

Key2 Consulting

Data Ingestion Framework (DIF): Configuration Database Guide

September 21, 2023

**Table of Contents**

[Document Background 3](#_Toc146267391)

[Metadata Information 3](#_Toc146267392)

[Configuration Entries 11](#_Toc146267393)

[Initial Project Set-up 16](#_Toc146267394)

[Ingestion Scenarios 19](#_Toc146267395)

[Partition 19](#_Toc146267396)

[Source Filtering 19](#_Toc146267397)

[Source Audit Query 20](#_Toc146267398)

[High Watermark Tracking – Incremental Extracts 20](#_Toc146267399)

[Custom Source Query 20](#_Toc146267400)

[DIGroupTask Setup 21](#_Toc146267401)

[Pipeline Logging 22](#_Toc146267402)

# Document Background

The data ingestion process brings data from disparate systems into the data lake. The data is landed into the Bronze Zone and then transferred and transformed into the Silver Zone.

The Data Ingestion Framework (DIF) is used to consume data and is targeted at ingesting data using both full and incremental design patterns from SQL Server and file data sources. This framework can be extended to include other data sources such as Excel and JSON. The goal of the framework is to create a template pipeline and inform those pipelines via metadata.

The DIF includes two specific components:

* Metadata and configuration information (as well as logging data)
* Template Azure Data Factory (ADF) pipelines to ingest data into the Azure environment.

This document describes the configuration and metadata entries needed in the DIF database, along with the logging information that can and should be captured as part of the ADF pipeline logic.

# Metadata Information

The first step to using the ingestion framework is to add metadata information into the DIF database to describe both the source data that you want to ingest as well as the destination that you are going to populate. At a minimum, the sources and destinations should include information about the **systems** involved (the servers that contain the data), the **repositories** (databases, folders, etc.), and the **datasets** themselves (the tables, views, delimited files, etc.). Optionally, the metadata tables also support storing schema information, meaning the columns, fields, etc. along with their attributes (data types, max lengths, nullability, primary/foreign key designation, etc.). The following table lists the four metadata tables that are used to capture this information:

|  |  |
| --- | --- |
| Metadata Table Name | Description |
| metadata.System | Contains the list of systems (servers, Azure storage accounts, etc.) that serve as either a source of data or a destination for data |
| metadata.Repository | Includes the list of repositories (databases, file shares, etc.) that contain one or more sets of data that will be used as sources or destinations |
| metadata.Dataset | Contains details related to each dataset (table, view, file, etc.) that contains actual source data or will be the destination for a set of data |
| metadata.Attribute | Includes the granular list of attributes (columns, fields, etc.) for a given dataset, to include details about each attribute (data type, length, scale, precision, etc.) |

These four metadata tables rely on several reference tables that contain standard types and classifications that are used to describe the metadata entries. The following table lists those reference tables and the current table entries as of the writing of this document:

|  |  |  |
| --- | --- | --- |
| Reference Table Name | Description | Available Values |
| metadata.SystemType | Contains a list of standard system types | RDBMS |
| File Server |
| FTP Server |
| Azure Storage Account |
| metadata.EnvironmentType | Contains a list of standard environment types | Development |
| Testing |
| Staging |
| Pre-Production |
| Production |
| metadata.RepositoryType | Contains a list of standard repository types | SQL Server Database |
| Oracle Database |
| Azure DB Database |
| Azure Data Lake Zone |
| Windows Fileshare |
| metadata.DatasetClass | Contains a list of standard dataset classes | Fact Table |
| Dimension |
| Reference Table |
| Log Table |
| Reference View |
| metadata.StorageType | Contains a list of standard data storage types | Relational Table |
| Relational View |
| Parquet File |
| CSV File |
| metadata.PartitionType | Contains a list of standard partition types | Quarter |
| Month |
| Year |
| None |
| metadata.DataType | Contains a list of standard data types | char |
| varchar |
| nchar |
| nvarchar |
| text |
| ntext |
| binary |
| varbinary |
| image |
| uniqueidentifier |
| tinyint |
| smallint |
| int |
| bigint |
| date |
| time |
| smalldatetime |
| datetime |
| datetime2 |
| datetimeoffset |
| bit |
| decimal |
| numeric |
| smallmoney |
| money |
| float |
| real |

To populate each of the four metadata tables, a set of stored procedures can be used to add and update table entries. The following table describes each of these stored procedures, including details regarding the parameters that can be used for each:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stored Procedure | Description | Parameter | Required | Usage |
| DIF.AddSystem | Adds a new system | @SystemType | Yes | Indicates the system type (from the list above) |
| @EnvironmentType | Yes | Indicates the environment type (from the list above) |
| @SystemName | Yes | Provides the system name; must be unique and cannot duplicate any existing table entries |
| @SystemDescription | No | Describes the system |
| @SystemProgram | No | Indicates the Key2 group that owns or is responsible for the system |
| @SystemFQDN | Yes | Provides the fully-qualified domain name for the system; must be unique and cannot duplicate any existing table entries |
| @SystemUserName | No | Provides a user name if required for connecting to the system |
| @SystemSecretName | No | Provides a key vault secret name if required for connecting to the system |
| @SystemIsEnabled | Yes | Indicates if the system is enabled for data integration tasks |
| DIF.UpdateSystem | Updates an existing system | @SystemKey | Yes | Identifies the system to be updated |
| @SystemType | Yes | Indicates the system type (from the list above) |
| @EnvironmentType | Yes | Indicates the environment type (from the list above) |
| @SystemName | Yes | Provides the system name; must be unique and cannot duplicate any existing table entries |
| @SystemDescription | No | Describes the system |
| @SystemProgram | No | Indicates the Key2 group that owns or is responsible for the system |
| @SystemFQDN | Yes | Provides the fully-qualified domain name for the system; must be unique and cannot duplicate any existing table entries |
| @SystemUserName | No | Provides a user name if required for connecting to the system |
| @SystemSecretName | No | Provides a key vault secret name if required for connecting to the system |
| @SystemIsEnabled | Yes | Indicates if the system is enabled for data integration tasks |
| DIF.AddRepository | Adds a new repository | @SystemName | Yes | Identifies the parent system |
| @RepositoryType | Yes | Provides the repository type (SQL Server Database, Windows Fileshare, etc.) |
| @RepositoryName | Yes | Provides the name of the repository |
| @RepositoryDescription | No | Provides a repository description |
| @RepositoryPathPattern | No | Indicates the pattern for accessing a dataset within the repository |
| @RepositoryIsEnabled | Yes | Indicates whether the repository is enabled for data integration tasks |
| DIF.UpdateRepository | Updates an existing repository | @RepositoryKey | Yes | Identifies the repository to be updated |
| @SystemName | Yes | Identifies the parent system |
| @RepositoryType | Yes | Provides the repository type (SQL Server Database, Windows Fileshare, etc.) |
| @RepositoryName | Yes | Provides the name of the repository |
| @RepositoryDescription | No | Provides a repository description |
| @RepositoryPathPattern | No | Indicates the pattern for accessing a dataset within the repository |
| @RepositoryIsEnabled | Yes | Indicates whether the repository is enabled for data integration tasks |
| DIF.AddDataset | Adds a new dataset | @SystemName | Yes | Identifies the parent system |
| @RepositoryName | Yes | Identifies the parent repository |
| @StorageType | Yes | Provides the storage type for the dataset |
| @PartitionType | No | Provides the partition type (with a default of "None") |
| @DatasetNameSpace | No | Provides the dataset namespace (i.e., schema for a table or view) |
| @DatasetName | Yes | provides the name of the dataset (within the associated repository) |
| @DatasetDescription | No | Provides a dataset description |
| @DatasetPath | Yes | Provides the path to the dataset (i.e., <schema>.<name>, \<parentfolder>\<subfolder>, etc.) |
| @DatasetExternalVersionID | No | Provides a version identifier if one is available |
| @DatasetIsCreated | Yes | Indicates if the dataset already exists |
| @DatasetIsEnabled | Yes | Indicates if the dataset is enabled for data integration tasks |
| DIF.UpdateDataset | Updates an existing dataset | @DataSetyKey | Yes | Identifies the Dataset |
| @SystemName | Yes | Identifies the parent system |
| @RepositoryName | Yes | Identifies the parent repository |
| @StorageType | Yes | Provides the storage type for the dataset |
| @PartitionType | No | Provides the partition type (with a default of "None") |
| @DatasetNameSpace | No | Provides the dataset namespace (i.e., schema for a table or view) |
| @DatasetName | Yes | provides the name of the dataset (within the associated repository) |
| @DatasetDescription | No | Provides a dataset description |
| @DatasetPath | Yes | Provides the path to the dataset (i.e., <schema>.<name>, \<parentfolder>\<subfolder>, etc.) |
| @DatasetExternalVersionID | No | Provides a version identifier if one is available |
| @DatasetIsCreated | Yes | Indicates if the dataset already exists |
| @DatasetIsEnabled | Yes | Indicates if the dataset is enabled for data integration tasks |
| DIF.AddAttribute | Adds a new attribute | @DatasetKey | Yes | Key value for the dataset |
| @DataType | Yes | Identifies the associated DataType. Examples are varchar, int, datetime2 |
| @AttributeName | Yes | Attribute name in the associated dataset |
| @AttributeSequenceNumber | Yes | Attribute order in the corresponding dataset |
| @AttributeMaxLength | No | maximum value for a given data type. Example A varchar(50) attribute would have a max length of 50 |
| @AttributePrecision | No | Precision is the number of digits in the given data type. For example, decimal(12,2) the precision would be 12 |
| @AttributeScale | No | Scale is the number of digits to the right of the decimal point. For example, decimal (12,2) the scale would be 2 |
| @AttributeIsNullable | No | Identifies if the Attribute is nullable  Default to 1 |
| @AttributeIsPrimaryKey | No | Identifies if the Attribute is part of the PK in the dataset  Default to 0 |
| @AttributeIsUnique | No | Identifies if the Attribute is part of the unique identifier of the dataset  Default to 0 |
| @AttributeIsForeignKey | No | Identifies if the Attribute is part of a foreign key reference in the dataset  Default to 0 |
| @AttributeIsWaterMark | No | Identifies if the Attribute is used to track high watermark values  Default to 0 |
| @AttributePartitionKeyOrder | No | Identifies if the Attribute is used to partition the dataset  Default to 0 |
| @AttributeDistributionKeyOrder | No | Identifies if the Attribute is part of the distribution key  Default to 0 |
| @AttributeIsEnabled | No | Determines if an Attribute is active or not  Default to 1 |
| DIF.UpdateAttribute | Updates an existing attribute | @AttributeKey | Yes | Identifies the Attribute key |
| @DatasetKey | Yes | Key value for the dataset |
| @DataType | Yes | Identifies the associated DataType. Examples are varchar, int, datetime2 |
| @AttributeName | Yes | Attribute name in the associated dataset |
| @AttributeSequenceNumber | Yes | Attribute order in the corresponding dataset |
| @AttributeMaxLength | No | maximum value for a given data type. Example A varchar(50) attribute would have a max length of 50 |
| @AttributePrecision | No | Precision is the number of digits in the given data type. For example, decimal(12,2) the precision would be 12 |
| @AttributeScale | No | Scale is the number of digits to the right of the decimal point. For example, decimal (12,2) the scale would be 2 |
| @AttributeIsNullable | No | Identifies if the Attribute is nullable  Default to 1 |
| @AttributeIsPrimaryKey | No | Identifies if the Attribute is part of the PK in the dataset  Default to 0 |
| @AttributeIsUnique | No | Identifies if the Attribute is part of the unique identifier of the dataset  Default to 0 |
| @AttributeIsForeignKey | No | Identifies if the Attribute is part of a foreign key reference in the dataset  Default to 0 |
| @AttributeIsWaterMark | No | Identifies if the Attribute is used to track high watermark values  Default to 0 |
| @AttributePartitionKeyOrder | No | Identifies if the Attribute is used to partition the dataset  Default to 0 |
| @AttributeDistributionKeyOrder | No | Identifies if the Attribute is part of the distribution key  Default to 0 |
| @AttributeIsEnabled | No | Determines if an Attribute is active or not  Default to 1 |

To correctly use the ingestion framework and to move to the next step in terms of configuration information, both the source of data and the destination for that data must be described in the tables listed above. For example, suppose the following exists:

* A dimension table named **Dim.Property**
* A database named **ProjectX** contains that dimension table
* A SQL Server named **testserver.key2consulting.com** houses that database

Furthermore, supposed the data in that table needs to be ingested into the following data lake location:

* A subfolder named **dim\_property**
* A folder named **bronzezone** contains that subfolder
* A storage account at the URL **https://stkey2deveastus001.blob.core.windows.net/** houses that folder

In this case, the following entries (at least) would need to be made into the appropriate tables (using the stored procedures above, providing appropriate information for each of the parameters used with each):

* Two entries in the metadata.system table:
  + One to describe the **testserver** SQL Server
  + Another to describe the **stkey2deveastus001**storage account location
* Two entries in the metadata.repository table:
  + One to describe the **ProjectX** source database
  + Another to describe the destination **bronzezone** destination folder
* Two entries in the metadata.dataset table:
  + One to describe the **Dim.Property** source table
  + Another to describe the **dim\_property** subfolder

Optionally, schema information for the columns in the Dim.Property table can be added to the metadata.attribute table using the DIF.AddAttribute and DIF.UpdateAttribute stored procedures.

***Note****: While the stored procedures listed above can be used to add or update entries in each of the metadata tables, it is also acceptable to insert or update entries directly using SQL statements, ADF pipelines, or any other mechanism that supports writing information to an Azure SQL database. Care should be taken, however, to ensure that any updates applied do not affect other users dependent on existing metadata information.*

# Configuration Entries

The second step to using the ingestion framework is to add configuration entries into the DIF database that describe the ADF pipelines and associated data integration tasks that you want those pipelines to execute. The following table lists the seven configuration tables that are used to capture this information:

|  |  |
| --- | --- |
| Configuration Table Name | Description |
| config.Project | Contains a list of projects that “own” pipelines, data integration tasks, etc. |
| config.EnvironmentConfig | Includes a list of environment configuration entries for each project that can be used for things like dynamic ADF-linked services and datasets |
| config.Pipeline | Describes the ADF pipelines that are owned by a project and that are used for data integration tasks |
| config.PipelineGroup | Contains parent-child groupings of pipelines |
| config.DIGroup | Contains a list of data integration groups that can be executed by a given ADF pipeline |
| config.DITask | Contains a list of data integration tasks that describe data movement from a source to a destination |
| config.DIGroupTask | Includes many-to-many relationships between DIGroup records and DITask records, allowing a given data integration task to belong to many groups and a given data integration group to contain many tasks |

These seven configuration tables rely on a few additional reference tables that contain standard types and classifications that are used to describe the configuration entries. The following table lists those reference tables and the current table entries as of the writing of this document:

|  |  |  |
| --- | --- | --- |
| Reference Table Name | Description | Available Values |
| metadata.ProjectType | Contains a list of standard project types | Key2 |
| metadata.LoadType | Contains a list of standard data integration load types, along with their subsequent load types if a given data integration task should be changed automatically from one load type to another | Full Only |
| Full Then Incremental |
| Incremental |

To populate each of the seven metadata tables, a set of stored procedures can be used to add and update table entries. The following table describes each of these stored procedures, including details regarding the parameters that can be used for each:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stored Procedure | Description | Parameter | Required | Usage |
| DIF.AddProject | Adds a new project | @ProjectType | Yes | Identifies the project type name |
| @ProjectName | Yes | Identifies the new project name |
| DIF.UpdateProject | Updates an existing project | @ProjectKey | Yes | Key for the updated Project |
| @ProjectType | Yes | Identifies the project type |
| @ProjectName | Yes | Identifies the updated project name |
| DIF.SetEnvironmentConfig | Sets (adds or updates) an environment configuration entry for a given project | @ProjectName | Yes | Provides the name of the project the environment config entry belongs to |
| @EnvironmentConfigName | Yes | Provides the environment config entry name |
| @EnvironmentConfigValue | Yes | Provides the environment config entry value |
| DIF.AddPipeline | Adds a new pipeline | @ProjectName | Yes | Identifies the new project name |
| @PipelineShortName | Yes | New PipelineShortName value |
| @PipelineFullName | Yes | New PipelineFullName as it is deployed in ADF |
| @PipelineFolder | No | New PipelineFolder value from ADF |
| @PipelineDescription | No | New PipelineDescription value |
| DIF.UpdatePipeline | Updates an existing pipeline | @PipelineKey | Yes | Identifies the updated Pipeline |
| @ProjectName | Yes | Identifies the new Pipeline name |
| @PipelineShortName | Yes | New PipelineShortName value |
| @PipelineFullName | Yes | New PipelineFullName as it is deployed in ADF |
| @PipelineFolder | No | New PipelineFolder value from ADF |
| @PipelineDescription | No | New PipelineDescription value |
| DIF.AddPipelineGroup | Adds a new pipeline group |  |  |  |
| DIF.UpdatePipelineGroup | Updates an existing pipeline group |  |  |  |
| DIF.AddDIGroup | Adds a new data integration group | @DIGroupName | Yes | Name for the new DIGroup |
| @PipelineKey | Yes | Key value for the Pipeline associated with this new DIGroup |
| @DIGroupIsEnabled | No | Bit value to determine if it is enabled or not |
| DIF.UpdateDIGroup | Updates an existing data integration group | @DIGroupKey | Yes | Key value for the given DIGroup |
| @DIGroupName | Yes | Name for the updated DIGroup |
| @PipelineKey | Yes | Key value for the Pipeline associated with this new DIGroup |
| @DIGroupIsEnabled | No | Bit value to determine if it is enabled or not |
| DIF.AddDITask | Adds a new data integration task | @SourceDatasetKey | Yes | Key value for the source dataset |
| @DestinationDatasetKey | Yes | Key value for the destination dataset |
| @LoadType | Yes | Identifies the LoadType record |
| @DITaskSourceFilterLogic | No | Used to filter the source dataset. Example "and City = ‘Atlanta’" |
| @DITaskWaterMarkLogic | No | Used to execute incremental extracts from the source dataset. Example "and ETLEditDate > %MinWaterMark% and ETLEditDate <= %MaxWaterMark%" |
| @DITaskEnabled | No | Defaults to 1 |
| @SourceFilterLogicIsEnabled | No | Determines if the SourceFilterLogic is active or not. Defaults to 0 |
| DIF.UpdateDITask | Updates an existing data integration task | @DITaskKey | Yes | Key value for the DITask |
| @SourceDatasetKey | Yes | Key value for the source dataset |
| @DestinationDatasetKey | Yes | Key value for the destination dataset |
| @LoadType | Yes | Identifies the LoadType record |
| @DITaskSourceFilterLogic | No | Used to filter the source dataset. Example "and City = ‘Atlanta’" |
| @DITaskWaterMarkLogic | No | Used to execute incremental extracts from the source dataset. Example "and ETLEditDate > %MinWaterMark% and ETLEditDate <= %MaxWaterMark%" |
| @DITaskEnabled | No | Defaults to 1 |
| @SourceFilterLogicIsEnabled | No | Determines if the SourceFilterLogic is active or not. Defaults to 0 |
| @NotebookPath | No | Specifies the path of the Databricks notebook |
| DIF.AddDIGroupTask | Assigns a data integration task to a data integration group | @DIGroupKey | Yes | Key value for the given DIGroup |
| @DITaskKey | Yes | Key value for the given DITask |
| @DIGroupTaskPriorityOrder | No | Sets the order and precedence of the various tasks in each DIGroup. Defaults to 1 |
| DIF.UpdateDIGroupTask | Updates the assignment of a data integration task within a data integration group | @DIGroupTaskKey | Yes | Key value for the given DIGroupTask |
| @DIGroupKey | Yes | Key value for the given DIGroup |
| @DITaskKey | Yes | Key value for the given DITask |
| @DIGroupTaskPriorityOrder | No | Sets the order and precedence of the various tasks in each DIGroup. Defaults to 1 |

## Initial Project Set-up

To set up each new Project within the DIF, use the stored procedures listed below from the DIF database. These stored procedures are ordered to follow the data flow needed to conform to the various PK/FK relationships in these config/meta tables.

1. Add Project

DECLARE @PK SMALLINT

EXEC [DIF].[AddProject]

@ProjectType = 'Key2'

,@ProjectName = 'Key2-Project-SQLServerToDataLake'

,@ProjectKey = @PK OUTPUT

SELECT @PK

1. Add Pipeline

DECLARE @PK SMALLINT

EXEC [DIF].[AddPipeline]

@ProjectName = 'Key2-Project-SQLServerToDataLake'

,@PipelineShortName = 'Test Pipeline'

,@PipelineFullName = 'PL\_TEST\_PIPELINE'

,@PipelineFolder = 'PL\_TEST'

,@PipelineDescription = 'This is a test of the AddPipeline stored proc'

,@PipelineKey = @PK OUTPUT

SELECT @PK

1. Add DIGroup

DECLARE @DK SMALLINT

EXEC [DIF].[AddDIGroup]

@DIGroupName = 'Group Test'

,@PipelineKey = 12

,@DIGroupIsEnabled = 1

,@DIGroupKey = @DK OUTPUT

SELECT @DK

1. Add Dataset – This will need to be done in source/destination pairs

DECLARE @DK SMALLINT

EXEC [DIF].[AddDataset]

@SystemName = 'KEY2SQLSQLDEV001'

,@RepositoryName = 'KEY2DB\_DEV'

,@DatasetClass = 'Dimension'

,@StorageType = 'Relational Table'

,@PartitionType = 'None'

,@DatasetNameSpace = 'dbo'

,@DatasetName = 'customer'

,@DatasetDescription = 'Table of Key2 customers'

,@DatasetPath = 'dbo.customer'

,@DatasetExternalVersionID = '1'

,@DatasetIsCreated = 1

,@DatasetIsEnabled = 1

,@DatasetKey = @DK OUTPUT

SELECT @DK

1. Add Attribute – Add as many Attribute records as needed to support the various functions like tracking partition and high watermark columns

DECLARE @AK INT

EXEC [DIF].[AddAttribute]

@DatasetKey = 1

,@DataType = 'int'

,@AttributeName = 'TestColumn'

,@AttributeSequenceNumber = 1

,@AttributeMaxLength = null

,@AttributePrecision = null

,@AttributeScale = null

,@AttributeIsNullable = 0

,@AttributeIsPrimaryKey = 1

,@AttributeIsUnique = 1

,@AttributeIsForeignKey = 0

,@AttributeIsWaterMark = 1

,@AttributePartitionKeyOrder = null

,@AttributeDistributionKeyOrder = null

,@AttributeIsEnabled = 1

,@AttributeKey = @AK OUTPUT

SELECT @AK

1. Add DITask

DECLARE @DK INT

EXEC [DIF].[AddDITask]

@SourceDatasetKey = 1

,@DestinationDatasetKey = 2

,@LoadType = 'Full Only'

,@DITaskSourceFilterLogic = null

,@DITaskWaterMarkLogic = null

,@DITaskEnabled = 1

,@SourceFilterLogicIsEnabled = 1

,@DITask = @DK OUTPUT

SELECT @DK

1. Add DIGroupTask

EXEC [DIF].[AddDIGroupTask]

@DIGroupKey = 1

,@DITaskKey = 1

,@DIGroupTaskPriorityOrder = 1

1. Validate the Configurations

EXEC [DIF].[ValidateDIGroupConfigurations]

@DIGroupName = 'Config Test'

The template ADF pipelines created for the ingestion framework use the information in the configuration tables to understand what data integration tasks need to be completed when executed. The PL\_DIF\_[DescriptiveName]\_GroupOrder template ADF pipeline accepts two parameters when it is executed:

* ProjectName: the value passed in must exist as a project in the config.Project table
* DIGroupName: the value passed in must exist as a data integration group in the config.DIGroup table

Based on the parameter values provided, the pipeline will look up all of the corresponding execution groups defined by the various DIGroupTaskPriorityOrder values to determine the necessary execution order for the DITasks. The pipeline will then call the PL\_DIF\_[DescriptiveName]\_TaskLoop child pipeline. This pipeline will spawn multiple threads, one for each DITask in a given DIGroupTaskPriorityOrder execution group. Lastly, the PL\_DIF\_[DescriptiveName]\_Ingestion pipeline is called. This pipeline looks up the specific information related to the source and destination datasets and performs the copy operation needed to ingest that data.

Based on this, to use the ingestion framework correctly, entries must be made across several tables.

Returning to the example started above, suppose the following:

* A new project named Project X requires the data from the source described above to be ingested into the destination described above
* Project X plans on ingesting additional tables in groups that will include several reference dimension tables, several member demographics tables, and several fact tables

The Project X team should use the stored procedures listed above to generate the following configuration table entries:

* An entry in the config.Project table for their **Project X** effort
* Three entries in the config.DIGroup table:
  + One to describe the **Dimensions** group of data integration tasks
  + A second to describe the **Member Demographics** group of data integration tasks
  + A third to describe the **Fact Table** group of data integration tasks
* Entries in the config.DITask table to describe each of the source-to-destination data ingestion tasks required for their project:
  + One to describe the ingestion of **Dim.MemberDemographics** as explained above
  + Others to describe additional data integration tasks as required
* Entries in the config.DIGroupTask table to assign each data integration task added to the config.DITask table to one of the groups added to the config.DIGroup table
  + One to assign the task for ingesting the **Dim.MemberDemographics** table to the **Member Demographics** group
  + Others to assign additional tasks to different data integration groups

***Note****: While the stored procedures listed above can be used to add or update entries in each of the configuration tables, it is also acceptable to insert or update entries directly using SQL statements, ADF pipelines, or any other mechanism that supports writing information to an Azure SQL database. Care should be taken, however, to ensure that any updates applied do not affect other users dependent on existing configuration information.*

## Ingestion Scenarios

The Data Ingestion Framework supports several different ingestion scenarios that allow for flexibility when pulling data from the given source. Each of these scenarios can be configured by setting up certain values, flags, and metadata records across the various DIF tables.

### Partition

Partitioning is a common practice used to achieve performance gains and improve maintenance activities on large datasets. If the given source data uses partitioning, this can be leveraged in the DIF to increase the performance of the extract so that it runs in parallel based on the partitioned value.

**Note**: *This functionality is only available on Full or the initial extract of a Full then Incremental load type.*

Setup Steps:

1. Determine which Partition Type the source dataset follows by checking the [reference].[PartitionType] table
2. Verify that the partition values for that type are defined in the [config].[PartitionBoundary] table matches the partitioning scheme of the source data
3. Create the appropriate source record in the [metadata].[Dataset] table and set the [PartitionTypeKey] column to the corresponding key value
4. Add a record in the [metadata].[Attribute] table that matches the partitioned column in the source. Set the [AttributePartitionKeyOrder] column to 1

### Source Filtering

In some cases, it will be necessary to apply filters to the source data. This might be needed to limit the amount of data returned to the last two years based on a business date or to exclude certain record types that are not necessary for downstream activities. To add filtering to an ingestion task in the DIF database, set a value in the [DITaskSourceFilterLogic] column in the [config].[DITask] table. The value entered in this column should be a statement that could be appended as an additional predicate of a WHERE clause. The [SourceFilterLogicIsEnabled] column value should also be set to 1. If at some point the source filter logic isn’t needed for a given ingestion process, that value can be set to 0 to disable the application of that logic against the source dataset.

Example:

To filter the source data to only return records after a certain effective date, enter the following value in the [DITaskSourceFilterLogic] column.

*“AND EffectiveDate > ‘12/31/2013’”*

### Source Audit Query

The SourceAuditQuery column in the config.DITask table can be used to track dataset counts in the TableCopy pipeline. When this value is set, it collects the record count defined in the column against the source and stores that in the logging.PipelineResult table.

Example SourceAuditQuery Value:

SELECT COUNT\_BIG(\*) AS [SrcAuditRowCount]

FROM <SourceTableName>

WHERE 1=1

### High Watermark Tracking – Incremental Extracts

Tracking a high watermark value is essential for any incremental extract activities. This is where the MAX value of a given attribute is recorded from the previous extract to be used as the starting point of the next extract. The DIF process can be configured to track these types of values and apply them during Incremental load types.

Setup Steps:

1. Determine which source attribute is used to track new and changed records. The data type of this attribute will be needed as well.
2. After the source dataset record has been created in the [metadata].[Dataset] table, add a new record in the [metadata].[Attribute] table based on the attribute from the source environment that tracks changes
3. Set the appropriate [DataTypeKey] value for this new [metadata].[Attribute] record
4. Set the [AttributeIsWaterMark] value to 1
5. In the [config].[DITask] table, set the [DITaskMaxWaterMark] column to the expression needed to track the high watermark value. This expression will be added to the WHERE clause as an additional predicate.
6. Be sure to include the %MinWaterMark% and %MaxWaterMark% replaceable parameters in the expression

Example:

*“and ModifiedDateTime > %MinWaterMark% and ModifiedDateTime <= %MaxWaterMark%”*

### Custom Source Query

In some ingestion scenarios, the logic needed to extract the source data goes beyond the standard filtering or high watermark processes created in the DIF. To support these more complex use cases, a custom source query can be defined. This allows for more involved table joins and filtering on the source data set.

**Note:** *This custom source query should only be used if the source data cannot be extracted using the standard filtering and high watermark capabilities of the ingestion framework.*

One example of a scenario that would require a custom source query is if the value that tracks the high watermark is stored in a separate table in the source environment. In that case, a query with a join between the base source table and the supporting table would be needed to perform this extract.

Setup Steps:

1. Define the source config record in the [metadata].[Dataset] table as you would with any other source
2. Add a record to the [config].[DITask] table for the given source and destination datasets
3. Set the [SourceOverrideQuery] value in this DITask record to the query needed to pull this source data. The example query below can be used as a guide.
4. This query value will be used both as the source query and as the way to determine the MAX high watermark value. The columns defined in the SELECT portion of the query will be replaced with a MAX(<WaterMarkAttribute>) statement that is generated based on the value from the [metadata].[Attribute] table that has the [AttributeIsWaterMark] column set to 1.

**Example Customer Source Query:**

SELECT A.\*

FROM DIF.TestSQLIncrementalBatch A--Base table referenced in the DIF.Dataset config record.

INNER JOIN DIF.TestSQLExtractBatch B--Additional table used to determine high watermark from the base table

ON A.BatchID = B. BatchID

WHERE 1=1

### DIGroupTask Setup

The config.DIGroupTask table controls how the different ingestion tasks are grouped and orchestrated. Each DITask record is associated with a DIGroup and a DIGroupTaskPriorityOrder value is set to determine the execution order and precedence of that task. All tasks in each DIGroupTaskPriorityOrder must complete before the next set of tasks is processed. In the example below, all those tasks in DIGroupTaskPriorityOrder 1 will complete before going on to those in group 2.

The exception to this ordering and precedence is for the record below that has a DIGroupTaskPriorityOrder of 0. That is a “related” task. These related tasks should be kicked off immediately after their corresponding task completes. In the example below, the “property” task in priority order #1 has a related task that should run once it completes. That related task has a priority order of 0. The scenario that these two tasks are illustrating is where the task with a priority order of 1 is the source to bronze ingestion. Once that is complete, it needs to immediately kick off the bronze to silver task that is related to that first step. Rather than waiting for all tasks in the prior order group #1 to finish, setting this related task value allows for that second step to start right away.

Tasks with a priority order of 0 are not picked up by the GroupOrder pipeline. Only those starting at 1 will be considered. All tasks with a priority order of 0 must be linked as a related task to be executed.

See the example below:

**Configuration Validation**

Once the configuration records have been added to the various DIF tables, the DIF.ValidateDIGroupConfigurations stored procedure can be executed to ensure that there are no issues with the current group setup. The stored procedure will return either an “all-clear” message or a series of statements indicating what configuration settings are incomplete.

EXEC [DIF].[ValidateDIGroupConfigurations]

@DIGroupName = '<DIGroupName>'

# Pipeline Logging

The template ADF pipelines created as part of the ingestion framework provide logging of the start and end of each pipeline run. Additional logging activities can be included in the pipelines to log other information as desired. The following table lists the four logging tables that are available in the DIF database which can be used to capture logging information:

|  |  |
| --- | --- |
| Logging Table Name | Description |
| logging.PipelineRun | Logs the start and end of a pipeline execution |
| logging.PipelineActivity | Logs the start and end of a specific activity within a pipeline |
| logging.PipelineParameter | Logs the values of one or more parameters within a pipeline |
| logging.PipelineResult | Logs results that are captured within the pipeline (row counts, max values, distinct counts, etc.) |

These four logging tables rely on two additional reference tables that contain pipeline status and pipeline result names. The following table lists those reference tables and the current table entries as of the writing of this document:

|  |  |  |
| --- | --- | --- |
| Reference Table Name | Description | Available Values |
| metadata.PipelineStatus | Contains a list of standard pipeline status values | InProgress |
| Succeeded |
| Failed |
| Canceled |
| Canceling |
| Queued |
| metadata.PipelineResultName | Contains a list of standard pipeline result names; any pipeline result names that have the PipelineResultNameIsStandard flag set to 1 should be considered a standard result that every pipeline should record if possible | RowsRead |
| RowsCopied |
| CopyDuration |
| Throughput |
| HighWaterMark |

To populate each of the four logging tables, a set of stored procedures can be used to add and update table entries. The following table describes each of these stored procedures, including details regarding the parameters that can be used for each:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stored Procedure | Description | Parameter | Required | Usage |
| DIF.StartPipelineRun | Adds a new pipeline execution log record (starts logging) |  |  |  |
| DIF.EndPipelineRun | Updates a pipeline execution log record (ends logging) |  |  |  |
| DIF.StartPipelineActivity | Adds a new pipeline activity execution log record |  |  |  |
| DIF.EndPipelineActivity | Updates a pipeline activity execution log record |  |  |  |
| DIF.LogPipelineParameter | Adds a pipeline parameter log record |  |  |  |
| DIF.LogPipelineResult | Adds a pipeline result log record |  |  |  |
| DIF.AddPipelineResultMultiple | For a given PipelineRun and PipelineActivity, this store procedure will pass in a delimited list of values to be logged |  |  |  |

As mentioned above, the logging tables provide a mechanism for data ingestion engineers to log information from within their specific pipelines. The template ADF pipelines provided as part of the ingestion framework include activities that call several of the stored procedures above. As data engineers expand on the template pipelines or create their own, additional stored procedures can be used to include additional logging.

***Note****: While the stored procedures listed above can be used to add or update entries in each of the configuration tables, it is also acceptable to insert or update entries directly using SQL statements, ADF pipelines, or any other mechanism that supports writing information to an Azure SQL database. Care should be taken, however, to ensure that any updates applied do not affect other users dependent on existing configuration information.*