task Execution Provider. This should exist in the same folder as the manifest, or in the GAC.
- **<<Provider_Assembly>.** The name of the containing Assembly for the concrete class.

Using the information in the manifest, the Batch Service can invoke a concrete Task Execution Provider. When the Batch Service executes a Task in the workflow, the Batch Service invokes the concrete Task Execution Provider where the manifest <Provider_Name> = the workflow <Task_Provider>, and executes the Process method of the concrete class passing the Task Name as a parameter.

Dimodelo Architect supports an open API for developers to create custom Task Execution Providers. It uses the concept of a Factory class to create concrete representations of an Abstract TaskExecutionProvider class. Only the Process method of the abstract class needs to be implemented by the concrete class. The abstract class takes care of error handling, background worker management etc.

The basic pseudo code of a process method is:

Use the Task Name Pattern to get a list of tasks.

```
Iterate through the tasks for the Task
        this.taskStarted(taskName)
        Execute the task.
        If error then
                this.TaskFailed(taskName)
        Else
                this.taskComplete(taskName)
        End If
End iteration
```

Each task execution provider should be written to interpret wild card task names into a set of tasks to execute, as is the case with the StoredProcExecutionProvider.

For more information about the Task Execution API please contact Dimodelo Solutions.

DO I HAVE TO USE THE DIMODELO MANAGEMENT CONSOLE TO RUN ETL BATCHES?

The code generated by Dimodelo Architect is just plain SSIS Packages and Stored Procedures. It is possible to orchestrate and schedule the execution of the SSIS Packages and Stored Procedures through an SQL Server job, or master SSIS package, so there is no requirement to have Dimodelo Management Console installed on your server.

However, you will need to code yourself all the functionality that Dimodelo Management Console provides, like Batch orchestration, Batch logging, and Batch Notifications.

Dimodelo Architect does generate a license check component into each SSIS package, so each server you run the packages on must be licensed with a server license. Each developer license includes a single server license. The license component can be deleted from the package, but each time you generate, it will return.