



User Guide

The Any to Any Data Replication Engine.

Released On: 10/14/2022

This documentation provides detailed configuration settings, options, and technical specifications for configuring data replication jobs across data sources.

Contents

Introduction	5
Concepts and Terminology	5
Data Sync Agents	5
Job Readers and Job Writers Overview	6
Synthetic CDC.....	7
Direct Sync Operations	7
Job Types.....	8
Multicasting	8
Connection Strings	8
Database & ODBC Connections	9
SQL Server Connection.....	9
MySQL Connection.....	10
Oracle Connection.....	10
Teradata Connection.....	11
Generic ODBC Connection	11
Drive & FTP Connections.....	11
Local Path	11
AWS S3 Bucket	12
Azure Blob Container	12
FTP Site.....	12
HTTP/S REST Connections	12
API Key	13
Basic Authentication	13
Token Authentication	14
Bearer Authentication.....	14
Messaging Connections	16
AWS SQS.....	16
Azure Event Hub	16
Azure Service Bus	17
Google PubSub.....	17
Kafka.....	17
MSMQ	17

RabbitMQ.....	18
Big Data/NoSQL Connections.....	18
Google BigQuery	18
CosmosDB Databases.....	18
Job Readers	18
Create a Sync Job	18
Database Job Reader	21
Change Data Capture	22
Change Data Capture	22
Change Tracking.....	23
Overriding the Deleted Records Identification Method	24
Using the Timestamp / DateTime Column.....	26
Drive Job Reader (Parquet, XML, JSON, CSV).....	27
Reading CSV Files	27
HTTP/S REST API Job Reader	28
JSON/XML Column Transformation	29
Dynamic Parameters.....	29
XML Example.....	31
JSON Document Example.....	32
JSON Array Example.....	33
Messaging Consumers	34
Pointer Values: Editing the High Watermark.....	35
Job Writers.....	36
Create a Job Writer from An Existing Job Reader	36
Database Targets	39
SQL Server (Native Provider).....	39
ODBC Drivers & Enzo Server	39
Sending an XML/JSON Document.....	40
One SQL Command per Record	40
One SQL Command per Batch.....	41
Create Target DDL Scripts	42
Known Targets	42
Generic Targets.....	43

Preview SQL Commands	44
HTTP/S Targets.....	44
Messaging Targets	45
Drive Targets	45
Common Write Options	45
Shadow Copy.....	46
Create New Files	46
Update Existing Files (Data Distribution/Bucketization).....	46
Date Replacement Tokens	47
Parquet Files.....	47
JSON Files	48
XML Files	48
CSV Formatting Options.....	48
Big Data/NoSQL Targets.....	49
Google Big Query	49
CosmosDB	50
Automatic JSON Document Generation	50
Build a Custom JSON Document	51
Data Sync Files	52
Replay Sync Files	52
Resync Data.....	53
Full Sync Files	54
Job Triggers	54
Logging	55
Execution Log	55
Execution History	56
Data Pipeline	57
Data Filter.....	58
Data Hashing	59
Data Masking	59
Data Quality	60
Dynamic Data Column	61
SQL Expression	61

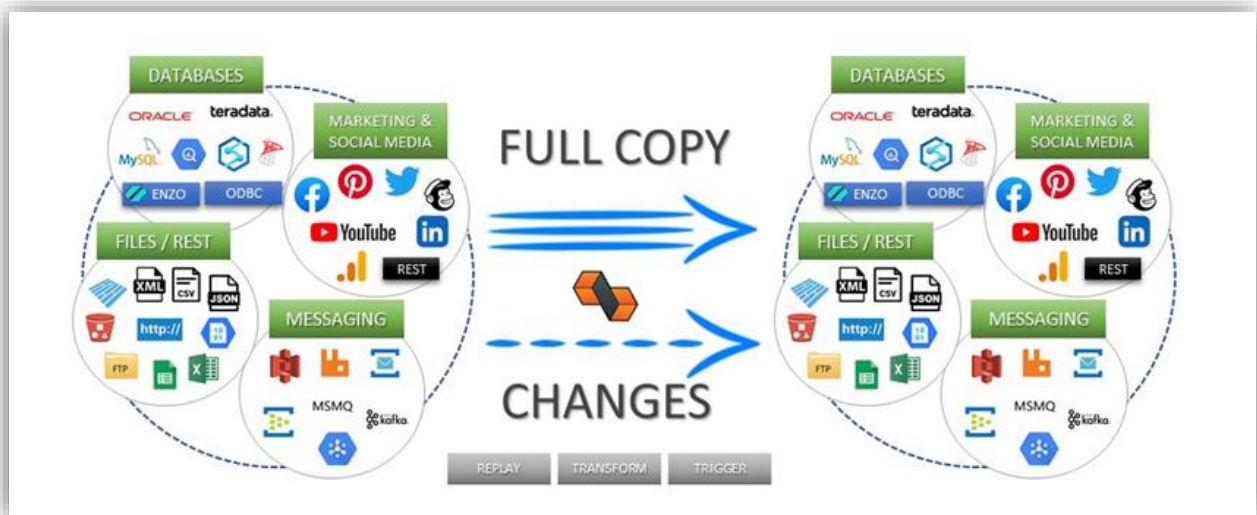
Enzo Expression	62
Remove Column	62
DataZen Functions	62
Nested Functions	64
Uses	64
Enzo HTTP Functions	64
Example 1: Unauthenticated HTTP Get	65
Example 2: Unauthenticated HTTP Post with Headers	65
Example 3: Authenticated HTTP Get	65
Sync Agent REST API	66
Appendix	68
Limitations	68
Target DDL Script	68
Upsert Script	68
Data Replication	68
Transactional Consistency Considerations	68
Data Type Conversions	69
Oracle to SQL Server Data Mapping	69
MySQL to SQL Server Data Mapping	69

Introduction

DataZen is a data replication platform that allows you to copy data from any source system into any target system. DataZen supports the following use cases:

- Copy all records from any source system into one or more target systems
- Identify changes made to a source system and forward them to one or more target systems
- Listen for messages and forward them to any target system
- Use JSON, XML, CSV, or Parquet files as data sources or target files

The following diagram depicts the various scenarios that are possible with DataZen. Because DataZen can inspect messages, perform relevant message conversions, and supports a large number of authentication mechanisms, it can forward full record sets or only the identified changes in the correct target format.



Concepts and Terminology

Let's start by introducing a few technologies and concepts related to data replication that apply to DataZen.

Data Sync Agents

A Data Sync Agent is needed to read from and/or write to a system. It is best to install a Data Sync Agent close (for optimum network bandwidth reasons) to the systems that the agent will read from or write to.

For example, if you have a source system (ex: an Oracle database) in New York, and want to replicate changes to a Kafka endpoint in Atlanta, you would likely install two agents: one in each location. However, it is possible to use a single agent to perform both the read and write operation for most implementations.

The Data Sync Agent is made of a few components:

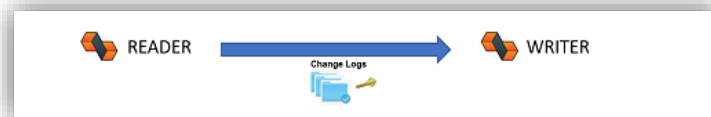
- Job Readers – read data from any source system
- Job Writers – write data to any target system
- CDC Engine – internal engine that automatically detects changes (Synthetic CDC)
- Staging Store – stores configuration settings and state information of the source data
- HTTP/S Listener – a controller listener that DataZen Manager uses to manage the agent
- Sync File – a Universal Change Log that can be copied and played back

Data Sync Agents require a connection string to a Staging database stored in SQL Server, as specified in the Installation instructions. Each agent is licensed separately.

Job Readers and Job Writers Overview

Job Readers and Job Writers are designed to communicate to source and target systems supported by DataZen. A Job Reader Job fetches data from the source system at the specified interval, when started manually, or automatically for messaging consumers. Job Readers then forward data extracted from the source system as-is, or forwards the data to the DataZen Change Data Capture (CDC) Engine to extract changes automatically (a.k.a. **Synthetic CDC**). If a change is detected in the source data, a Sync File is produced in the specified output directory.

Change Logs hold the changes that were detected from the source system and stored in a universal format so that it can be played back on any target system. When first created, the Change Log may contain all the records from the source system. To learn more about Sync Files, see the Data Sync Files section.



Job Writers start on a schedule and inspect a shared folder for new Sync Files; they can also be started upon completion of a Job Reader using a Job Trigger. When a new Change Log has been detected, the Job Writer extracts the data found in the log, converts each record in the format expected by the target system, and executes the necessary command(s) against the target system.

When the target system is a relational database, an Upsert operation is either an Insert or an Update operation depending on whether the record is found in the target system or not based on the Key Columns specified.

This architecture allows Job Readers and Job Writers to reside on entirely different networks, with only a network share or cloud folder (or an FTP site) in common. Because the Change Log can also be encrypted using PGP, the Change Log can also be stored on any public cloud platform.

Synthetic CDC

DataZen has a built-in data differential engine that can quickly identify net changes of data sources over time. This feature is designed to extract changes from source systems that do not offer a change stream or that do not have a mechanism to keep a high watermark pointer.

DataZen triggers the internal synthetic CDC identification automatically based on the type of source system and options selected for the job. Generally speaking, this differential engine tracks the signature of source records in an internal staging database and compares them over time as records are being extracted from the source system.

The synthetic CDC engine is bypassed in the following scenarios:

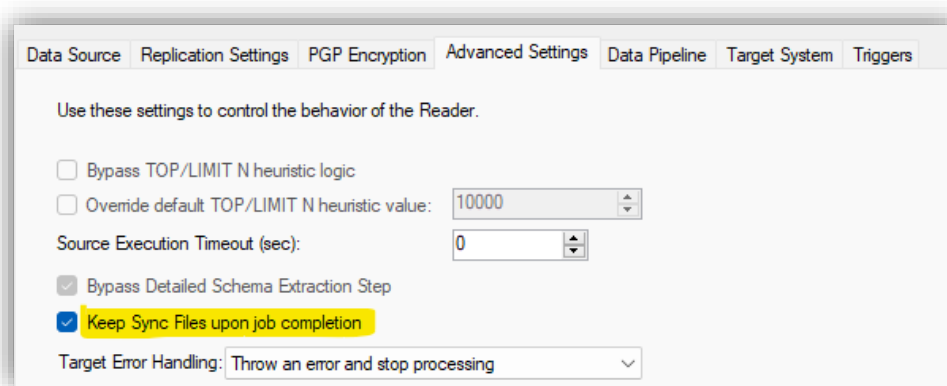
- the source system is a CDC data source itself (such as SQL Server CDC or Change Tracking)
- the job reader does not specify a Key Column

When the Synthetic CDC option is bypassed, all records detected from the source system are forwarded to the Sync File.

Direct Sync Operations

While Job Readers and Job Writers can be created individually for maximum reuse and multi-casting, it is possible to create a single job that performs both Read and Write operations. A Direct Sync operation creates Sync Files behind the scenes; these files can be automatically deleted if desired.

The Advanced Settings tab allows you to control whether sync files are deleted upon completion:



Use these settings to control the behavior of the Reader.

☐ Bypass TOP/LIMIT N heuristic logic

☐ Override default TOP/LIMIT N heuristic value: 10000

Source Execution Timeout (sec): 0

☒ Bypass Detailed Schema Extraction Step

☒ Keep Sync Files upon job completion

Target Error Handling: Throw an error and stop processing

Direct Sync Jobs do not offer PGP encryption

Job Types

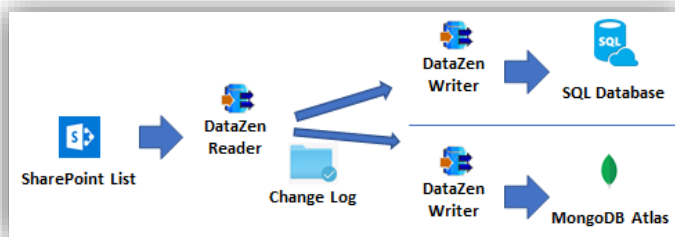
Once a Data Sync agent has been configured and registered in DataZen Manager, you can start managing Job Readers and Job Writers. The following types of jobs can be created:

- **Data Sync Job** – a reader (and optionally a writer) that can access any source system except messaging platforms
- **Sync Writer Job** – a writer that reads from a Sync File
- **Messaging Consumer** – a reader that listens from a messaging platform (and optionally a writer)
- **Passthrough Consumer** – a simpler reader that forwards messages from one messaging platform to another

Multicasting

This allows you to replicate data to multiple systems if the Job Reader being used as the source does not have a Job Writer specified as part of its definition; in other words, the Target System for the Job Reader must be set to None.

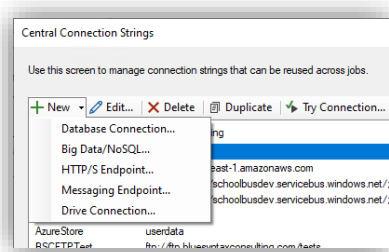
For example, the source system could be a SharePoint List; you can create two Job Writers pointing to the same folder where the Sync Files are located, and replicate data to a SQL Server database and MongoDB Atlas database using an ODBC Driver or Enzo Server.



Connection Strings

Before creating replication jobs, it is necessary to first configure connection strings. Connection strings help you define connection settings for all the necessary source and target endpoints for all supported data sources, including Cloud Drives & FTP sites, Databases & ODBC drivers, Messaging endpoints, and HTTP/S endpoints.

To manage connections, select the desired agent and choose **Configuration -> Central Connection Strings**. From this screen you can add, duplicate, delete or edit connection strings.



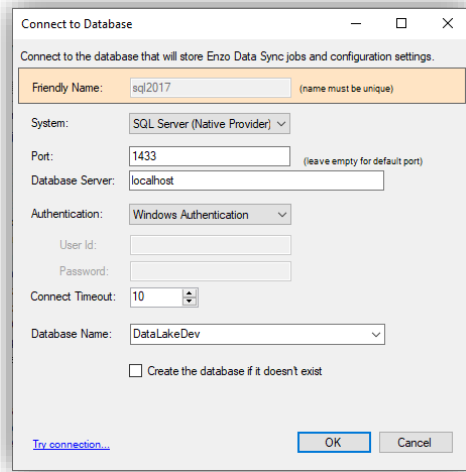
To create a connection, select **New** and choose the type of connection you would like to create.

To edit a connection, double-click on it.

Database & ODBC Connections

Database connections typically require an ODBC driver installed on the computer where the agent is installed, and on the server where the Manager is installed. Specific options are available for SQL Server, Oracle, MySQL, Teradata, and a more generic configuration screen for other ODBC drivers.

SQL Server Connection



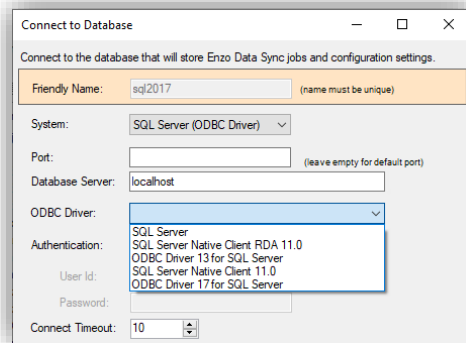
The screenshot shows the 'Connect to Database' dialog box. The 'Friendly Name' is 'sql2017'. The 'System' is 'SQL Server (Native Provider)'. The 'Port' is '1433'. The 'Database Server' is 'localhost'. The 'Authentication' is 'Windows Authentication'. The 'User Id' and 'Password' fields are empty. The 'Connect Timeout' is '10'. The 'Database Name' is 'DataLakeDev'. There is a checkbox for 'Create the database if it doesn't exist' which is unchecked. At the bottom, there are buttons for 'Try connection...', 'OK', and 'Cancel'.

Choose SQL Server (Native Provider) when connecting to an on-premise SQL Server database, Azure SQL Server or for Enzo Server.

Blank out the Port number to use the default port.

If no database is provided in the dropdown list, type the database name to connect to.

Click [Try Connection...](#) to make sure the credentials are correct. Then click OK.

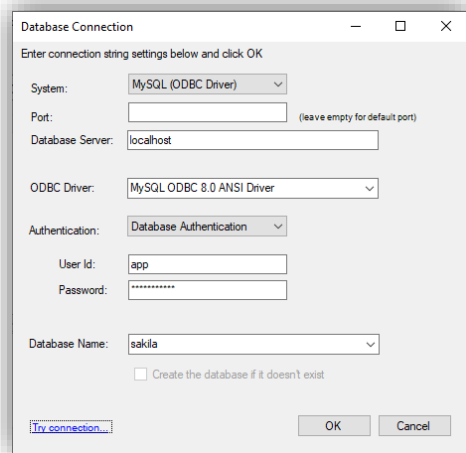


The screenshot shows the 'Connect to Database' dialog box. The 'Friendly Name' is 'sql2017'. The 'System' is 'SQL Server (ODBC Driver)'. The 'Port' is empty. The 'Database Server' is 'localhost'. The 'ODBC Driver' dropdown is open, showing a list of drivers: 'SQL Server', 'SQL Server Native Client RDA 11.0', 'ODBC Driver 13 for SQL Server', 'SQL Server Native Client 11.0', and 'ODBC Driver 17 for SQL Server'. The 'Authentication' is 'Windows Authentication'. The 'User Id' and 'Password' fields are empty. The 'Connect Timeout' is '10'. At the bottom, there are buttons for 'Try connection...', 'OK', and 'Cancel'.

Choose SQL Server (ODBC Driver) if you would like to use a specific SQL Server driver installed on your machine. An ODBC Driver list will be displayed showing you all available 64-bit drivers.

Note: Only 64-bit ODBC drivers can be used.

MySQL Connection



To connect to a MySQL database, choose My SQL (ODBC Driver). You must have installed a valid 64-bit MySQL ODBC driver separately.

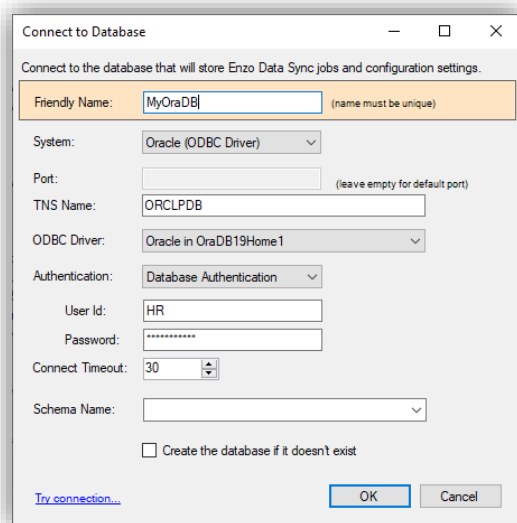
Note: Only 64-bit ODBC drivers can be used.

Blank out the Port number to use the default port.

If no database is provided in the dropdown list, type the database name to connect to.

Click [Try Connection...](#) to make sure the credentials are correct. Then click OK.

Oracle Connection



To create an Oracle connection, choose the Oracle (ODBC Driver) option.

Note: Only 64-bit ODBC drivers can be used.

Select the correct Oracle ODBC driver (must be installed separately) from the list and enter your database credentials.

Click [Try Connection...](#) to make sure the credentials are correct. Then click OK.

Teradata Connection

To connect to Teradata, select Teradata (ODBC Driver) from the list of options.

Note: Only 64-bit ODBC drivers can be used.

Blank out the Port number to use the default port.

Select the Teradata driver (must be installed separately) and enter your database credentials.

Click [Try Connection...](#) to make sure the credentials are correct. Then click OK.

Generic ODBC Connection

To use a generic driver, such as a Simba ODBC driver, choose Other (ODBC Driver). You can either enter the full connection string here, or enter a DSN name you have created using the ODBC Manager (64-bit).

Note: Only 64-bit ODBC drivers can be used.

Click [Try Connection...](#) to make sure the credentials are correct. Then click OK.

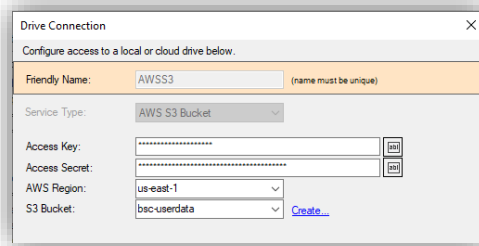
Drive & FTP Connections

Drives and FTP connections allow you to replicate files (XML, JSON, CSV and Parquet) stored locally, on cloud drives, or on FTP servers. In addition, these connections can be used to share Sync Files between Job Readers and Job Writers.

Local Path

Choose Local Path when the files are located on a local drive or a UNC path. If the path requires credentials, specify them in the Login as user and password fields.

AWS S3 Bucket



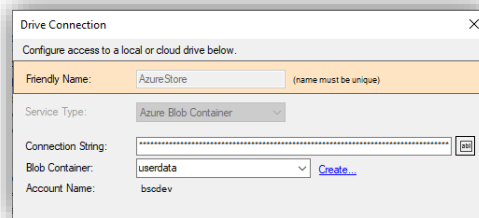
The 'Drive Connection' dialog box is shown with the following fields and values:

- Friendly Name:** AWSS3 (name must be unique)
- Service Type:** AWS S3 Bucket
- Access Key:** [Redacted]
- Access Secret:** [Redacted]
- AWS Region:** us-east-1
- S3 Bucket:** bsc-userdata
- Create...** button

Choose AWS S3 Bucket when the files are located on the Amazon Web Service platform. Both an Access Key and Access Secret key is required. Select or type one of the available regions, then enter the S3 Bucket to use.

When the keys are valid, you can use the Dropdown to view all available buckets, and create a bucket if it doesn't exist.

Azure Blob Container

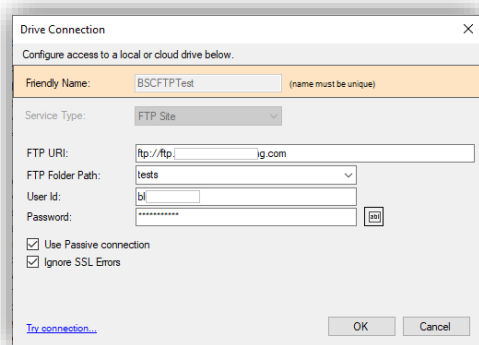


The 'Drive Connection' dialog box is shown with the following fields and values:

- Friendly Name:** AzureStore (name must be unique)
- Service Type:** Azure Blob Container
- Connection String:** [Redacted]
- Blob Container:** userdata
- Account Name:** bscdev
- Create...** button

Choose Azure Blob Container, and specify an Azure Connection String; you can create a Blob Container from the screen if needed, or select a default container from the dropdown box.

FTP Site



The 'Drive Connection' dialog box is shown with the following fields and values:

- Friendly Name:** BSCFTPTest (name must be unique)
- Service Type:** FTP Site
- FTP URI:** ftp://ftp.ig.com
- FTP Folder Path:** tests
- User Id:** bl
- Password:** [Redacted]
- ☒ Use Passive connection
- ☒ Ignore SSL Errors
- Try connection...** button
- OK** and **Cancel** buttons

Select FTP Site to connect to an FTP server.

Enter the full FTP URI, starting with ftp://...

The Folder Path must be entered separately and start at the root of the FTP site.

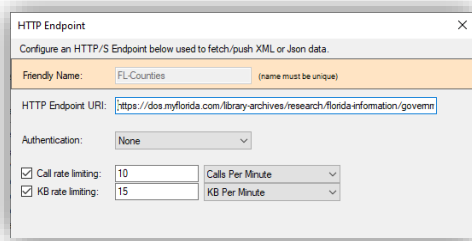
Enter the User ID and Password to use, and check the Use Passive Connection if needed.

*The **Ignore SSL Errors** option is currently experimental; its purpose is to eliminate SSL errors from FTP sites using older or outdated SSL certificates.*

Click [Try Connection...](#) to make sure the credentials are correct. Then click OK.

HTTP/S REST Connections

Use this connection type when accessing REST API endpoints, such as Twitter, Facebook, Google or any other cloud service directly. All HTTP/S connections require an endpoint, an authentication mechanism, and offer Call Rate and Kilobyte Rate limiting options to limit the speed of the calls.



Enter a name for the connection, and a complete HTTP URL.

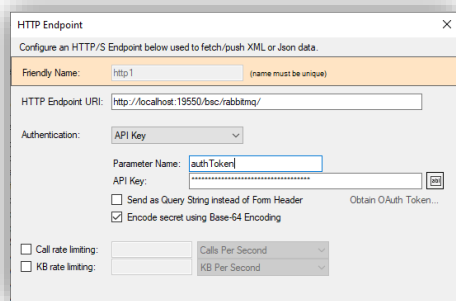
When no authentication is required, select None in the Authentication option.

You may enter rate limiting for both the frequency of the calls and the amount of data retrieved from the service within a time window. In this example, we have set a limit of up to 10 calls per minute and no more than 15KB per minute. When either of these thresholds is reached, the job will slow down.

Click [Try Connection...](#) to make sure the credentials are correct. Then click OK.

The following section explains how to setup various authentication options.

API Key



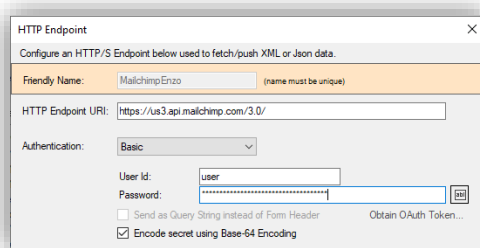
Some services require API Key authentication. This is usually done using a custom header key/value pair, or sent as a Query Parameter added to the URL.

This example shows you how to setup an HTTP connection to a local Enzo Server to call the RabbitMQ adapter.

To send the API Key as a query parameter, check the **Send as Query String** option.

The encode the key using Base-64 encoding, select the **Encode secret using Base-64 Encoding** option.

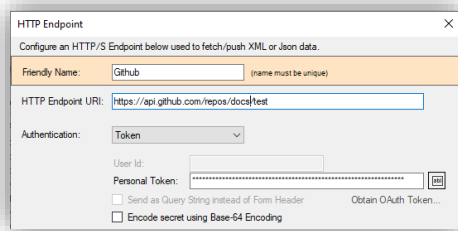
Basic Authentication



Some services require Basic Authentication. This is usually done using a User Id and Password, and hashed into a single value as part of the Authentication header of a request.

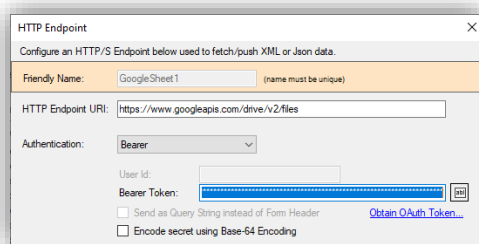
This example shows you how to setup an HTTP connection to the Mailchimp v3.0 API using Basic Authentication.

Token Authentication



Some services require Token Authentication. This is usually done by first obtaining a Personal Access Token (PAT), and added as part of the Authentication header of a request using the Token authentication scheme.

Bearer Authentication



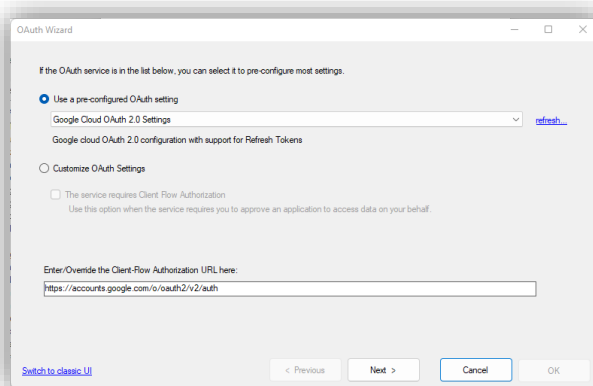
Some services require Bearer Authentication. This typically requires obtaining an OAuth 2.0 token. If you have the token already available, simply paste it in the Bearer Token field.

To obtain an OAuth token or a Refresh Token, click on the **Obtain OAuth Token...** link.

Certain services (such as Twitter) generate a Bearer token directly, and/or the Bearer Token never expires. In these cases, you may choose to paste the Bearer Token into the Bearer Token directly without using the OAuth Token wizard.

DataZen supports OAuth 2.0 with or without Refresh Tokens and allows you to customize the requests made to regenerate tokens automatically. The OAuth Wizard also supports client-flow authentication and allows you to authorize DataZen directly.

To generate a new Bearer Token click on the **Obtain OAuth Token** link and follow these steps:



If the service you would like to configure is listed in the pre-configuration list, select it. In this case, the Google Cloud OAuth 2.0 option is selected. Because this service requires a client-flow authorization, a Token URL is automatically set below.

In some cases, you may need to modify the URL to include an Account ID or Tenant ID, as needed.

If you want to connect to another endpoint, not found in the pre-configured list, choose Customize OAuth Settings. Select whether or not the service requires client-flow authorization.

Click Next.

Enter the necessary information below and click Next.

Client Parameter and Id:

This service does not require a client flow authentication. Please enter your Client ID or API Key for the 'client_id' field and click Next.
If this service doesn't require a client id or API Key, leave the field blank and click Next.

[Switch to classic UI](#) < Previous Next > Cancel OK

Without Client-Flow Authentication

If the service does not require client-flow authentication, a simpler screen will be displayed in which you can enter your client id, which will be used in the following screen.

Enter the necessary information below and click Next.

Client Parameter and Id:

Redirect URL:

Response Type:

Scopes:

[Start Client Authorization Flow...](#)

Authorization Code (access token):

Enter the necessary information and click on Start Client Authorization Flow to obtain an authorization code.

URL: <https://accounts.google.com/o/oauth2/v2/auth>

[Switch to classic UI](#) < Previous Next > Cancel OK

With Client-Flow Authentication

If the service requires client-flow authentication, this screen will display the necessary options to start the flow. Each parameter is customizable, including the parameter names and values.

Once the values are specified, click on Start Client Authorization Flow. This will open up your local browser and allow you to authentication and obtain a code (or access token).

NOTE: in most cases, DataZen will intercept the response and automatically extract the code. If necessary, you can also obtain the code from the browser window manually and paste it in the Authorization Code box. Keep in mind that these codes (or access tokens) expire relatively quickly.

Select or manually add the scopes that the authentication token will include.

Scope: [Add Scope](#)

[Remove](#)

Scope

<https://www.googleapis.com/auth/spreadsheets>

<https://www.googleapis.com/auth/drive.file>

<https://www.googleapis.com/auth/drive>

[Click here to view all the scopes provided by the service. \(opens a browser\)](#) OK Cancel

Most services require the use of Scopes, to determine what access level you require. When possible, DataZen selects a default scope for you; however, you should review the service documentation to ensure you select all the scopes you need to authorize DataZen to access the desired service endpoints.

To select the desired scopes or to open the URL that lists the scopes available (when possible), click on the ... next the scopes textbox. Please note that not all available scopes may be listed in the dropdown box.

Enter the Token URL below along with the Client Secret

Bearer Token Refresh Token Settings

Grant Type:

HTTP Method:

Authorization:

OAuth Bearer Token URL:

Enter the Token URL for the service

Secret Only:

Query Params:

Payload:

[Call Bearer Token endpoint](#)

☐ URL Encode ☐ URL Encode

Bearer Token:

Refresh Token:

[Switch to classic UI](#) < Previous Next > Cancel OK

This last step allows you to obtain a Bearer Token, and when possible/available a Refresh Token. If you selected a pre-configured endpoint on the first screen, these settings will be selected for you; just click on **Call Bearer Token Endpoint** to obtain your token(s). This call may only work once; access tokens generated in the previous step can only be used once.

Some services require the use of an additional user id in addition to a secret; for these endpoints, select User Id & Secret in the dropdown box; the UI will be changed to allow you to enter a user id value.

This screen allows you to change the majority of the settings to call the token endpoint, including the HTTP Verb used (GET, POST, PUT), and Authentication Mode (None,

The following replacement tokens are available for both the Query Parameters and the Payload:

{user_name} – the User Id provided on this screen
{client_secret} – the Secret value provided on this screen
{code} – the code obtained on the prior screen
{scope} – the list of scopes provided on the prior screen
{redirect_uri} – the redirect URI provided on the prior screen
{client_id} – the Client Id value provided on the prior screen
{grant_type} – the grant type entered on this screen
{token} – the current Bearer Token
{refresh_token} – the current Refresh Token

Basic) and the Grant Type for example. You can also customize the Query Parameters and the Payload.

When building a custom HTTP Query and/or Payload, you can either enter the necessary values directly, or use replacement tokens.

For example, the following queries are equivalent assuming the redirect URI provided in the previous screen was https://localhost:

```
redirect_uri=https://localhost
redirect_uri={redirect_uri}
```

When the service supports Refresh Tokens, you can further configure the way the refresh operation will be made. Click on the Refresh Token Settings tab.

Similarly, you can configure the grant type, the HTTP method, the authentication mode, and the URI to call along with the query and payload parameters.

If the refresh URI is left blank, the bearer token endpoint specified on the Bearer Token tab will be used.

NOTE: If your refresh token has expired, you may need to go back to the previous screen and generate a new access token.

Messaging Connections

DataZen can connect to a number of messaging platforms both as a Message Consumer and Message Publisher.

AWS SQS

To connect to an AWS SQS queue, choose the AWS SQS Service Type. Enter the AWS SQS service URL (usually similar to <https://sqs.AWS-REGION-NAME.amazonaws.com>), your access key, access secret, and your AWS Account ID.

You can specify a default Queue Name to be used by Jobs. Click on the ellipses to view available queues.

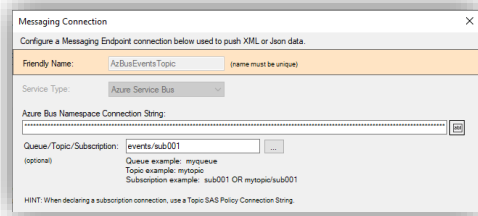
Azure Event Hub

To connect to an Azure Event Hub, choose the Azure Event Hub Service Type. Enter the **Event Hub Namespace** connection string.

WARNING: The Azure Portal also provides an Event Hub Connection string; it is recommended to use the Namespace connection string instead since it is not limited to a specific event hub.

You can specify a default Event Hub Name to be used by Jobs.

Azure Service Bus

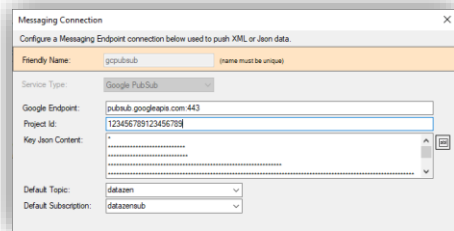


To connect to an Azure Service Bus, choose the Azure Service Bus Service Type. Enter the **Service Bus Namespace** connection string.

WARNING: The Azure Portal also provides specific Service Bus Connection string; it is recommended to use the Namespace connection string instead since it is not limited to a specific endpoint.

You can specify a default Queue, Topic, or Subscription to be used by Jobs.

Google PubSub

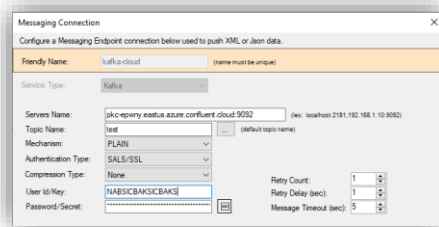


To connect to a Google PubSub service, choose the Google PubSub Service Type. Enter the PubSub endpoint, which is usually **pubsub.googleapis.com:443**.

Enter your Project ID and your entire OAuth 2.0 Client ID JSON file in the respective fields. You can download your Key JSON file from the Google Cloud portal.

You can specify a default Topic and Subscription to be used by Jobs.

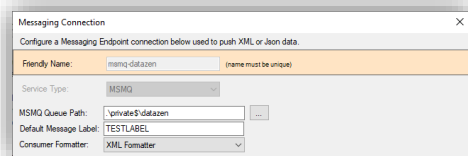
Kafka



To connect to a Kafka service, choose the Kafka Service Type. Enter the Kafka endpoint, which is similar to **pkc-epwmy.eastus.azure.confluent.cloud:9092** when connecting to a Confluent Kafka endpoint.

Enter the various connection settings, including your User Id and Secret. The Topic name is optional; you can view available topics by click on the ellipses.

MSMQ



To connect to a MSMQ queue, choose the MSMQ Service Type. Enter the MSMQ Queue Path. To connect to a local private queue, use this notation: **.\private\$myqueueuename**.

To locate queues on your network, click on the ellipses.

You can also specify the default formatter to use and a default message label when sending messages.

RabbitMQ

To connect to a RabbitMQ service, choose the RabbitMQ Service Type.

- Enter the host name where RabbitMQ is listening on
- To use the default exchange, leave the Exchange value blank
- You can choose specific exchanges, such as Direct, Fanout, or Topic
- Enter the User Id and Password used to authenticate
- Specify any additional option as needed

Big Data/NoSQL Connections

BigData and NoSQL endpoints are treated by DataZen as specialized HTTP/S endpoints; as such, they also support Call Rate and KB Rate limiting as explained in the **HTTP/S REST Connections** section.

Google BigQuery

To connect to a Google BigQuery endpoint, choose the Google BigQuery Service Type. The Google endpoint for this service is usually: <https://bigquery.googleapis.com/bigquery/v2/>

- Enter your project id
- Enter the entire OAuth 2.0 Client ID JSON file in the respective fields. You can download your Key JSON file from the Google Cloud portal
- Specify the default DataSet and Table to use

CosmosDB Databases

To connect to a CosmosDB database endpoint, choose the CosmosDB connection Type and enter the Azure Connection String to the CosmosDB service. The Account Name will be automatically extracted and displayed.

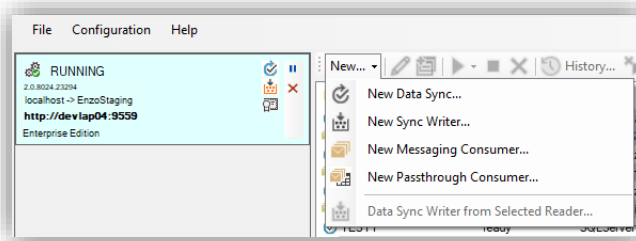
Choose a default database and table from the dropdown lists.

Job Readers

Data Sync Jobs consist of a Job Reader, and optionally a Job Writer. This section explains the various Job Reader options.

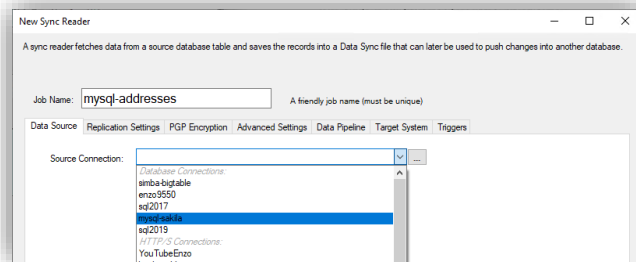
Create a Sync Job

To create a Data Sync job, select the desired Sync Agent, and follow these steps:



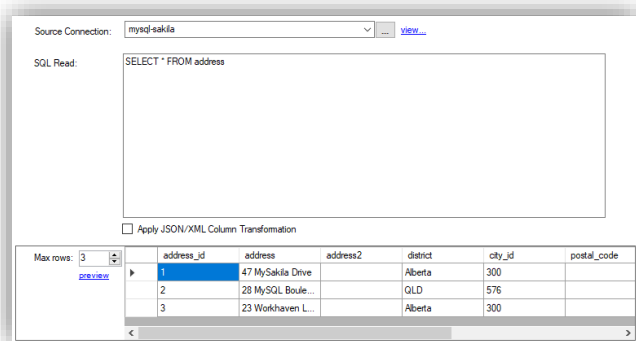
Select **New -> Data Sync...**

A Data Sync job always creates a reader, and optionally a writer. When a writer is also created, the Data Sync job is called a “**Direct**” job.



Specify a Job Name; the name must be unique for this agent; it is recommended to use a descriptive name. Special characters are not allowed.

Select an existing source connection, or click on the ellipses ... to manage connection strings. See the Connection Strings section for more information.

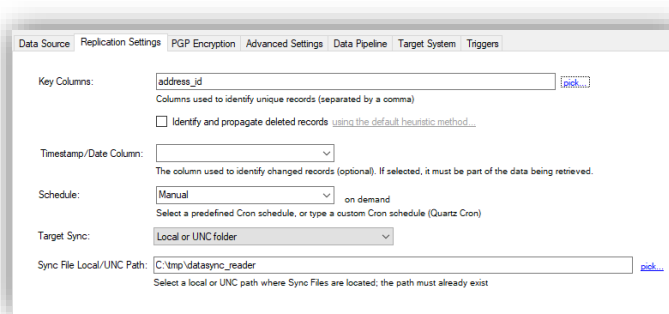


When using a database connection, enter a SQL command and click the [Preview...](#) link.

You can enter any valid SQL command including EXEC calls for SQL Server. However, simpler SQL commands may enable Data Zen to apply windowing heuristics to optimize data fetch operations.

NOTE: If the SQL command is too complex, the preview may return all available records.

The above example shows you how to create a Reader that connects to database or an ODBC driver. However, the Job Reader can read from flat files, HTTP REST APIs, and Messaging platforms. See the File-Based Replication, Messaging Replication, and HTTP REST API sections for more information.

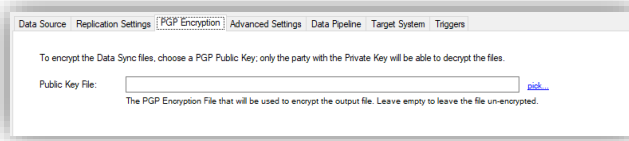


Click on the Replication Settings tab. Choose one or more Key Columns. Key Columns are used to identify changes.

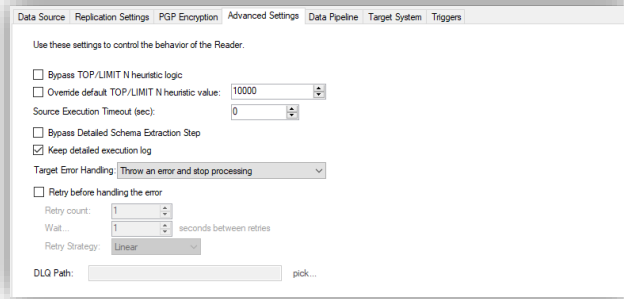
Select or type the desired Cron schedule.

NOTE: If no key columns are specified, Data Zen will only create Full Data Sync files and all source records will be forwarded every time.

Note: The Target Sync option allows you to choose a folder where the Sync Files will be created; you can also choose an FTP site or a cloud drive.

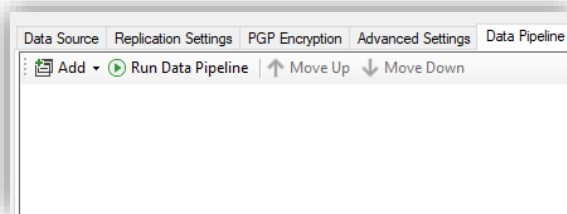


To encrypt the Data Sync File, select an existing PGP Public Key.

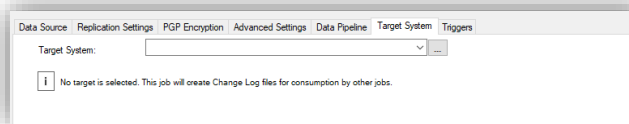


The Advanced Settings tab allows you to control the way DataZen selects records from the source system. These options can be useful to control memory usage for tables containing millions of records.

You can also choose how errors are handled using a retry strategy. The Dead Letter Queue (DLQ) path can be specified here. Check the “Keep detailed execution log” option if desired.

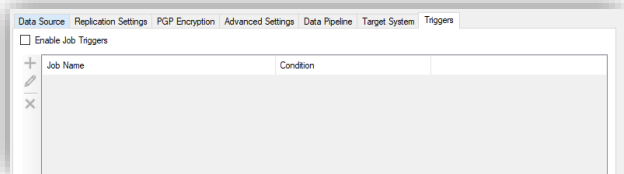


The Data Pipeline tab allows you to add data masking, data filtering, data quality validation, data hashing, and adding dynamic columns. See the Data Pipeline section for more information.

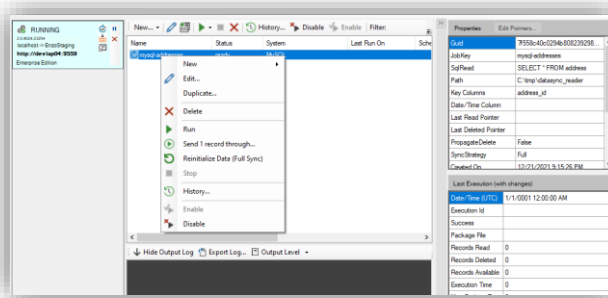


The Target System tab allows you to select the system where the records will be sent to. Leaving this option blank allows you to create Change Logs that can be replayed at any time in the future.

If you choose a target system, you will not be able to replay the change logs.



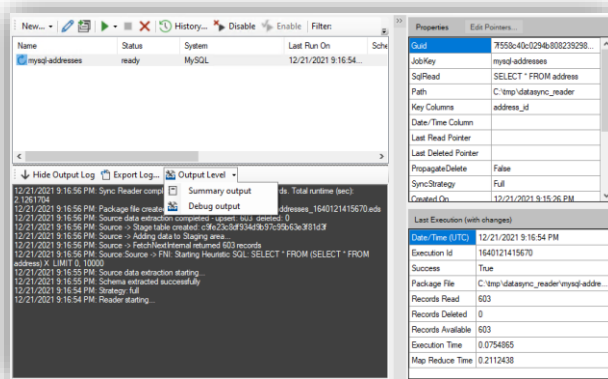
Job Triggers allow you to start other jobs when this one completes. Various options are available. See the Triggers section for more information.



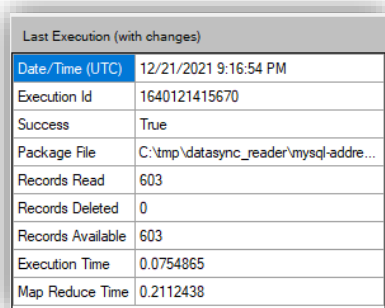
Click OK to save the Data Sync job.

Select the newly created reader, right-click on it, and select **Run**.

NOTE: If you have specified a Cron Schedule, the job starts automatically.



You can view the output of the job below in the Log section. Make sure to select **Debug Output** to see all relevant details.



Once the execution of the operation has completed, the **Last Execution (with changes)** window will show fetch statistics including the time it took to perform the read options and the change data capture. The output file (Package File) is also found here.

We can see that 603 records are available.

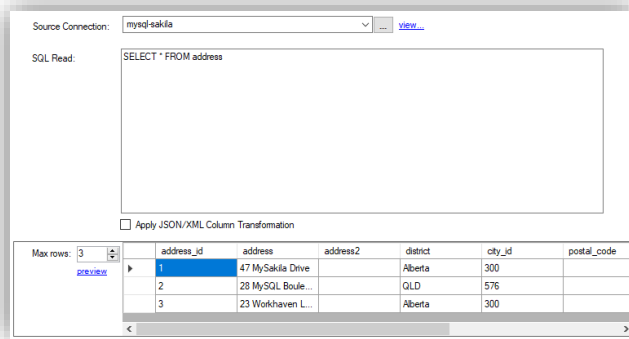
The Execution Id is a sequence number for a Sync file; because the Execution Id represents a date/time, this number increases every time a new Sync File is created.

Database Job Reader

When the source system is a database, such as a SQL Server database or an ODBC driver, the Job Reader consists of a SQL command. The SQL command itself needs to conform to the source database platform. However, depending on the settings of the Job Reader, the SQL command may be modified to reach higher performance or to select a subset of the records used for replication. If using Change Data Capture, the SQL command is generated automatically; see the **Change Data Capture** section for further information.

Generally speaking, the simpler the SQL Command, the easier it will be for DataZen to modify it for performance needs. However, these optimizations are optional. You can, in theory, create any script supported by the database engine here, including Group BY operations, JOINS and calling Stored Procedures and Functions.

*If the SQL Command is too complex, the **Max Rows** options may fail silently and return all available rows instead.*



When using a database connection, enter a SQL command and click the [Preview...](#) link.

You can enter any valid SQL command including EXEC calls for SQL Server. However, simpler SQL commands may enable Data Zen to apply windowing heuristics to optimize data fetch operations.

NOTE: If the SQL command is too complex, the preview may return all available records.

Change Data Capture

Using Change Data Capture instructs DataZen to read from known internal tables of the database engine in order to capture changes, and bypass DataZen's internal synthetic CDC Engine.

NOTE: When a CDC method is used, the Key Columns and Deleted Record identification options are automatically set and are disabled.

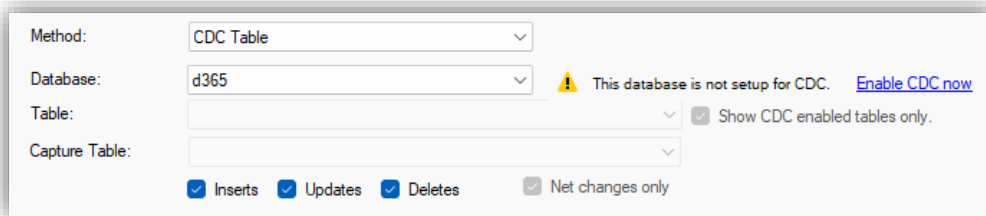
This feature is currently only available for SQL Server databases.

If you need to transfer all initial records from a source table, consider enabling CDC first, then perform a backup/restore of the table(s) manually. The CDC features only forward database changes; not the initial records.

Two modes are available for SQL Server: **Change Data Capture** and **Change Tracking**.

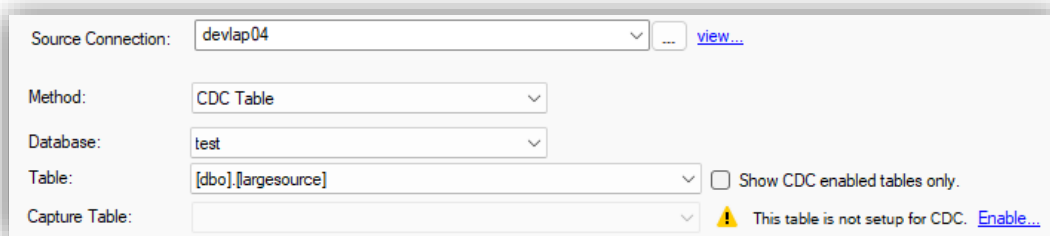
Change Data Capture

If your SQL Server database version supports the Change Data Capture option provided by the database engine, DataZen can read the underlying CDC tables to extract the changes directly. The database selected must be setup for CDC; DataZen can assist with this operation; click on **Enable CDC now** to allow CDC on the selected database.



SQL Server CDC is not available on Express Editions of SQL Server.

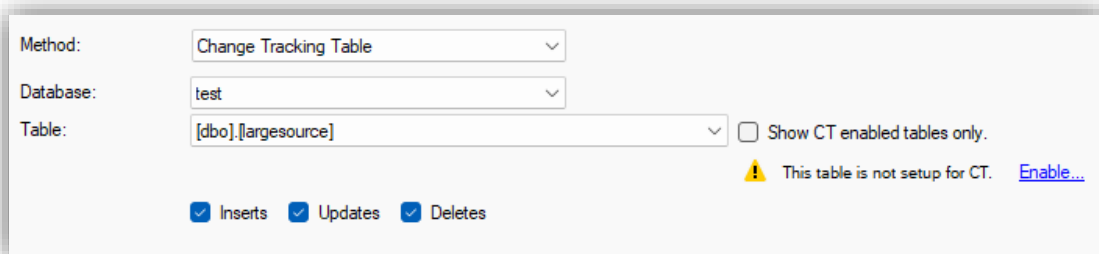
Once CDC is enabled on the database, select the table to capture. Uncheck the **Show CDC enabled tables only** to view all tables in the database. If the table is not enabled for CDC, a warning will be displayed; you can enable the table for CDC by checking the **Enable...** link.



Once enabled, select the Capture Table to track and select the types of changes to track: Inserts, Updates and Deletes. You can either capture all changes or only the net changes since the last call by checking the **Net changes only** option.

Change Tracking

Change Tracking is another change capture feature of SQL Server databases; these internal tables capture “net” changes only (as opposed to Change Data Capture). Choose the **Change Tracking Table** method, and select the table to capture. If the table is not configured for change tracking, a warning will be displayed; you can enable Change Tracking by clicking on the **Enable** link.



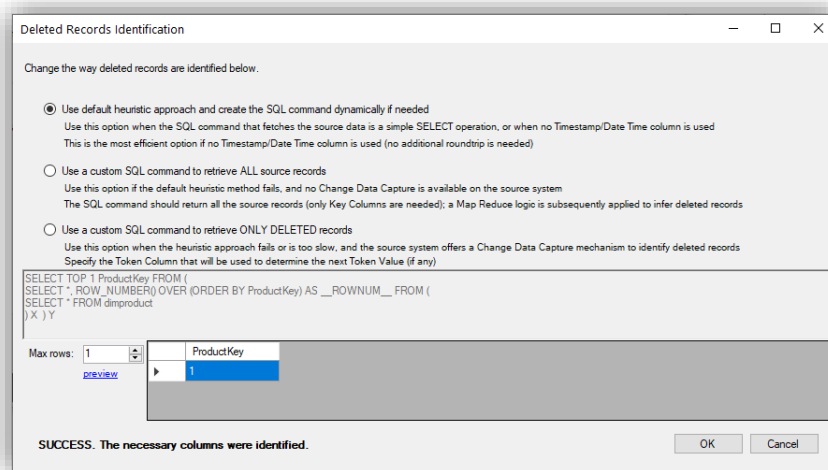
Choose which change types you would like to track: **Inserts, Updates, and Deletes**.

Overriding the Deleted Records Identification Method

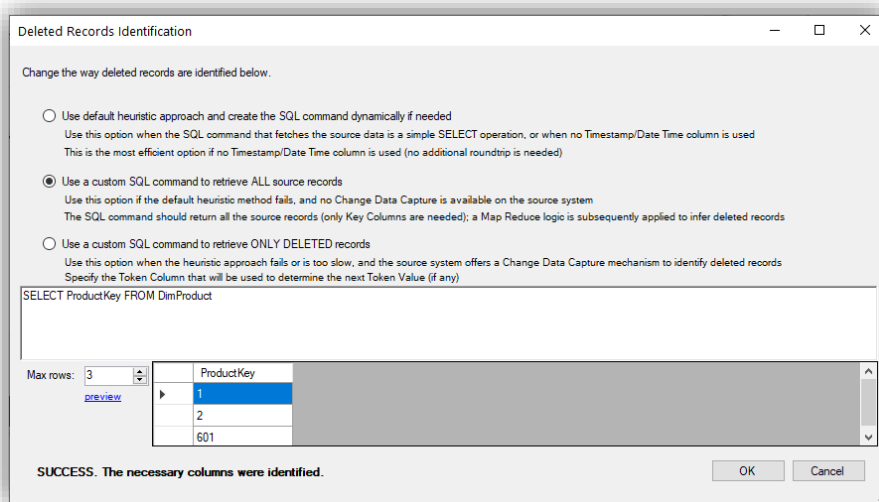
When the source system is a relational database, there are three ways DataZen can identify deleted records, if needed:

- Default Heuristic
- Custom Full Read
- Custom Change Data Capture

The **Default Heuristic** approach works best when no Timestamp/DateTime column is specified, or when the Source SQL provided is a simple SELECT operation and the source system has relatively few records. If no Timestamp/DateTime column is specified, DataZen infers deleted records by assuming the SQL command returns all available records, and assumes that records missing from the previous execution have been deleted. In other words, when no Timestamp/DateTime column is specified, DataZen does not need to make another call to the source system. You can inspect and execute the modified SQL command by clicking on the link next to the **Identify and Propagate Deleted Records** checkbox to preview the output of the SQL command.



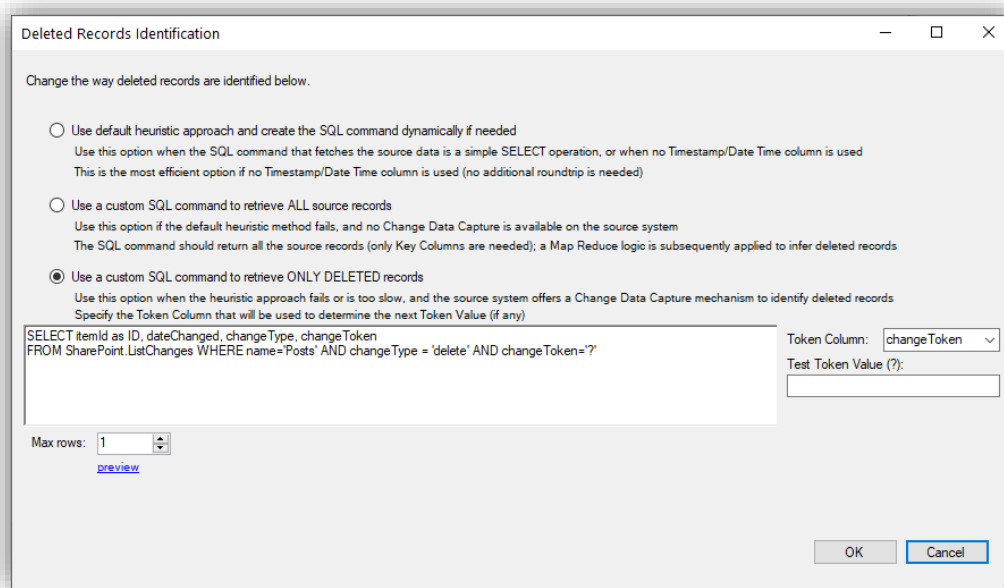
The **Custom Full Read** approach allows you to enter a custom SQL command that returns all available records from the source system; the command specified can be any valid SQL command, but must return the Key Columns. Similar to the Default method, this approach assumes that the SQL command returns all available records; DataZen assumes that records missing from the previous execution have been deleted. Use this option if the SQL command generated by the Default Heuristic method is too complex and the source system doesn't offer a Change Data Capture mechanism.



The **Custom Change Data Capture** option can be used if the source system offers a tracking mechanism for identifying which records were deleted (such as a Change Data Capture table). You can configure the Job Reader to execute a custom SQL command that returns the deleted records only; for best results, the SQL command specified should a column that identifies the change token that marks the last known change from the source system (identified in DataZen as the **Last Deleted Pointer**). This allows DataZen to keep track of the last known deleted operation, and only forward the relevant deleted records. In the example below, the SQL command returns two important columns: the Key Column (ID field) that identify uniquely the deleted records, and the ChangeToken field (specified as the Token Column) that tracks the last known change token from the source Change Data Capture.

Use a ?, '?' or '?:null' placeholder to inject the last token value retrieved; the placeholder has the following behavior the first time the command is executed (when no change token value exists):

- ? is replaced with a 0 (do not include quotes when the last token is a decimal)
- '?' is replaced with an empty string ("")
- '?:null' is replaced with a NULL value



Deleted Records Identification

Change the way deleted records are identified below.

☐ Use default heuristic approach and create the SQL command dynamically if needed
Use this option when the SQL command that fetches the source data is a simple SELECT operation, or when no Timestamp/Date Time column is used
This is the most efficient option if no Timestamp/Date Time column is used (no additional roundtrip is needed)

☐ Use a custom SQL command to retrieve ALL source records
Use this option if the default heuristic method fails, and no Change Data Capture is available on the source system
The SQL command should return all the source records (only Key Columns are needed); a Map Reduce logic is subsequently applied to infer deleted records

☒ Use a custom SQL command to retrieve ONLY DELETED records
Use this option when the heuristic approach fails or is too slow, and the source system offers a Change Data Capture mechanism to identify deleted records
Specify the Token Column that will be used to determine the next Token Value (if any)

SELECT itemid as ID, dateChanged, changeType, changeToken
FROM SharePoint.ListChanges WHERE name='Posts' AND changeType = 'delete' AND changeToken=?

Token Column:

Test Token Value (?):

Max rows: [preview](#)

OK Cancel

Identifying deleted records is only available if at least one Key Column has been selected.

Using the Timestamp / DateTime Column

By default, when the Timestamp is not specified, DataZen is capable of identifying changes from source systems automatically, even if the source system doesn't keep track of insert, update and delete operations. However, in order to achieve this capability, DataZen must read all available records from the source system every time the Job Reader runs. While this may be acceptable for most source systems, it could present a performance issue for very large tables.

In order to improve the performance of a Job Reader against a data source you can use the **Timestamp/DateTime** column setting, when the source SQL command is a SELECT operation. This option is equivalent to using a High Watermark setting for REST APIs. DataZen attempts to modify the Source SQL command to include a filter on the column specified. This optimization can work if the column specified is a Timestamp, Date/Time or a Numeric data type; in some cases it may also work with String data types. By specifying a Timestamp/DateTime column, DataZen can reduce the number of records read from the source system significantly. You can either select a field name from the dropdown box, or enter the column directly in the field manually. DataZen keeps track of the maximum value identified in the **Last Read Pointer** property.

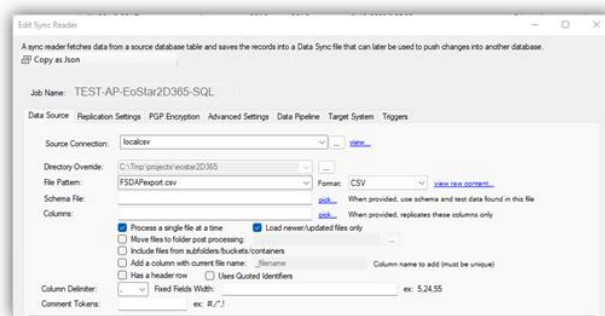
Specifying a Timestamp/DateTime column is only available for Database Sources and if the SQL Command is a SELECT operation.

Drive Job Reader (Parquet, XML, JSON, CSV)

A Sync Job can read data from files located on drives, or in the cloud, and process them as a regular data source. In order to be processed, the file needs to confirm to a supported format. If you need to read files using other formats not support by DataZen, you may still be able to do so by using an ODBC driver provided by a 3rd party vendor.

To read from files, you first need to create a **Drive Connection String**.

Parquet, XML, JSON and CSV files can be read from and written to a local drive, in Azure Storage Containers, in an AWS S3 bucket, or on an FTP server.



The **Directory Override** option allows you to choose a different root path from the default one provided by the selected source connection

The **File Pattern** field allows you read a subset of files based on the name of the file. You can use the * wildcard to process multiple files (ex: *-test.txt). When the file type is CSV, JSON or XML, you can preview the raw content of the file.

The **Columns** field allows you choose a subset of columns you would like to replicate. If empty, all the fields will be read.

The **Format** option allows you to limit the files that will be ingested. If you leave this option as **Auto-Detect**, the file selection process will determine the file type dynamically when it opens the file for reading.

The **Process Single File** option treats each file as a separate execution instance of the job and creates a Sync File per file

The **Load newer/updated files only** option allows to only read files that have changed since the last time the Job Reader executed.

The **Move files to folder** option allows to move files that were processed to a different folder (local files only).

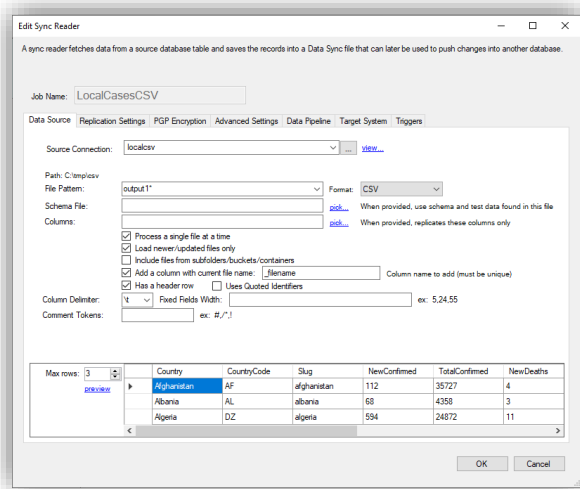
The **Include Files from Subfolders...** option allows you to read all files found under the directory and sub-directories.

The **Add a Column...** option allows you to add a column to the output with the name of the file where the record was found

Document root path: when dealing with an XML or a JSON document, uses the path provided to filter the document content or read from a specific path. See the [JSON/XML Column Transformation](#) section for more information on how to configure this field.

Reading CSV Files

Certain flat files, such as comma-delimited files (normally with a .CSV file extension) can be ingested by a Job Reader. To see all available options, select the CSV file format in the Format dropdown box.



Has Header Row: When checked, gives a hint to DataZen that the first row of data contains column names.

Uses Quoted Identifiers: check this box if the document contains data elements surrounded by double-quotes (such as content that spans multiple rows).

Column Delimiter: When using a field delimiter, indicates the character used to mark the end of a field. By default, this value is `\t` (tab).

Field Width: for fixed-length documents only; comma-separated length of each field.

Comment Tokens: Comma-separated list of strings used to identify comments in the document.

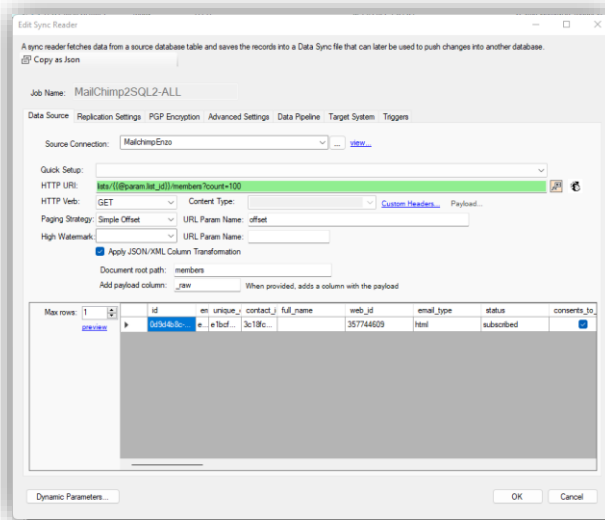
Note that DataZen will automatically identify the data type to use in a document by reading the first 99 records. If records past this limit contain an unexpected data type in a given column, the column will be converted back to a **String**.

HTTP/S REST API Job Reader

DataZen can be configured to extract data using REST APIs from a vast array of Internet data sources, including Social Media platforms, CRM/ERP SaaS platforms, and custom APIs. In order to consume REST APIs, DataZen supports the following capabilities:

- **API Service Authentication:**
 - API Key: use an API key as a parameter in the URL or as a custom header
 - Basic Auth: use basic authentication using a user id and encoded password
 - Token: use a shared token as an authentication header
 - Bearer Token: use a bearer token, normally required by OAuth 2.0 schemes
- **Refresh Tokens:** certain APIs leveraging OAuth 2.0 require the use of Refresh tokens
- **API Paging:** certain APIs implement a paging mechanism when calling certain endpoints; DataZen implements three paging strategies:
 - Simple Offset: uses a count offset mechanism through a URL parameter
 - Simple Paging: uses a paging offset mechanism through a URL parameter
 - Next URI Link: uses a link provided in the response document as the next URL
 - Bookmark/Token: uses a node in the response document to use as a URL parameter
- **High Watermark:** used when calling certain REST APIs from a last known value to avoid fetching all records too many times

While the API Service Authentication and Refresh Tokens are handled by the connection settings, the API Paging and High Watermark settings are specified here. The following example shows how to configure a job that extracts data from the Mailchimp service to obtain all the members of a specific list; this requires that the List ID is known (the list ID is provided as part of the URL).



You can pre-fill specific settings by choosing an existing Quick Setup option. These options are filtered based on the service connection selected, and as a result it may be empty.

- URL: lists/123456/members?count=100
- HTTP Verb: GET
- Paging Strategy: Simple Offset, using the **offset** Query Parameter
- High Watermark: left empty (extracting all members every time)
- Apply JSON/XML Tx: Yes, specifying **members** as the root path
- Add Payload Column: **_raw** (this setting is optional; it adds a column dynamically with the entire record payload as JSON)

JSON/XML Column Transformation

The JSON/XML transformation option allows you to convert a document into a row and columns format automatically. This option is optional calling a REST endpoint, and mandatory when loading JSON or XML files. By default, if left empty, the payload received will be turned into rows and columns from the top of the document.

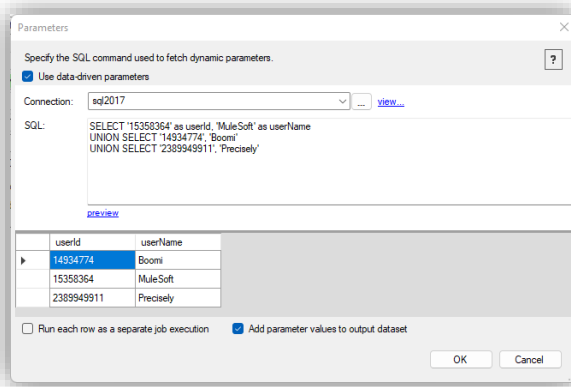
The path notation differs between JSON and XML documents, but essentially the complete document will be read. However, when specifying a path (as an XPath for XML documents or JSON Selector for JSON documents), only the content underneath this path will be converted into rows and columns.

ROOT PATH IS CASE-SENSITIVE

Dynamic Parameters

Dynamic Parameters are a feature of HTTP/S REST job readers; they allow parameter replacement within the URI being used to call the endpoint multiple times and provide a single, combined result set. When used as part of the request URI, the HTTP URI field displays a green background.

To define dynamic parameters, click on the Dynamic Parameters button at the bottom.



Check the **Use data-driven parameters** option, select a database source from the list, and enter a SQL command. The command can be any valid SQL request, including EXEC commands.

Click Preview to see the output.

To append the input parameter to each row from the source system, check the **Add parameter values to output dataset** option.

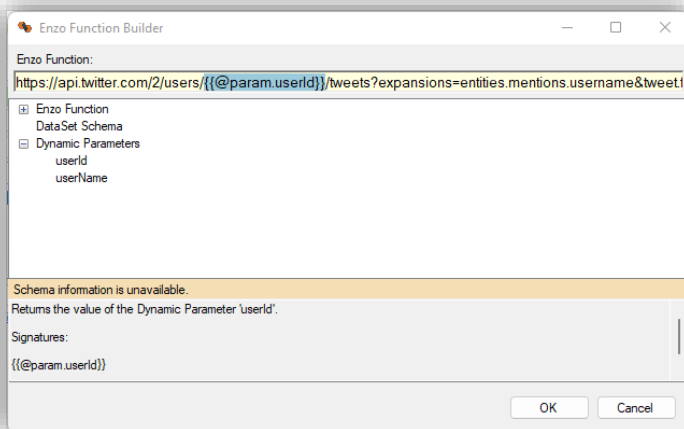
To request each input parameter to be executed separately as individual job execution (instead of appending all results into a single output), check the **Run each row as a separate job** option.

NOTE: THE FIRST COLUMN FROM THE DYNAMIC PARAMETERS LIST WILL BE USED AS THE KEY FOR TRACKING THE LAST "HIGH WATERMARK" VALUE. AS A RESULT, MAKE SURE THE FIRST COLUMN IS A UNIQUE VALUE IN THE ABOVE QUERY.

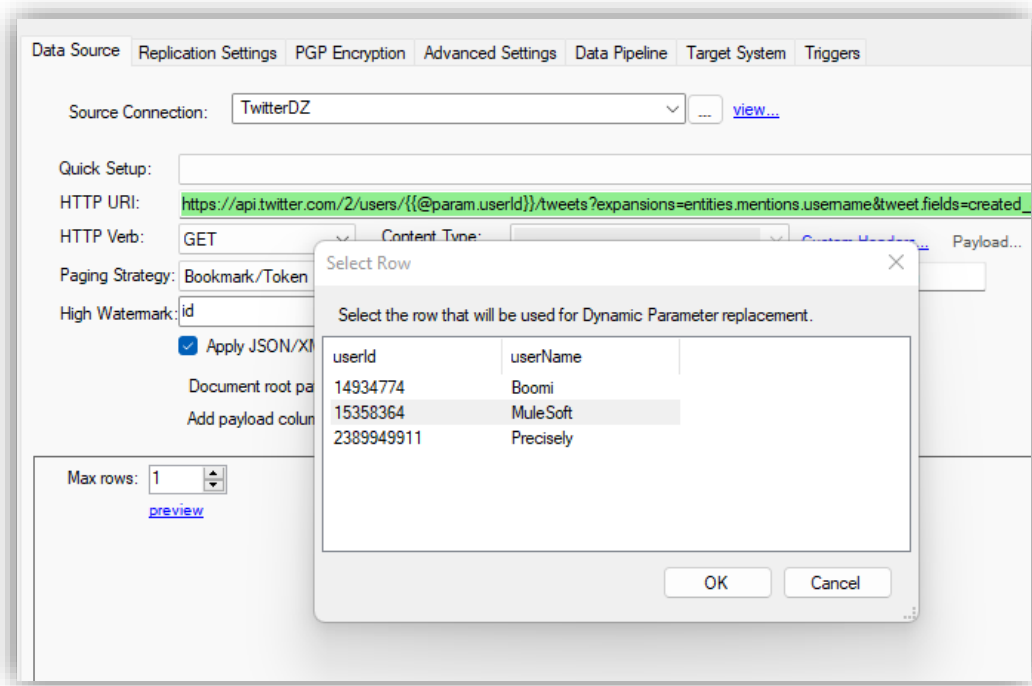
Once defined, you can use the column names of the dynamic parameters as input variables into the URI. For example, if the above represents twitter IDs, and the column name is "userId", use this notation: **{{@param.userId}}**.

HTTP URI: https://api.twitter.com/2/users/{{@param.userId}}/tweets?expansions=entities.mentions.username&tweet.fields=created_at,author_id&user.i

You can also build this URI by clicking on the DataZen Function icon. The dynamic parameter column names are available under the **Dynamic Parameters** node. Double-click on the desired parameter to insert it.



Finally, when Dynamic Parameters are defined, clicking on the Preview link will prompt you to select a parameter value.



XML Example

Let's consider the following XML document:

```
<?xml version="1.0" encoding="utf-8" ?>
<bookstore>
  <book genre = "autobiography" publicationdate="1981-03-22" ISBN="1-861003-11-0">
    <title>The Autobiography of Benjamin Franklin</title>
    <author>
      <first-name>Benjamin</first-name>
      <last-name>Franklin</last-name>
    </author>
    <price>8.99</price>
  </book>
  <book genre = "novel" publicationdate="1967-11-17" ISBN="0-201-63361-2">
    <title>The Confidence Man</title>
    <author>
      <first-name>Herman</first-name>
      <last-name>Melville</last-name>
    </author>
    <price>11.99</price>
  </book>
  <book genre = "philosophy" publicationdate="1991-02-15" ISBN="1-861001-57-6">
    <title>The Gorgias</title>
    <author>
      <name>Plato</name>
    </author>
    <price>9.99</price>
  </book>
</bookstore>
```


Root Path

Output

<blank>

bookstore

When the root path is empty, or set the **bookstore**, DataZen will inspect the first child element (book) and consider all **book** nodes for its rows. Each book attribute will become a column, and the book node itself will contain the inner XML of the book node.

	book	genre	publicationdate	ISBN
▶	<title>The Gorgias</title><author><name>Plato</nam...	philosophy	1991-02-15	1-861001-57-6

bookstore/book

When the root path is set to **bookstore/book**, DataZen will treat the attributes and children of the book node as column names. Since the author node is a parent node of additional information, it contains the inner XML of the node.

	genre	publicationdate	ISBN	title	author	price
▶	autobiography	1981-03-22	1-861003-11-0	The Autobiograp...	<first-name>Benj...	8.99

JSON Document Example

Let's consider this JSON document; the document contains three properties, one of which is an array of values.

```
{
  "range": "Sheet1!A2:B9",
  "majorDimension": "ROWS",
  "values": [
    [
      "Platform Acquisition Cost",
      "$100,000"
    ],
    [
      "SDK Cost",
      "$25,000"
    ]
  ]
}
```

Root Path	Output								
<blank> \$	<p>When the root path is empty, or set \$, DataZen will inspect the top node.</p> <table><tr><th></th><th>range</th><th>majorDimension</th><th>values</th></tr><tr><td>▶</td><td>Sheet1!A2:B9</td><td>ROWS</td><td>[["Platform Acquisition Cost", "\$100,000"], ["SDK Cost", "\$25,000"]]</td></tr></table>		range	majorDimension	values	▶	Sheet1!A2:B9	ROWS	[["Platform Acquisition Cost", "\$100,000"], ["SDK Cost", "\$25,000"]]
	range	majorDimension	values						
▶	Sheet1!A2:B9	ROWS	[["Platform Acquisition Cost", "\$100,000"], ["SDK Cost", "\$25,000"]]						
values \$.values	<p>When the root path is set to values, or \$.values, DataZen will only look within the values property. Since its content is an array of values, column names are created automatically as col0, col1...</p>								

	col0	col1
▶	Platform Acquisiti...	\$100,000
	SDK Cost	\$25,000

JSON Array Example

Consider the following JSON Array consisting of objects, taken from a Solr search response. The response header contains useful information; however, the **response.docs** node contains an array of objects with the search result.

```
{
  "responseHeader":{
    "status":0,
    "QTime":10,
    "params":{
      "q":"*:*",
      "_":"1615932159060"}},
  "response":
    {"numFound":3,"start":0,"numFoundExact":true,"docs":[
      {
        "id":"1",
        "title":["ACADEMY DINOSAUR"],
        "description":"A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies",
        "release_year":2006,
        "rating":"PG",
        "last_update":"2006-02-15T05:03:42.0000000",
        "_version_":1694589148527067136},
      {
        "id":"3",
        "title":["ADAPTATION HOLES"],
        "description":"A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory",
        "release_year":2006,
        "rating":"NC-17",
        "last_update":"2006-02-15T05:03:42.0000000",
        "_version_":1694589148544892929}
    ]
  }
}
```

Root Path	Output																		
<blank> \$	<p>When the root path is empty, or set \$, DataZen will inspect the top node. The response node contains further information about the search operation.</p> <table><tr><th></th><th>responseHeader.st</th><th>responseHead</th><th>responseHi</th><th>responseHe</th><th>response.nu</th><th>response.start</th><th>response.numFour</th><th>response</th></tr><tr><td>▶</td><td>0</td><td>10</td><td>+:*</td><td>16159321...</td><td>3</td><td>0</td><td><input checked="" type="checkbox"/></td><td>{ "numFound": 3...</td></tr></table>		responseHeader.st	responseHead	responseHi	responseHe	response.nu	response.start	response.numFour	response	▶	0	10	+:*	16159321...	3	0	<input checked="" type="checkbox"/>	{ "numFound": 3...
	responseHeader.st	responseHead	responseHi	responseHe	response.nu	response.start	response.numFour	response											
▶	0	10	+:*	16159321...	3	0	<input checked="" type="checkbox"/>	{ "numFound": 3...											
response \$.response	<p>When the root path is set to response, or \$.response, DataZen will inspect the response node. The docs property however contains the actual search result.</p>																		

	numFound	start	numFoundExact	docs
▶	3	0	<input checked="" type="checkbox"/>	[{ "id": "1", "title": ["ACADEM...

response.docs \$.response.docs

When the root path is set to **response.docs**, or **\$.response.docs**, DataZen will only look within the **docs** property. Since its content is an array of objects, column names are extracted from the JSON document automatically.

	id	title	description	release_year	rating	last_update	_version_
▶	1	["ACADEMY DI...	A Epic Drama of ...	2006	PG	2/15/2006 5:03 ...	1694589148527...
	3	["ADAPTATION...	A Astounding Ref...	2006	NC-17	2/15/2006 5:03 ...	1694589148544...

Messaging Consumers

DataZen can be used as a Messaging Consumer against a variety of messaging platforms, including MSMQ, AWS SQS, Kafka, Azure Event Hub, Azure Messaging Hub, Google PubSub and RabbitMQ. This section explains how to configure the listeners and the options available to forward these messages. DataZen offers the following capabilities:

- **Automatic reconnection:** Consuming messages is an ongoing operation: DataZen will connect to the messaging platform and will attempt to reconnect after network failures automatically and as quickly as possible
- **Automatic message buffering:** when configured, messages can be held temporarily before being forwarded in bulk to the target system if possible
- **Automatic message properties forwarding:** when forwarding messages from one platform to another, as many properties of the original message can be forwarded to the target platform if possible
- **Passthrough or Payload inspection:** DataZen supports passthrough messaging or deep payload inspection for forwarding the message content into a target system that expects rows and columns

When using a Messaging Platform as the source, select one of the processing options as needed; see below for further information.

Specify the Queue/Topic name and which message metadata properties to include, and any message headers if available.

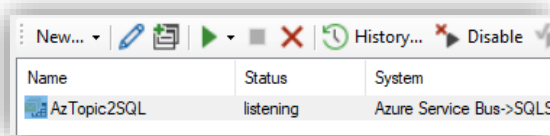
You can also specify a sample message payload to test the functionality of the reader without actually consuming a message from the messaging platform.

The following processing options are available:

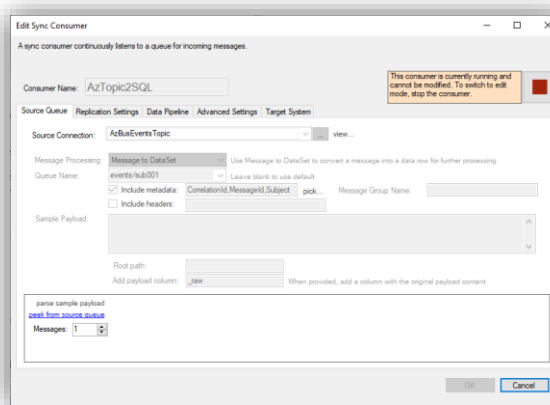
Processing Option	JSON/XML Transformation	Targets	Replay	Headers & Metadata Available
Passthrough	not available	Messaging Targets	No	Yes

Message to Dataset	not available	No Target Drives Targets Database Targets HTTP/REST Targets BigData Targets	Yes, with No Target	Yes
Payload to Dataset	available	No Target Drives Targets Database Targets HTTP/REST Targets BigData Targets	Yes, with No Target	No

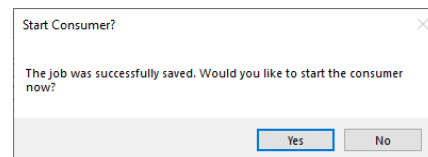
Note that unlike other Job Readers, you must disable the Job before changing its settings.



When the listener is running, the status for the job will be displayed as **listening** as shown here. You can either disable the job manually, or do it in the Edit screen. To edit this job, double-click on it.



The edit screen will be disabled if the job is listening. Click on the **Stop** button to disable the job and start changing the necessary options. An option to restart the job will be shown upon closing this screen.



Pointer Values: Editing the High Watermark

DataZen tracks last known values (known as **Last Pointers**) for its High Watermark tracking: when a Custom Change Data Capture mechanism is selected for identifying deleted records, when a Timestamp/DateTime column is identified for the Job Reader for databases, when the option to only use the latest files is selected for Files replication, or when a High Watermark field is selected for REST APIs.

In the case of Change Tracking Data Capture, the Job Reader tracks the last known value of the Change Token of the CDC table from the source system. This token is usually a string that represents a point in time that the CDC engine uses to identify subsequent changes. DataZen keeps a copy of the last known token so that only deleted operations from the last execution are returned by the CDC engine.

When a Timestamp/DateTime column is selected, DataZen modifies the SQL Read command to identify new and updated records since the last time the job ran. When a Job Reader runs and returns records, DataZen finds the maximum value of the Timestamp/DateTime column from the data returned and will use it as the new pointer the next time the Job Reader runs.

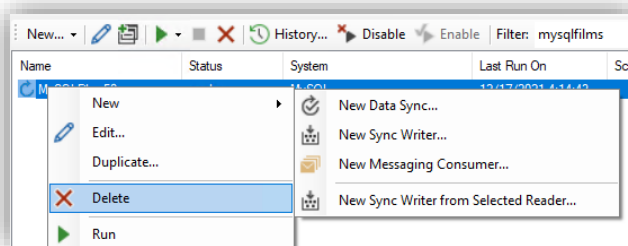
You can edit the Last Pointers if needed (click on the **Edit Pointers** button on the Properties window). This screen allows you to modify both values (when relevant), and either specify a date/time, a numeric value, or a custom field. You can also reset the Pointers to a NULL value.

Job Writers

Generally speaking, a Job Writer pushes data to a target system. A Job Writer can be created directly within a Data Sync job, or separately. When created as part of a Data Sync Job, the Sync Files are automatically consumed and not available for replay. However, when the Job Writer is created independently from the Sync Job, the Sync Files remain on disk and are available for other Job Writers and for future replay as needed.

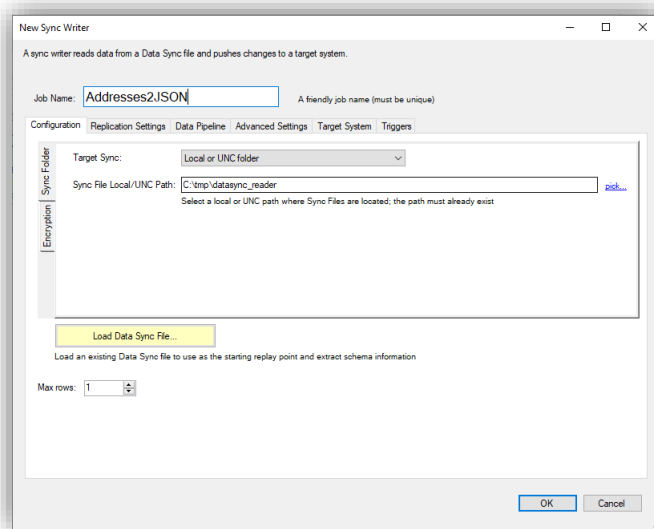
Create a Job Writer from An Existing Job Reader

The simplest way to create a Job Writer is to build it from an existing Job Reader. Doing so preloads information into the Job Writer configuration, such as the name of the corresponding reader, the location of the Sync Files, the schedule information and more.



Select the Job Reader previously created, right-click on it, and choose **New -> New Sync Writer from Selected Reader**.

NOTE: this option is not available for Job Readers that are Direct Jobs (Writer is already defined in the Sync Job itself) because the Sync Files are not available for replay.

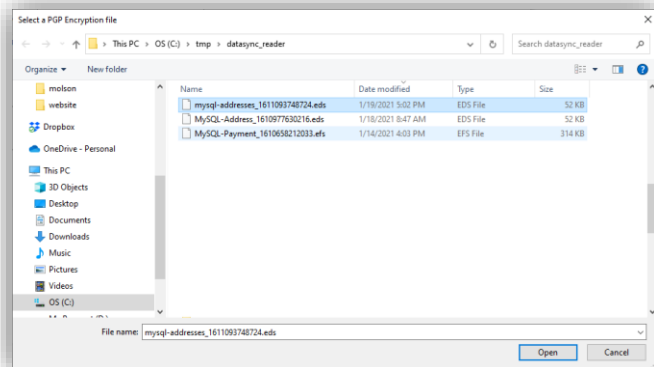


Enter a name for the job writer.

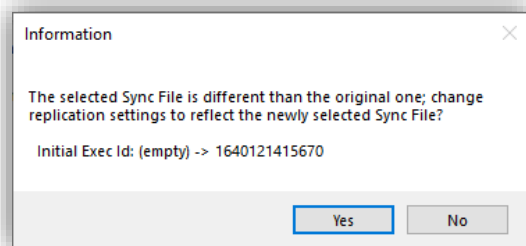
The Sync Path should point to the location of existing Sync Files; this should be automatically set.

If the data was encrypted with PGP, select the PGP Encryption tab, specify the PGP Private Key file and enter the password.

Click the Load Data Sync File button to open the Sync File, and click the [preview...](#) button. This data is being read from the Sync File. This step is needed to configure additional options.

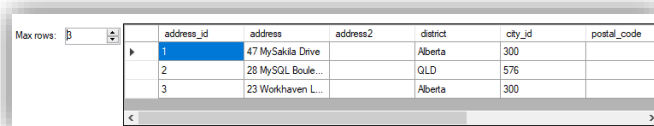


Select the file you would like to start reading from and click Open.



A confirmation window shows the changes that are about to take place for this Job Writer. In this case only the Initial Exec Id is about to be updated.

Click Yes.



Once loaded, you can view the content of the Sync File on the same screen.

New Sync Writer
A sync writer reads data from a Data Sync file and pushes changes to a target system.

Job Name: A friendly job name (must be unique)

Configuration | Replication Settings | Data Pipeline | Advanced Settings | Target System | Triggers

Source Job Name: Select or enter name of the source job that holds the changed records

Key Columns: [pick...](#)
Columns used to identify which records have been modified (separated by a comma)
Leave blank to implement an Append-Only Replication store

☐ Propagate deleted records if available

Initial Execution Id:
The initial Execution ID to play back from (leave blank to play back all sync files)

Schedule: on demand
Select a predefined Cron schedule, or type a custom Cron schedule (Quartz Cron)

Click on Replication Settings. All the information should be pre-filled based on the Sync File selected.

The Initial Execution Id represents the first sequence number that contains the full initial data set from the source system to be loaded into the target system.

Because the Key Columns fields is selected, records will be updated if the key values match, and inserted when they are not found in the target system. If this field is blank, all records from the Sync File would be considered new records (such as an audit log).

Select a Schedule if desired to let the Job Writer check for new files. Alternatively, you can update the Job Reader to trigger this Job Writer upon completion.

New Sync Writer
A sync writer reads data from a Data Sync file and pushes changes to a target system.

Job Name: A friendly job name (must be unique)

Configuration | Replication Settings | Data Pipeline | Advanced Settings | Target System | Triggers

Target System: [view...](#)

File Format: ☐ Date Field Identifier: ?
If not specified or invalid, uses the runtime date for date token replacement

Bucketization File Name Pattern
Container/Bucket/Directory: File Name:

☐ Use shadow copy unless file is more than hours old

Execution Timeout (sec):

☐ Use single record per file mode

Click on the Target System tab select a target connection. In this case, we are choosing a Local Path connection where we will create JSON documents.

Configure the target options as desired (see the Target Configuration Options section for more information).

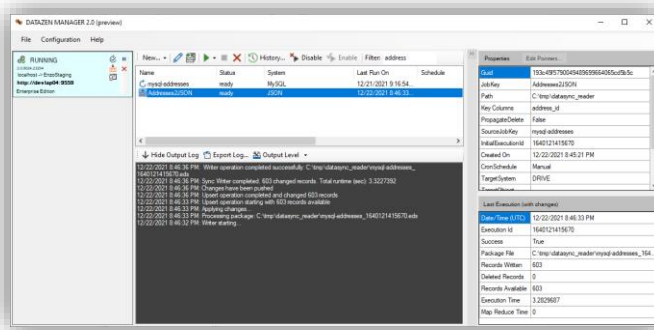
You can optionally configure Triggers that will start other jobs upon completion of this one in the Triggers section.

Click OK.

Name	Status	System
mysql-addresses	ready	MySQL
Addresses2JSON	ready	JSON

If no Cron schedule was selected, right-click on the Job Writer and select **Run**.

NOTE: If you have specified a Cron Schedule, the job starts automatically.



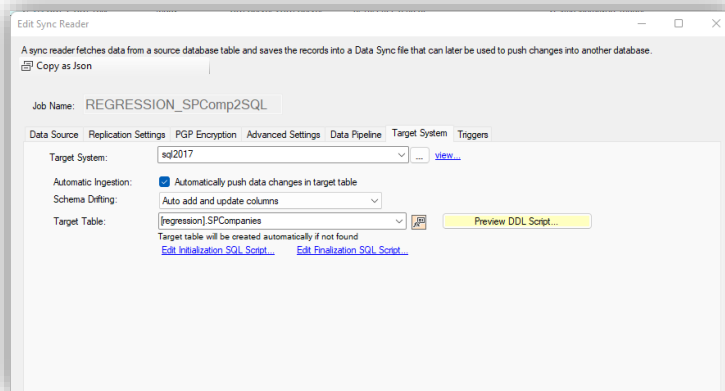
The **Last Execution (with changes)** window shows you that 603 records were available in the Sync File, and 603 were written to the target table.

It took about 3 seconds to push the records into JSON documents.

Database Targets

SQL Server (Native Provider)

When the **SQL Server (Native Provider)** system is selected in the Connection String form, DataZen uses a highly optimized Upsert operation against SQL Server (using a MERGE command).



Automatic Ingestion: when checked, schema drifting options are possible, allowing DataZen to automatically add columns to the target database and/or change data types as needed, depending on the option selected.

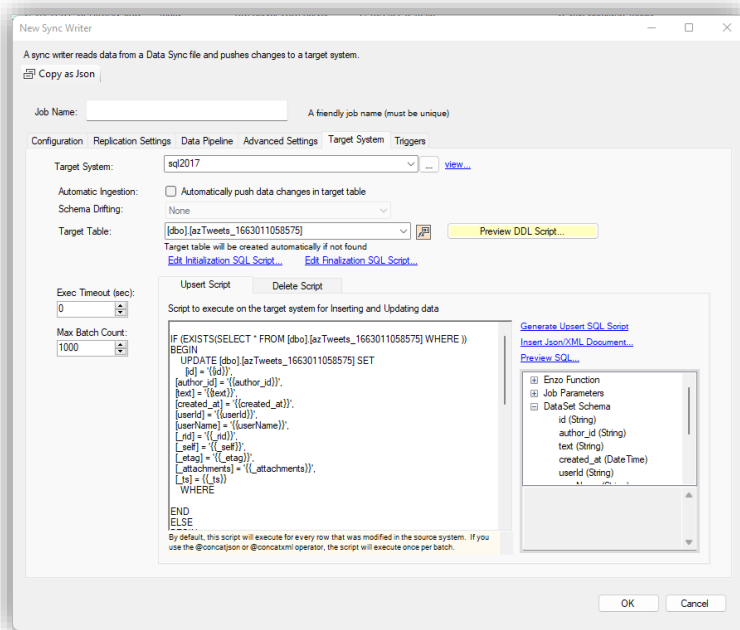
Target Table: Specify the name of the target table to use; DataZen functions can be used to change the name of the target database, schema, and table.

NOTE: If Automatic Ingestion is unchecked, you can provide a manual script to be executed; see the following section for details: **ODBC Drivers & Enzo Server**.

ODBC Drivers & Enzo Server

When a specific **ODBC** driver (SQL Server, Oracle, MySQL or Teradata) has been selected for a connection string, the target offers a less efficient, but more flexible replication options. The Job Writer Target System screen allows you to specify the replication scripts you want to use for upsert and delete operations.

*Note: You can also use replication scripts for Native SQL Server connections by unchecking the **Automatic Ingestion** option.*



Execution Timeout: The execution timeout applies to both the Upsert and Delete operations, and control how long the script is allowed to run before timing out. 0 means the script will not timeout.

Batch Count: The number of records sent to the target system (default: 1000). The larger the number, the more memory is required by DataZen to build the batch command; the maximum value is 100,000. For unique target platforms, such as Enzo Server, you may need to set this value to 1.

Messaging / HTTP/S Options: Additional options may be made available for Messaging endpoints. For example, when selecting an Azure Event Hub as the target system, the Event Hub Name can be specified.

Drive Connection: Parquet-specific options are available when choosing a drive. See the Parquet section for more information.

By default, the Upsert SQL script or Messaging body creates content for each data row available in the Sync File. However, in some cases, you may need to execute a single command with an XML or Json document per batch. See **Sending an XML/Json Document** below for more information.

Sending an XML/JSON Document

When a Job Writer uses a Custom SQL command for its Upsert logic, the SQL Batch send to the target system can be built in two different ways:

- One SQL command per record
- One SQL command per batch

One SQL Command per Record

This is the default mode of operation for an Upsert and Delete script: a SQL command will be created for each row found in the Sync File. The **Batch Count** setting controls how many SQL commands are sent to

Click on the Generate Upsert Script to build a SQL command that is compatible with the target system.

To specify a Delete script, use the **Delete Script** tab and click on the **Generate Delete Script** link.

You can modify this script at any time. The command created should be able to perform an Insert or an Update operation if a record exists. However, this script could be anything, including a call to a Stored Procedure.

Field values are represented by double curly brackets. For example, {{ID}} field will be replaced with the value of the ID field.

NOTE: Field names are case sensitive.

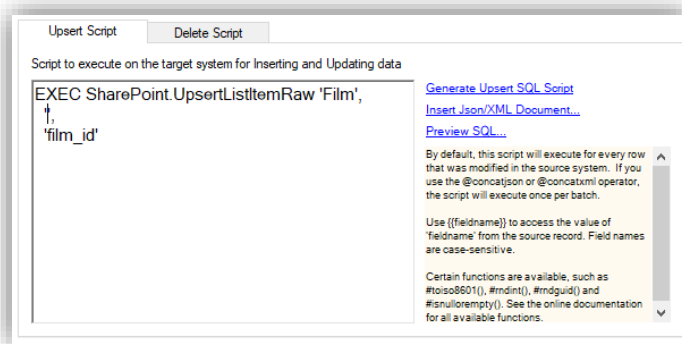
A number of pre-processing functions are available. Use the function area to insert specific fields or commands in the script directly by double-clicking on the desired item. See the **DataZen Functions** section for more information.

Click **Preview SQL** to inspect the command that will be sent to the target database.

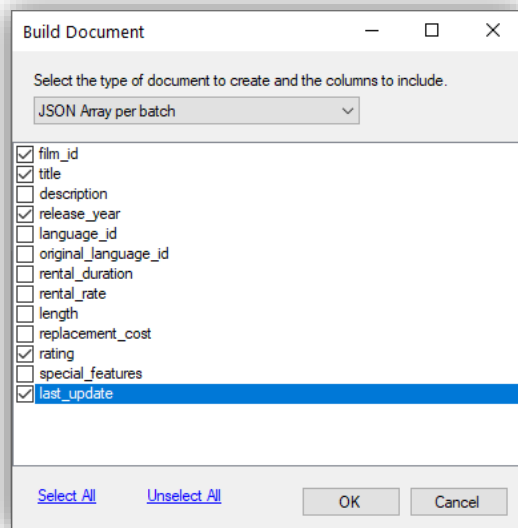
the target at one time. For example, if there are 100 records in the Sync File, and the Batch Count is set to 50, two batches of 50 SQL commands each will be sent to the target system.

One SQL Command per Batch

The Job Writer automatically switches to this mode if either the **@concatJson** or **@concatXml** operation is found in the Upsert or Delete script. When this mode is used, a Json or XML document is created and inserted into the SQL command where desired, and the **Batch Count** setting controls how many objects/nodes are created in the Json or XML document respectively. For example, if there are 100 records in the Sync File, and the Batch Count is set to 50, two SQL commands are sent to the target system, with each SQL command containing a XML/Json document of 50 nodes/objects each.

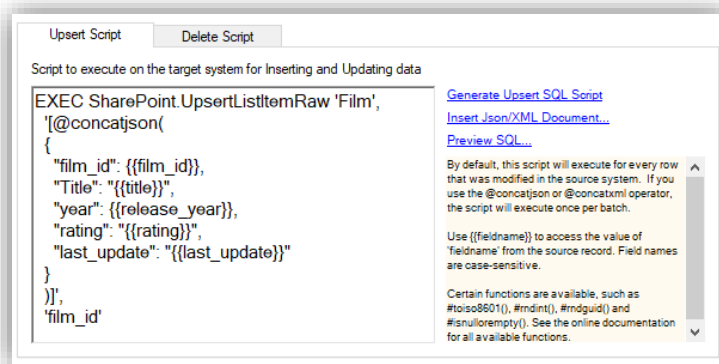


From the Upsert Script window, place the cursor where you would like to insert the Json/XML document, and click on **Insert Json/XML document...**



A dialog allows you to select the fields to include in the document, and the type of document to create. Choose either **JSON Array per batch**, or **XML Document per batch**; click OK.

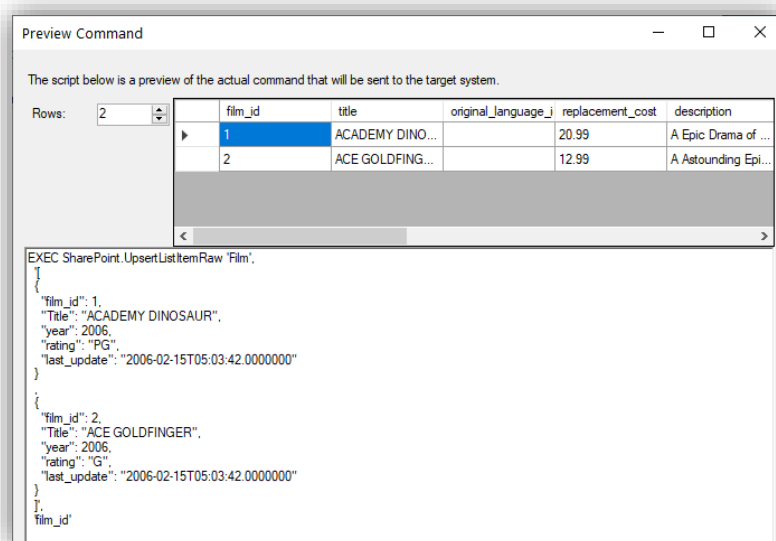
Selecting one of the "... per batch" options will automatically insert the **@concat...** operation.



The Upsert script will automatically be modified to add the appropriate **@concat...** operation at the cursor location.

You may need to rename the properties or node names as expected by the target system. You can also include DataZen functions as needed.

To view how the script will look like during execution, click on **Preview SQL...**



As you can see, the SQL Batch is a single EXEC command, and the second parameter is a Json array containing 2 objects.

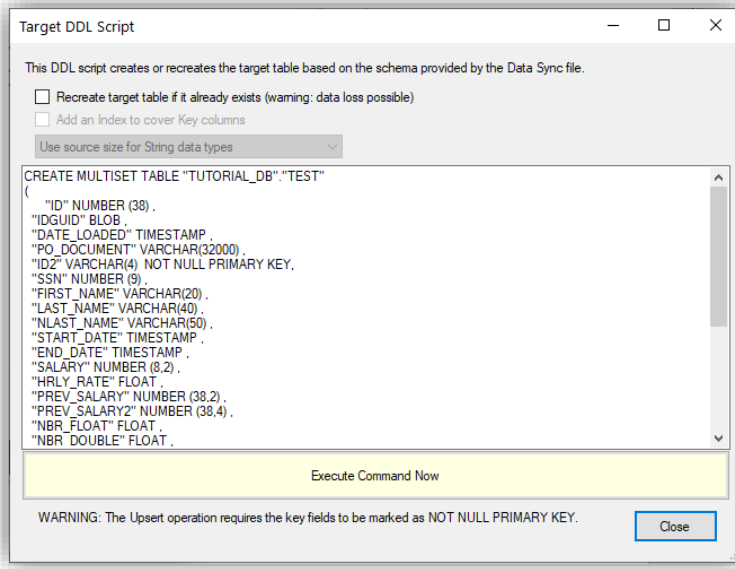
Create Target DDL Scripts

When building a Target job, you can create the target object if the system is a relational database. The target DDL script is automatically generated to facilitate the first-time creation of the destination table; however, some options may need to be reviewed by a DBA to ensure proper performance.

The options to create the target DDL script varies by data source; however, two main screen options exist for known database targets, and generic targets (Generic ODBC Driver).

Known Targets

Known database targets include SQL Server, MySQL, Oracle and Teradata. These targets provide a database creation script that is compatible with the database engine automatically. For example, a Teradata table creation script will be automatically generated based on the Change Log selected as follows:

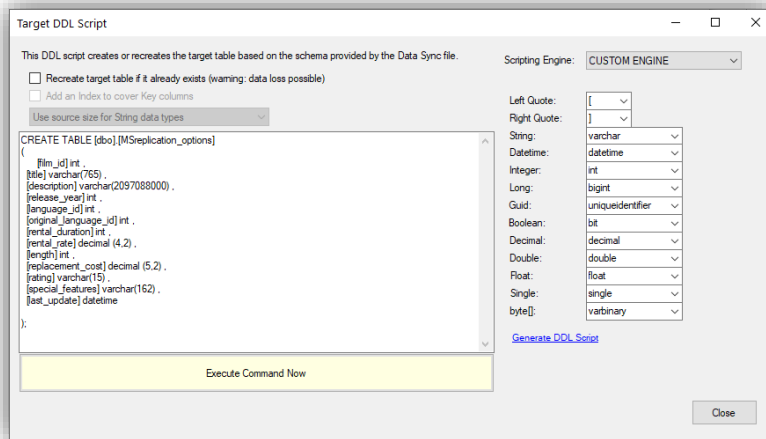


For Oracle, SQL Server and MySQL, you can optionally control the **string** data type creation based on the length of the data itself, as captured from the source system. You can modify this script directly, including adding columns and indexes.

Columns manually added to the table must be NULLABLE or have a DEFAULT value.

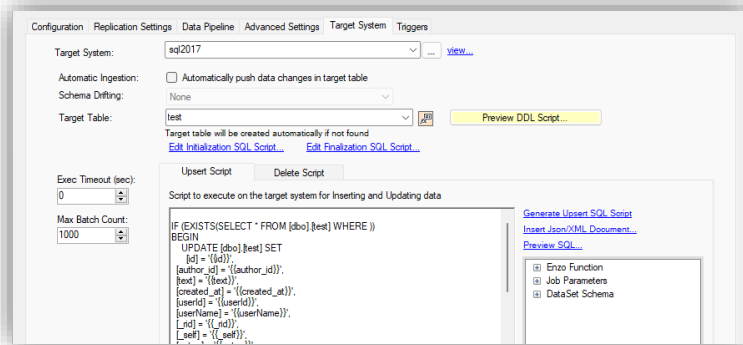
Generic Targets

When choosing a Generic ODBC Driver, additional options are available in the same screen, including the quoted identifier to use, and the target data type to use for most available source data types.

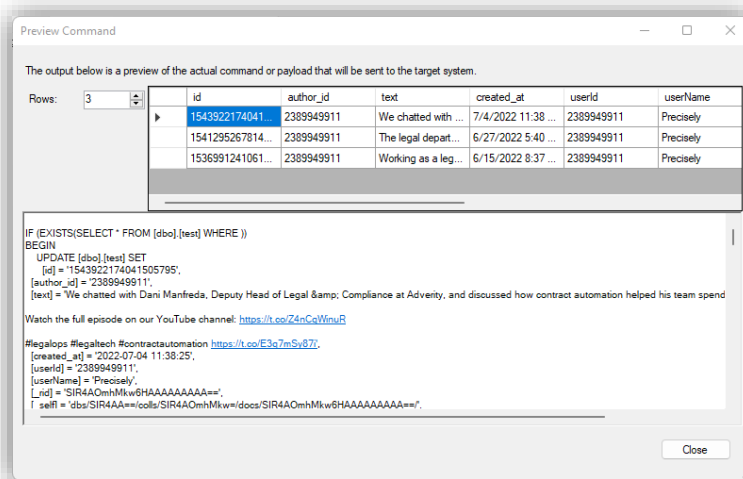


Preview SQL Commands

DataZen allows you to preview SQL commands with actual data taken from the Change Log. This allows you to visually inspect the commands being sent to the target system.



From the Target System tab, click on the **Preview SQL...** link to preview the commands that will be sent to the target system.



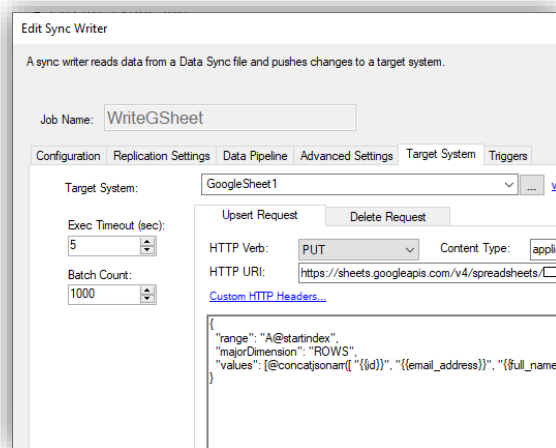
Although you cannot run this command directly, you can copy and paste it into a development environment to verify if it is working as expected.

NOTE: Changing the SQL command in this window will not update the template created in the previous screen.

Similarly, the Delete Script needs to be provided; the Generate Delete Script link provides a sample SQL command. The Delete script is only needed if the option to **Propagate Deleted Records** has been selected under the **Replication Settings** tab.

HTTP/S Targets

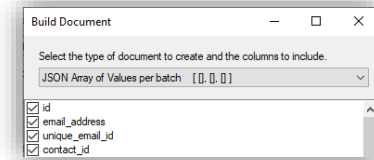
When selecting an HTTP/S, the body of the message can be crafted as a JSON, XML or free-form text content. When sending to a specific cloud platform, the payload should confirm to the expected format of the service. When necessary, you can also configure additional Custom HTTP Headers to confirm to the service endpoint requirements.



In this example, the target platform is a Google Sheet. Writing data to a Google Sheet is performed by sending a PUT request, using a Content-Type set to application/json, using the following URI:

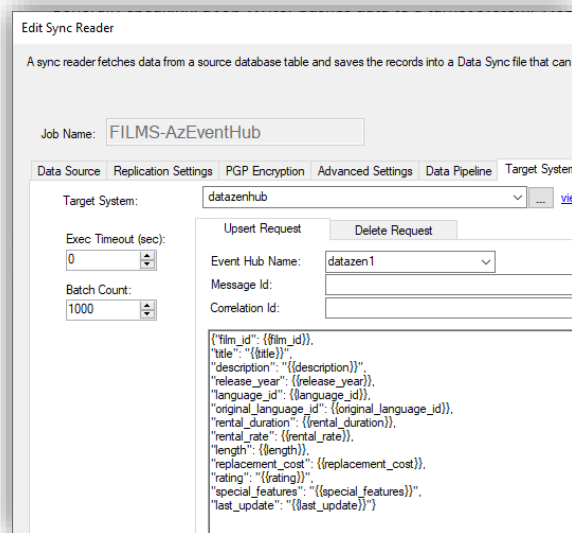
<https://sheets.googleapis.com/v4/spreadsheets/SHEETID/values/A@startindex?valueInputOption=RAW>

The payload itself is a JSON document that contains the data created as an array of JSON Arrays (@concatjsonarr).



Messaging Targets

Messaging targets allow you to specify a message payload only if the Job Reader is not a Messaging Listener with a Passthrough processing mode. The availability of the Delete Request tab depends on whether the Job Reader was configured to identify deleted records.



In this example, the target messaging platform is an Azure Event Hub. Each messaging platform has specific options that can be configured, such as the name of the target topic or queue name.

When the Payload option is available, you can specify a custom message format. Click on the **Generate Json/XML Payload** link to build a custom payload using the fields from the source Sync File or data source.

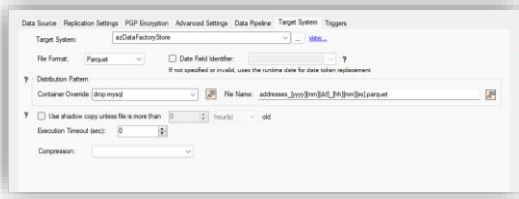
Drive Targets

You can write data read by any Job Reader into target files (Parquet, CSV, XML and JSON). DataZen includes the ability to distribute (bucketize) data dynamically using various options to spread the data across multiple files. When using a deterministic data distribution strategy, DataZen also allows you to push updates into existing files.

Common Write Options

When writing to a flat file, select the desired file format. Additional options will be displayed depending on the format selected. Some of the options provided are common to all file formats and explained

below. Additional considerations are file creation vs. updating existing files, and data distribution options.



The **Date Field Identifier** allows you to specify a date found in the source data as the value used for date replacement tokens (such as [mm]). If no field is selected, the job runtime will be used for date replacement tokens. See Date Replacement Tokens for more information.

Both the folder/container and file name can be specified with tokens and DataZen Functions.

Shadow Copy

By default, DataZen obtains a copy of a remote file before making changes to it (only the files being modified will be downloaded). Files are kept locally until the job completes, and are then uploaded as part of the completion of the job.

The shadow copy option is a performance optimization feature that allows DataZen to keep files locally cached between job executions to avoid unnecessary downloads; a time-to-live value indicates how old the file can be before downloading it from the target system. This is primarily useful if the files are very large and do not change frequently. For example, if a file could be modified daily on the target store, you could set the shadow copy to refresh the file every 6 hours.

Create New Files

DataZen will create a file if the folder/container and file name are not found in the target drive. However, in some cases you may want to force the creation of new files for certain scenarios, such as creating a delta lake. This can be done by controlling the naming convention of the target folder and/or file name in such a way that the combination of the folder and file name is never the same.

Here are a few examples on how to force the creation of new files:

- Leave the **Date Field Identifier** option unchecked to use the job runtime for date replacement tokens, and create a unique folder\filename combination:
 - o **container:** files **file name:** address-[yyyy][mm][dd]_{hh}[nn][ss].parquet
- Use a unique job setting variable:
 - o **container:** drop-@executionid **file name:** address.parquet
- Use an Enzo Function:
 - o **container:** drop-#rndguid() **file name:** address.parquet

Update Existing Files (Data Distribution/Bucketization)

DataZen includes the ability to distribute data dynamically using various options to spread the data across multiple files. When using a deterministic bucketization strategy, DataZen also allows you to push updates. This is useful when DataZen pushes changes only to the target file(s).

Here are a few examples on how to distribute data across multiple files:

- Check the **Date Field Identifier** option and select a date column from the data source used for date replacement tokens and use them as part of folder\filename combination:

- **container:** files **file name:** address-[yyyy][mm][dd]_[hh][nn][ss].parquet
- Use other fields from the source data set in the folder\name combination:
 - **container:** drop-{{country}} **file name:** address-{{state_code}}.parquet

Bucketization is the process of distributing data across multiple files in a way that simplifies future read access to the data. Determinism ensures that the name in which the row needs to be saved is based on the data itself, so that the name of the file is predictable.

Date Replacement Tokens

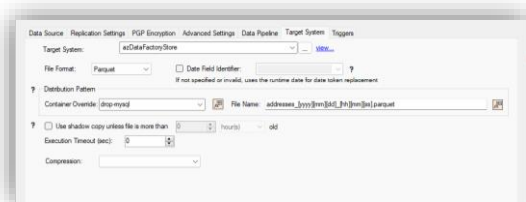
A **processing date** is used to provide a date dimension available for the bucketization process; by default, the processing date is the current time of the job when being executed; however, it can be set to a known field name from the data set being processed. Bucketization of the data is done by naming the file using specific tokens, and using these tokens as filters for the data being processed. The available tokens are as follows:

Token	Description	Example
[yyyy]	The 4-digit year of the processing date	2021
[yy]	The 2-digit year of the processing date	21
[mm]	The month of the processing date	1
[dd]	The day of the processing date	27
[doy]	The day of the year of the processing date (from 1 to 366)	159
[dow]	The day of the week of the processing date (0: Sunday)	4
[hh]	The hour of the processing date (from 0 to 23)	17
[nn]	The minute of the processing date (from 0 to 59)	44
[ss]	The seconds of the processing date (from 0 to 59)	30

Parquet Files

DataZen was optimized to read and write Parquet files efficiently. Internally, DataZen manages data type transformation automatically if necessary and dynamically changes the schema of a Parquet file when detecting schema drifting (for example, an Integer becoming a Long value).

When writing files, an additional option is provided allowing you to choose your compression algorithm.



The **Compression** option allows you to choose from: None, GZip, and Snappy. By default, **Snappy** compression is used if unspecified.

The following conversions occur automatically:

- **Guid -> String**
If the data set contains a **Guid** data type in its schema, it is automatically converted to a **string**

data type in the Parquet file. In addition, a custom metadata is added tracking the conversion: `DZ_CAST_GUID::{fieldname}`. For example, if the **userId** field is a **Guid** in the source data, the metadata added will be **DZ_CAST_GUID_userId**

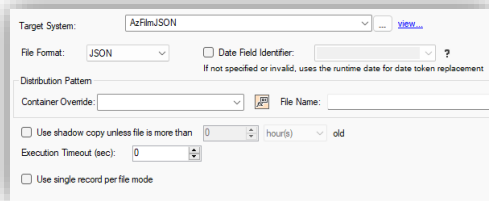
- DateTime -> DateTimeOffset

If the data set contains a **DateTime** data type in its schema, it is automatically converted to a **DateTimeOffset** data type in the Parquet file. In addition, a custom metadata is added tracking the conversion: `DZ_CAST_DT::{fieldname}` or `DZ_CAST_DT_UTC::{fieldname}`. For example, if the **createdOn** field is a **DateTime** in the source data and the date field is a UTC format, the metadata added will be **DZ_CAST_DT_UTC_createdOn**

If the above metadata items are found in Parquet files, DataZen performs the reverse conversion when reading the file.

JSON Files

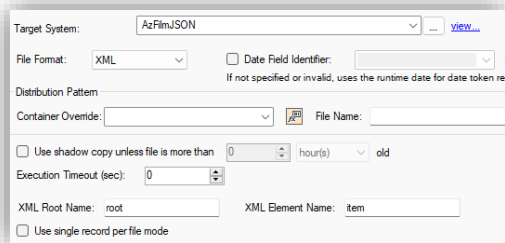
You can emit JSON files by selecting JSON as the file format.



To create a single output file per record, check the **Use single record per file** option. You should ensure that the file name being created is unique for every record following the guidance provided in the **Data Distribution/Bucketization** section.

XML Files

You can emit XML files by selecting XML as the file format. DataZen can emit simple XML files in the form `<root><item></item></root>` or `<item></item>` format in the **Single record per file** mode.



To create a single output file per record, check the **Use single record per file** option. You should ensure that the file name being created is unique for every record following the guidance provided in the **Data Distribution/Bucketization** section.

Specify the XML Root element, and the main element name to be created. When selecting a single record per file, the root element is ignored and the output is a single XML node.

CSV Formatting Options

When exporting to a flat file (CSV File Format) additional options become visible.

Column Delimiter: the field delimited to use; default: \t (tab).

Fixed Fields Width: comma-separated list of widths to use for the fields being exported.

Add Header Row: when checked, adds a row containing the name of the columns for each field.

Force Quoted Identifiers: when checked, adds a double-quote to all fields and values. When unchecked, only adds double-quotes when a field name or value contains a delimiter or contains new-line characters.

Flatten Text: removes new-line characters from text fields.

Trim Whitespace: Removes trailing white space characters.

Date/Time Format: Date/Time default formatter using the local culture for date fields. A sample output is provided adjacent to the field.

Big Data/NoSQL Targets

Google Big Query

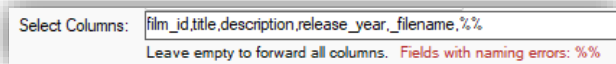
DataZen supports writing data to Google Big Query tables. By design, Google Big Query Tables do not support updating records; it is an insert only database.

On the top of the screen, select a Google Cloud Big Query Target System and enter a DataSet and Table name. To create them automatically, choose **Create if not found**. You can use DataZen Functions to create random tables.

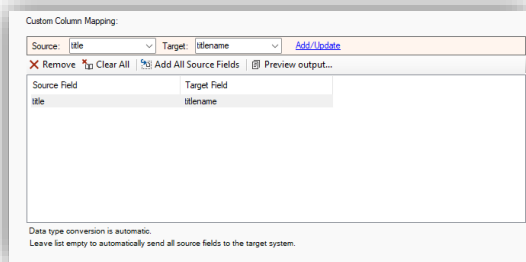
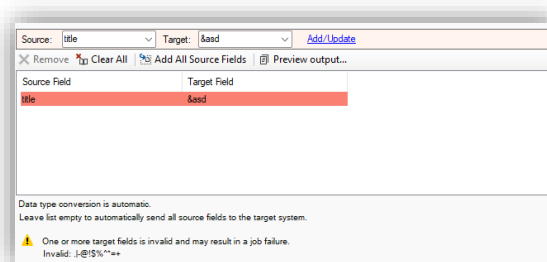
The Write Behavior option is an important factor to ensure data successfully lands in this database. You can choose between:

- Append to existing data (Write Append)
- Truncate Table First (Write Truncate)
- Only write if table is empty (Write If Empty)

To push specific columns from the source data, enter the list of columns separated by a comma. Click on the **pick** link to choose from an available list of columns, or leave this field empty to send all available columns. If a column name fails naming validation, an error will be displayed.



You can further map source column names to new ones in the target database. In the example below, the **title** column will be named **titlename** in the database. If the column name provided is invalid an error will be displayed.

CosmosDB

CosmosDB allows Insert, Update, and Delete operations. To write to a CosmosDB database, select a connection configured to connect to an Azure CosmosDB endpoint. Enter the database and container name, or select an existing one. You can use DataZen Functions to create them dynamically.

The **PartitionKey Path** allows you to indicate how records are to be partitioned in CosmosDB. By default, the **'id'** field will be used.

Because CosmosDB stores data as JSON documents, you can create your own JSON template; check the **Build a custom JSON Document** option to see additional options. If unchecked, the JSON document will be created automatically by DataZen.

Automatic JSON Document Generation

When the **Build a custom JSON Document** option is unchecked, you have multiple options to control field mapping and column selection. You can choose which columns to send to CosmosDB by entering a comma-separated list of columns in the Selected Columns field; click the **pick** link to select from the source data set. Leave this field empty to send all available fields.

*CosmosDB expects an 'id' field in the data set. If this field doesn't exist, it can be created automatically using the **PartitionKey Behavior** option selected.*

If the data source does not contain an 'id' field, or you would like to use another field for the 'id' value, select the desired **PartitionKey Behavior** option.

cosmosdbTweets [view...](#)

Account: enzodev

Database: [FX](#) ☐ Create if not found

Container: [FX](#) ☐ Create if not found

PartitionKey Path: /id

☐ Build a custom JSON Document (must have an 'id' field)

Selected Columns: [pick...](#)
Leave empty to forward all columns.

PartitionKey Behavior: Use existing 'id' field from the source [FX](#) ☐ Use this option if the source data doesn't have an 'id' field. Note: the 'id' field is case-sensitive in CosmosDB.

The following screenshot shows you how to map the **film_id** field to the 'id' attribute in the JSON document. You can also use DataZen Functions to generate unique values or combine fields.

PartitionKey Behavior: Set or create the 'id' field from another column

Field or function: film_id [FX](#)

Unless you are building a custom JSON document, you can map field names if you want to modify the source fields. If the target field name is invalid, an error will be displayed.

Custom Column Mapping:

Source: title Target: title_name [Add/Update](#)

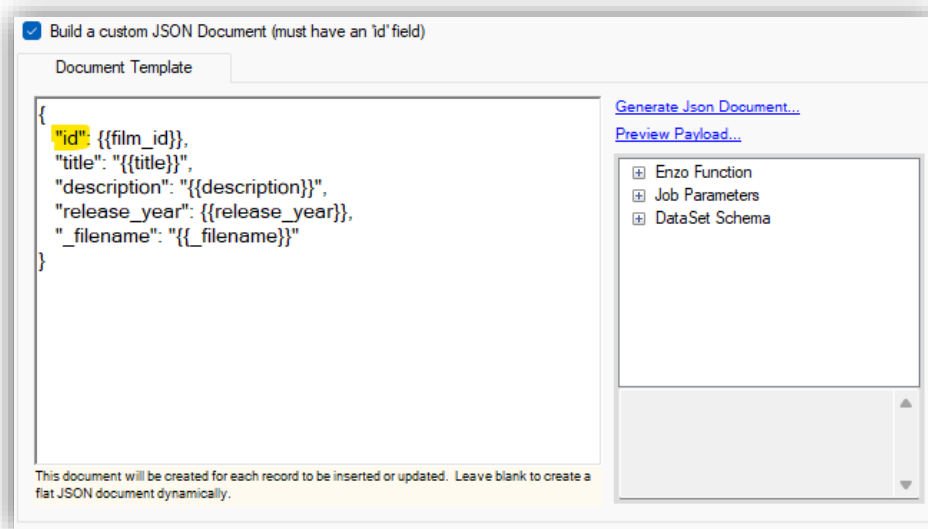
Source Field	Target Field
title	title_name

Data type conversion is automatic.

Build a Custom JSON Document

When the Build a Custom JSON Document is checked, you can create your own JSON document template and use DataZen Functions as needed. You can also preview the JSON payload. If you use this

option, you must ensure your JSON document contains an “**id**” field as highlighted in the following screenshot.



Data Sync Files

Data Sync files contain schema information about the source system, including the number of records modified, and the date/time of the data extraction. These files also contain the actual data from the source system in a compressed format. For advanced security, the files can be encrypted using PGP. Conceptually, these files are similar to backup files, except they store data in a universal format that can be sent to any target system in the correct format.

The naming convention of the Data Sync File ensures that each file can be replayed in the proper sequence. This ensures that changes are applied in the correct order when being replayed.

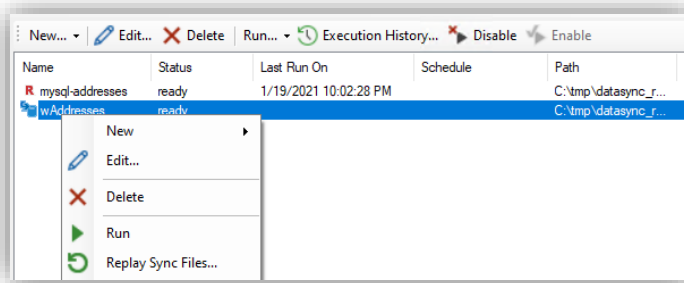
In addition, a “.EFS” extension of the sync file indicates the file contains all the source records (such as an initialization or a full data resync), while a “.EDS” extension indicates that it contains changes that took place since the last Data Sync File was generated. EFS change logs are usually much larger than EDS files.

Replay Sync Files

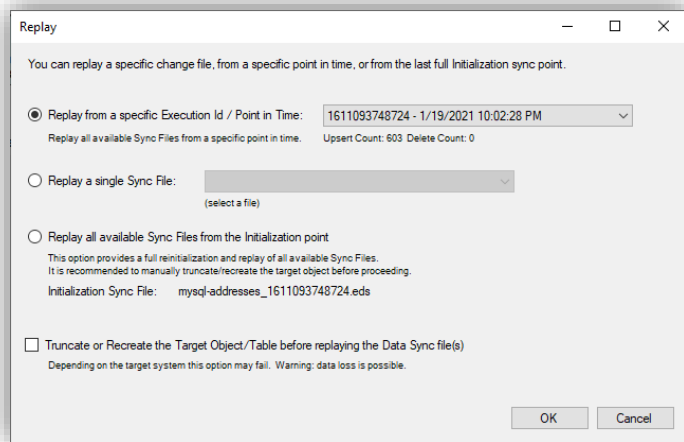
You can, at any time, choose to replay a Sync File (or multiple Sync Files) on a target system. Depending on the target system and the options selected, replaying Sync Files allows you to:

- Recreate a target table with the last known state of the source system
- Replay a single Sync File if the target system was temporarily unavailable

To replay a Sync File, select a Job Writer, and follow these steps:



Select the Job Writer where you would like to replay Sync Files, then right-click on it and choose **Replay Sync Files...**



You can replay from a specific point in time (the dropdown shows the Execution Id and its creation date), choose a single Sync File, or replay from the **Initial Execution Id** that you selected when the Job Writer was created.

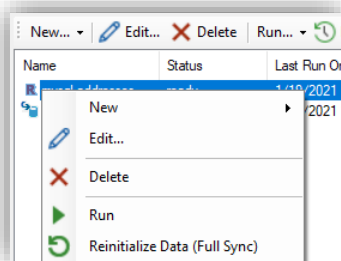
Depending on the target system, you may have the ability to Truncate or Recreate the target table before replaying the Sync File.

Click OK to start replaying the Sync File(s).

Resync Data

DataZen Manager allows you to resync the data from the source system by recreating a full initialization file (**Full Sync**). Once the Sync File has been created with all the records, it will be processed automatically by all the Job Writers that have a schedule defined.

To create a full Sync File, follow these steps:



Select the Job Reader you would like to resync, then right-click on **Reinitialize Data (Full Sync)**.

A window will come up asking to confirm the resync operation; click **Yes**.

Once completed, a new Sync File will be created with an **EFS** extension.

Resync operations may not work as expected for CDC data sources since the CDC and Change Tracking tables only hold recent changes made to the data.

Full Sync Files

Sync Files will contain a full copy of the source records when:

- The Job Reader is first executed
- A Resync operation is performed on an existing Job Reader
- A field is modified on all records in the source system (such as a timestamp)
- A field has been added or removed, and it was previously retrieved by the SELECT statement

While all the above scenarios will create a Sync File with all the records, only the Initialization and Resync operations will create a Sync File with the **EFS** extension (Enzo Full Sync). All other files will have the **EDS** extension (Enzo Delta Sync).

Full Sync Files for CDC data sources only contain the data found in CDC and Tracking tables; you may need to create a different job to read all records from these tables.

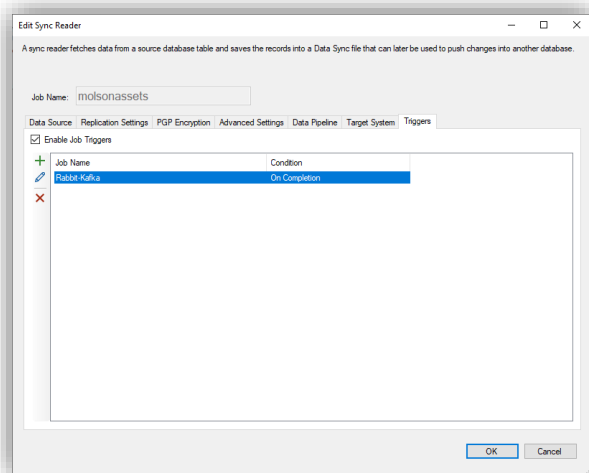
Job Triggers


Job Triggers allow you to start other jobs upon completion of the current job where the triggers are defined. Job Triggers are available for Job Readers and Job Writers

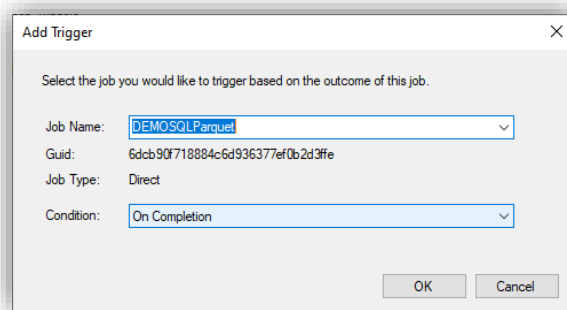
Job Triggers are not available for Messaging Jobs since they run continuously.

Multiple jobs can be triggers with different conditions: On Completion, On Success (always), On Success (with data), and On Failure.

Condition	Description
On Completion	The trigger starts regardless of the success or failure status of the current job.
On Success (Always)	The trigger starts only if the current jobs succeeds whether any records were fetched or processed.
On Success (With Data)	The trigger starts only if the current jobs succeeds and at least one record was fetched or processed.
On Failure	The trigger starts only if the current jobs failed to execute.



In the Triggers tab, click on the  icon to add a new trigger. Triggers are started virtually at the same time but in no specific order.



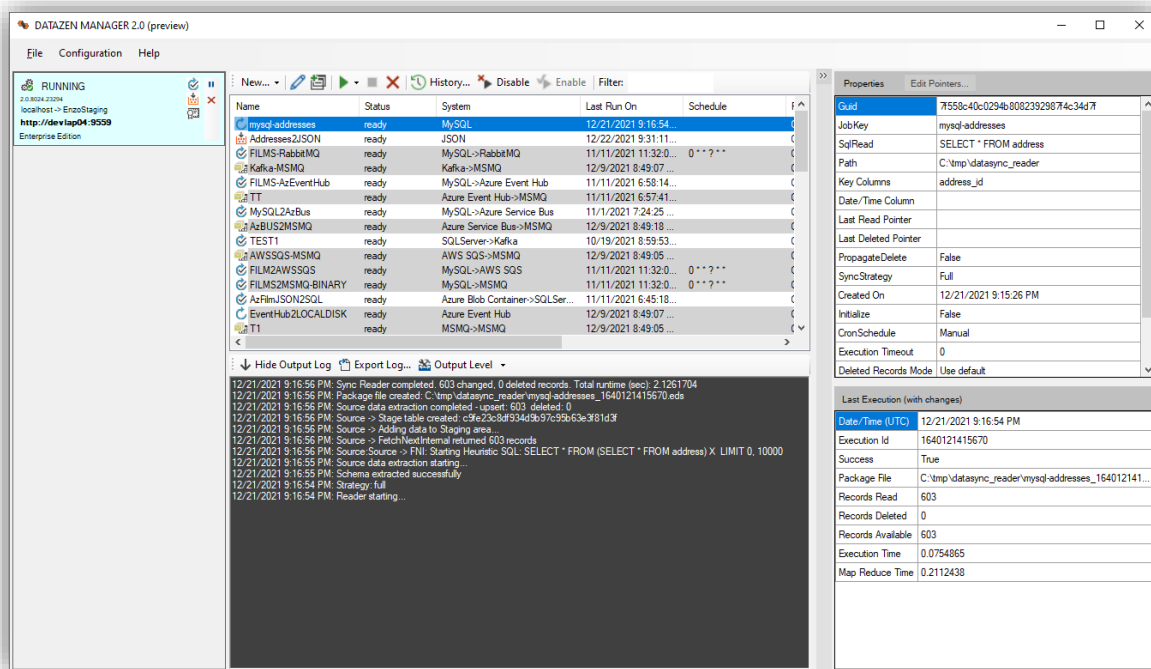
When adding a trigger, start typing for the job name, or select it from the list. Select the desired condition for executing the trigger and click OK.

Logging

Execution Log

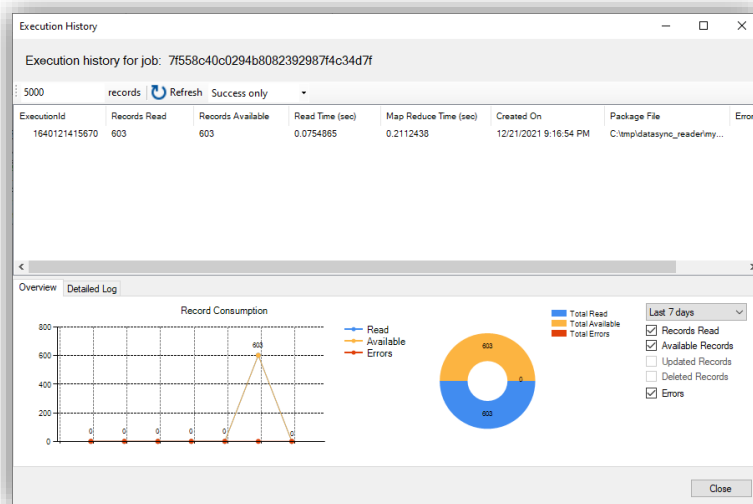
DataZen exposes its internal operation log for more detailed information and to validate that the Jobs are working as expected. The output window in DataZen Manager shows the last few entries available in memory. To view the full log, and previous logs, see the **Execution History** section.

Execution logs are automatically stored by DataZen for auditing purposes.



Execution History

DataZen Manager allows you to see the history of past Job Reader and Job Writer executions. Select the Job you would like to review and click on the **Execution History** button.



By default, the execution history shows the jobs that resulted in data being available for processing. When Job Readers find no new changes, or Job Writers find no Sync File to apply, a history record is created with 0 Records Available.

To view all executions, including those with 0 Records Available, choose the **Show All Executions** option in the dropdown box.

Runtimes and execution date/time is also provided on this screen. Any processing errors are displayed as well.

To view the execution log of a specific job execution, click on the desired execution and view the **Detailed Log** tab.

Execution History

Execution history for job: 852e16ebf45342fb8c96b21244993b4f -> 852e16ebf45342fb8c96b21244993b4f

5000 records Refresh Success only

ExecutionId	Records Read	Records Available	Records Updat...	Records Deleted	Read + Write Time ...	Map Reduce Time (sec)	Start Time
1663012499114	3	1	1	0	21.3648806	0.1475889	9/12/2022 7
1660413474590	3	3	3	0	4.2599184	0.0074872	8/13/2022 5
1660413422375	3	2	2	0	4.6655516	0.0170077	8/13/2022 5
1647111291862	3	2	2	0	5.0436421	0.0174487	3/12/2022 6
1641759041096	3	3	3	0	0.6469907	0.0837576	1/9/2022 8:

Overview Detailed Log

Date/Time	Log	Level
8/13/2022 5:57:02 PM	Writer configuration settings updated... applying changes now	Debug
8/13/2022 5:57:02 PM	Reader starting...	Debug
8/13/2022 5:57:02 PM	Source data extraction starting...	Debug
8/13/2022 5:57:02 PM	HTTP Source -> Calling HTTP endpoint: https://us3.api.mailchimp.com/3.0/lists	Debug
8/13/2022 5:57:07 PM	HTTP Source -> Fetch completed. Total byte count: 17946	Debug
8/13/2022 5:57:07 PM	HTTP Source -> Call was throttled (calls limit) 2.31 seconds	Debug
8/13/2022 5:57:07 PM	HTTP Source -> Column Document to Resultset transformation successful	Debug
8/13/2022 5:57:07 PM	HTTP Source -> Found 3 records	Debug
8/13/2022 5:57:07 PM	HTTP Source -> Returned 3 records	Debug
8/13/2022 5:57:07 PM	HTTP Source -> Calling HTTP endpoint: https://us3.api.mailchimp.com/3.0/lists/#name=2	Debug

Close

Data Pipeline

A Data Pipeline is a data transformation processing engine that modifies the data before the data is saved. A Data Pipeline can help implement the following components:

- Data Filtering**
 Applies a secondary filter to the data by adding a SQL Where clause as a Data Filter. This can be useful when the data source doesn't support a full SQL syntax, or when filtering needs to occur after hashing or masking has occurred.
- Data Hashing**
 Applies a hash algorithm to a selected data column (must be a string data type); supported hashing algorithms are MD5, SHA1, SHA256, SHA384 and SHA512.
- Data Masking**
 Applies masking logic to a selected data column, such as credit card number or a phone number. Supports generating random numbers, free-form masking, and generic / full masking.
- Data Quality**
 Validates simple rules against a data column and throws a warning or an error if the data doesn't match the specified rules.

- **Dynamic Data Column**

Adds a new column to the output as a constant value, a simple SQL formula, or an Enzo Function.

- **Remove Columns**

Remove unwanted columns from the final output.

Both Job Readers and Job Writers can implement a Data Pipeline. When implemented by a Job Reader, a Data Pipeline modifies the data before it is saved in the Sync File; this ensures that any Job Writer reading from the Sync File will get the same data modifications. When applied to a Job Writer, the data is modified before it is saved to the target system. This allows pushing the same data to different systems using different Data Pipeline masking rules and filters.

address_id	address	address2	district	city_id	postal_code
1	A677F8FC41887B543686AF64EA78FFB2AF59E25		AlXXXXXXXXta	300	
3	0282D44CA89C019E65730914312A86594D1137EB		AlXXXXXXXXta	300	

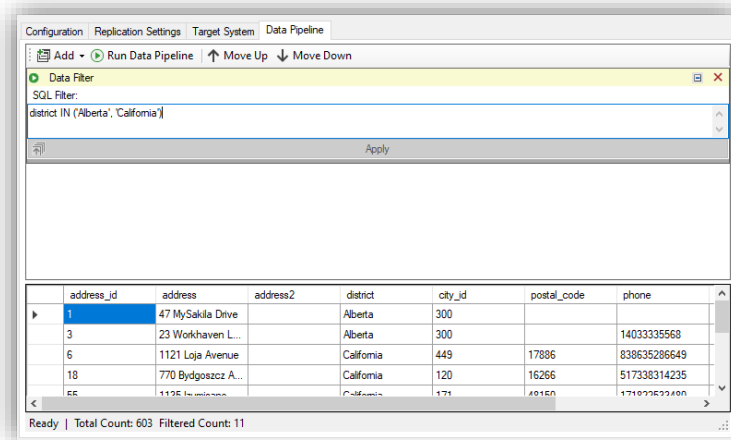
Multiple components can be added to a pipeline; they are executed in the order provided, from top to bottom. You can move the components using the **Move Up** and **Move Down** buttons. You can also disable a component without removing it; disabling a component bypasses its processing, but keeps it saved so it can be re-enabled in the future.

Click the green arrow to enable/disable a component.

For example, the Data Pipeline above filters the records to the Alberta district, hashes the address field, and masks the district according to the rule specified.

Data Filter

The Data Filter component allows you to apply a SQL-like filter to the data set, so that only the matching records are replicated. When applied to a Job Reader, the data is filtered before being saved to the Sync File.

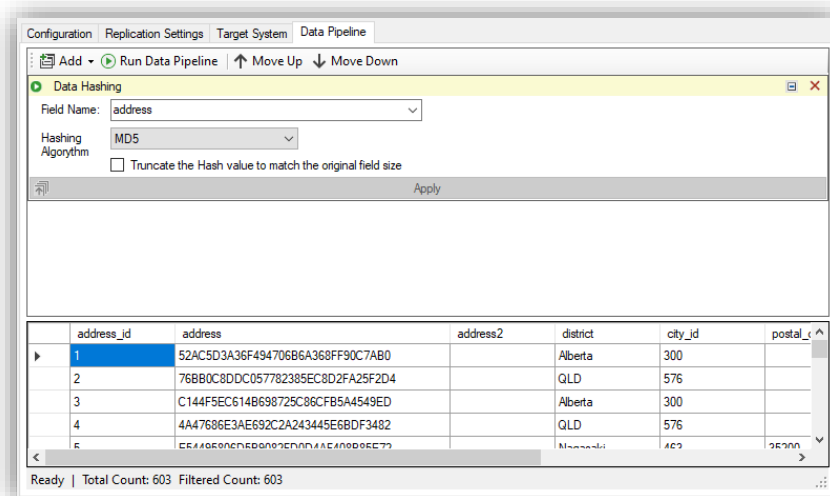


When applying a filter, you can see how many records are returned by the filter at the bottom of the screen (showing 11 records in this example).

You can apply multiple filters using an SQL-like syntax. See the Microsoft DataColumn Expression documentation on the available syntax for the filter.

Data Hashing

The Data Hashing component allows you to hash a Column of a string data type to a hash value, using either MD5, SHA1, SHA256, SHA384 or SHA512 algorithms. Because no vector/salt is generated, the hash values generated are always the same for the same input string. However, even a slight change in the input string will generate a very different hash output. In the example below, the **address** field is hashed using the MD5 algorithm. You can add as many Data Hashing components if you need to hash multiple fields. If the target system expects a specific length for the field being hashed, you can check the **Truncate Hash Value** option; this will truncate the hash value to match the original string length; however, this may reduce the strength of the hashing algorithm in its ability to generate unique values for different input values.

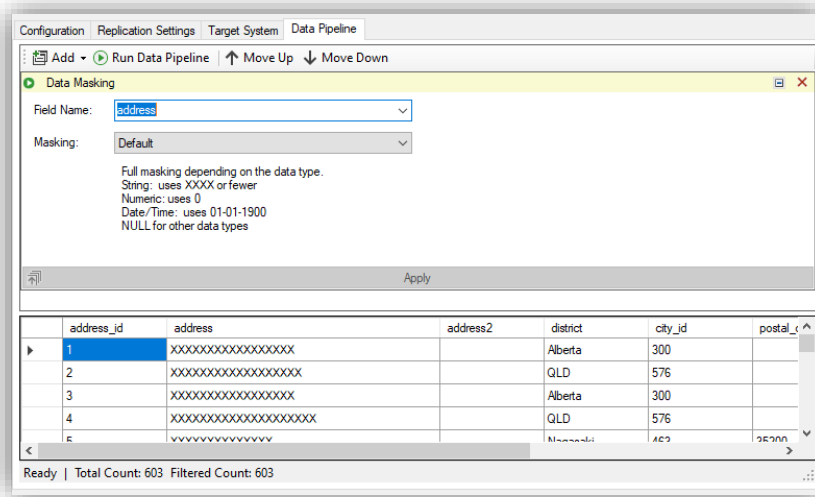


Data Masking

The Data Masking component allows you to modify for privacy or compliance reasons. For example, you may choose to mask a Phone Number field, or a Social Security number. Five masking options are available:

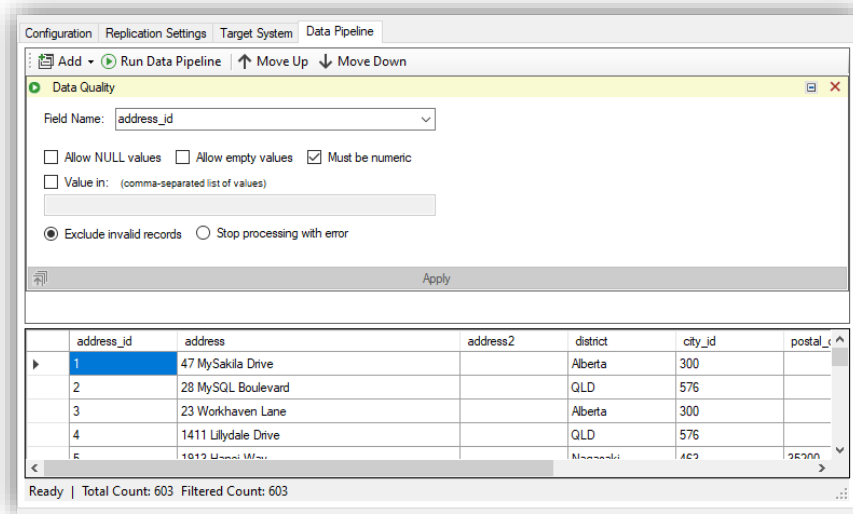
- Default – replaces a string field with the character X; replaces a numeric field with 0; replaces a date field with 1900-01-01 and NULL for other data types
- Credit Card – Exposes the last 4 digits of the credit card and replaces all other characters with X
- Email – Exposes the first character, and replaces the remainder with XXX@XXX.com
- Random Number – Replaces a numeric field with a random value based on the data type
- Custom String – Exposes the first few and last characters, as desired, and places a constant string in the middle as provided

As many Data Masking components can be added, when masking multiple fields is needed. In the example below, the default masking option is used on the address field.



Data Quality

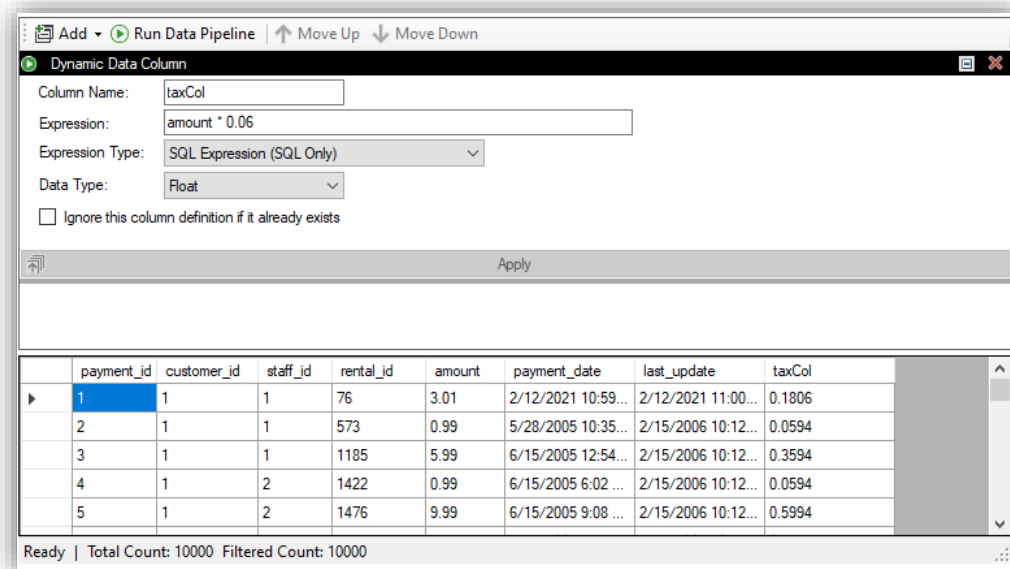
The Data Quality component allows you to validate basic rules against a field, such as making sure the field is numeric, not empty, not null, and is within a list of known values. For example, the following Data Quality component ensures the **address_id** field is not NULL, not empty, and is a numeric value. If the condition fails, DataZen can either exclude the records from the output, or throw an error and stop further processing.



Dynamic Data Column

The Dynamic Data Column component allows you add columns dynamically to a data set, specify its data type, and use existing fields to construct it if desired.

For example, the following screenshot shows you how to add a dynamic column, named **taxCol**, that calculates a tax amount of 6%, with a SQL Expression that uses the **amount** field (**amount * 0.06**), and setting the column as a **Float** data type.



The expression can either use a SQL-like syntax, or a simpler Enzo format.

SQL Expression

The SQL Expression treats the formula as a simpler SQL command, allowing you to leverage a number of operations on data. Because the SQL syntax uses a limited version of the T-SQL syntax, string quotation

marks are single quotes. Operations such as in, between, like, not, or, and are available. You can use brackets when using field names. The following functions are allowed: CONVERT, LEN, ISNULL, IIF, TRIM, and SUBSTRING. Here are a few valid examples:

- SUBSTRING(phone, 7, 8)
- IIF(ISNULL(amount), 0, amount * 10)

See Microsoft's documentation on DataColumn Expressions for a complete description of.

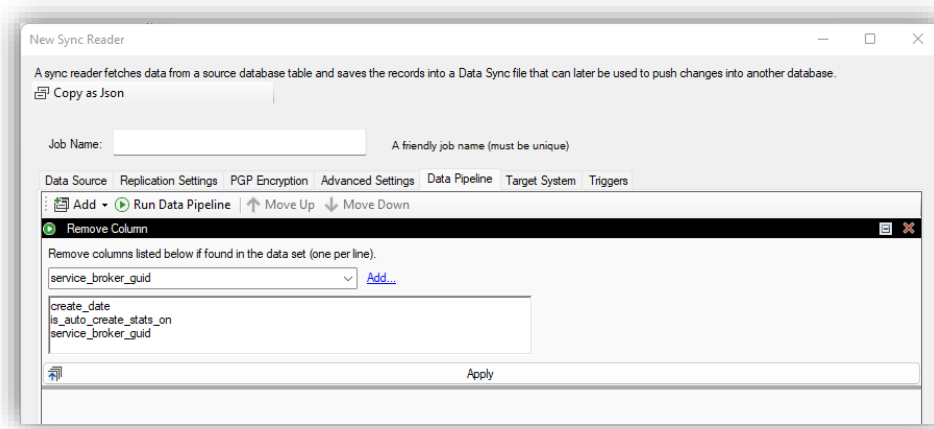
Enzo Expression

If an Enzo Expression is used, every row is processed individually using the DataZen Functions. These must be simple expressions that can use other fields using the {{field}} convention. For example, if the Expression is #rndguid(), the new column will have a unique GUID value assigned in every row.

See the DataZen Function section for more information.

Remove Column

The Remove Column component allows you to ensure a specific column does not get replicated. You can either select an existing column from the list provided, or type the name of a column (one per line).



DataZen Functions

DataZen functions give you the ability to transform data before data is processed for execution, but after the data values of the source dataset have been replaced. These functions are only available in multiple places (see the Uses section below).

Functions are identified by a # moniker. For example, the "#toiso8601({{payment_date}})", converts the value of the "payment_date" field from the source system into an ISO 8601 compatible format.

Function	Description	Comments
#decrement(a) #decrement(a, b)	Decrements a value by one, or by the amount specified.	#decrement({{field1}}, 1)

#eval(code)	Treats the inner parameter as a C# script and executes its content. Expects the output of the script to be a string or an object that supports the .ToString() method.	Use this function to execute a more complex C# operation. See the Microsoft documentation for the CSharpScript.EvaluateAsync<dynamic> operation.
#format(a, b, ...)	Uses the C# string.Format syntax to build a string with any number of parameters	The C# syntax uses {0} for the first parameter, {1} for the second, and so forth. #format('User {0} lives in {1}', {{userId}}, {{state}})
#formatdate(date, format) #utcformatdate(date, format)	Formats a string value into a date with a specific format (following the C# DateTime.ToString(format) notation)	#formatDate(11/01/2000T08:00:00.0000, 0:MM/dd/yy H:mm:ss)
#increment(a) #increment(a, b)	Increments a value by one, or by the amount specified.	#increment({{field1}}, 25)
#http_get(...) #http_post(...) #http_put(...) #http_delete(...)	Performs an HTTP/S call to an internet resource and returns a string as output. If the call to the HTTP resource is an image, the content is returned as an Hexadecimal representation of the bytes of the image. See the Enzo HTTP Functions section for more information.	#http_get({{field_url}})
#isnullorempy(a,b)	Returns value 'b' if value 'a' is a NULL value or an empty string	You can use this function to replace a NULL value with a constant, or the content of another field #isnullorempy({{field1}}, 0)
#left(a, n)	Returns the left-most n characters of a string	#left('this is a test', 5)
#now()	Returns the current datetime in the local timezone of the server	
#pick(a,b,c,...)	Randomly selects a value from the list provided as parameters	Any number of parameters can be provided
#replace(a, b, c)	Replaces all occurrences of b with c, in string a	#replace('this is a cat', 'cat', 'dog')
#right(a, n)	Returns the right-most n characters of a string	#right('this is a test', 5)
#rnddouble(a,b)	Generates a random double value between two numbers	Both the lower and upper bound values are required
#rndguid()	Returns a new random GUID	

#rndint(a,b)	Generates a random integer value (Int32) between two numbers	Both the lower and upper bound values are required
#tohex64(a)	Transforms a value into an Hex64 representation	#tohex64('this is a test')
#toiso8601(x)	Converts a datetime field into a ISO 8601 compliant format	This function converts a datetime field into a string in the ISO 8601 format assuming the date is in the current timezone
#torfc1123(x)	Converts a datetime field into a RFC 1123 compliant format	
#utcnow()	Returns the current datetime in UTC format	
#utctoiso8601(x)	Converts a UTC datetime field into a ISO 8601 compliant format	This function converts a datetime field into a string in the ISO 8601 format assuming the date is in the UTC timezone
#utctorfc1123(c)	Converts a UTC datetime field into a RFC 1123 compliant format	
#urlencode(c)	URL encodes the value provided so it is safe to pass as a URL parameter.	

Nested Functions

You can nest functions by adding them together in a call. For example, the following operation gets the current time in UTC, returns an ISO8601 compliant date format, and URL encodes the output so it can be passed in a URL function:

```
#urlencode(#toiso8601(#utcnow()))
```

Uses

Enzo Functions can be used in multiple places throughout DataZen, namely:

- Data Pipelines Dynamic Columns
- URL of a Sync Job using an HTTP Connect as the source
- Custom SQL Scripts in a Database Target operation
- Name of a database table a database Target
- URI Endpoint, XML or JSON payloads in an HTTP Target
- Initialization and Finalization database scripts when using a DB Target

Enzo HTTP Functions

Calling HTTP Endpoints within a Data Pipeline is a feature that allows you to use the data provided by the job as input records. In other words, for each input record, an HTTP method will be called. Enzo HTTP Functions can be invoked in multiple ways to make both authenticated and unauthenticated calls to HTTP REST endpoints.

The parameters used by the HTTP functions depend on whether or not the call is authenticated and the type of call being made. The following signatures are allowed:

Function	Param 1	Param 2	Param 3	Param 4	Description
http_get http_delete	uri				Call to HTTP endpoint without authentication
	headers	uri			Call to HTTP endpoint with custom headers without authentication
	@conn(key)	uri			Call to HTTP endpoint using a pre-defined HTTP connection string
	@conn(key)	headers	uri		Call to HTTP endpoint with custom headers using a pre-defined HTTP connection string
http_post http_put	body	uri			Call to HTTP endpoint without authentication with the specified body
	body	headers	uri		Call to HTTP endpoint with custom headers without authentication with the specified body
	@conn(key)	body	uri		Call to HTTP endpoint using a pre-defined HTTP connection string with the specified body
	@conn(key)	body	headers	uri	Call to HTTP endpoint with custom headers using a pre-defined HTTP connection string with the specified body

When specifying a pre-defined connection, use the following format: **@conn(key)**. The key should be the name of the Central Connection String to use that represents the HTTP Endpoint, its authentication mechanism, and Rate Limiting setting. HTTP Enzo Function calls can use any of the supported authentication mechanisms including API Keys and OAuth 2.0 bearer tokens.

When specifying custom headers, the headers parameter should be comma-separated, and the header itself should be in the normal HTTP header notation: **header:value**. When specifying multiple headers, the parameter should be single-quoted: **'header1:val1,header2:val2'**.

Example 1: Unauthenticated HTTP Get

The following example performs an HTTP GET operation and returns a string from a service endpoint running on the localhost without authentication:

```
http_get(https://localhost/func1)
```

Example 2: Unauthenticated HTTP Post with Headers

Sends an HTTP POST operation to a localhost endpoint unauthenticated, passing a body and a custom Content-Type header. Sends the **userKey** and **loginId** column values for each row processed by the job.

```
http_post({'id': {{userKey}}, "userId": "{{loginId}}"}, 'Content-Type:text/json', https://localhost/func1)
```

Example 3: Authenticated HTTP Get

Sends an HTTP GET operation to the **"TWITTER"** HTTP connection string, using the **/users** relative URI, and passing the **userid** value found in the source dataset for each record processed by the job. This call

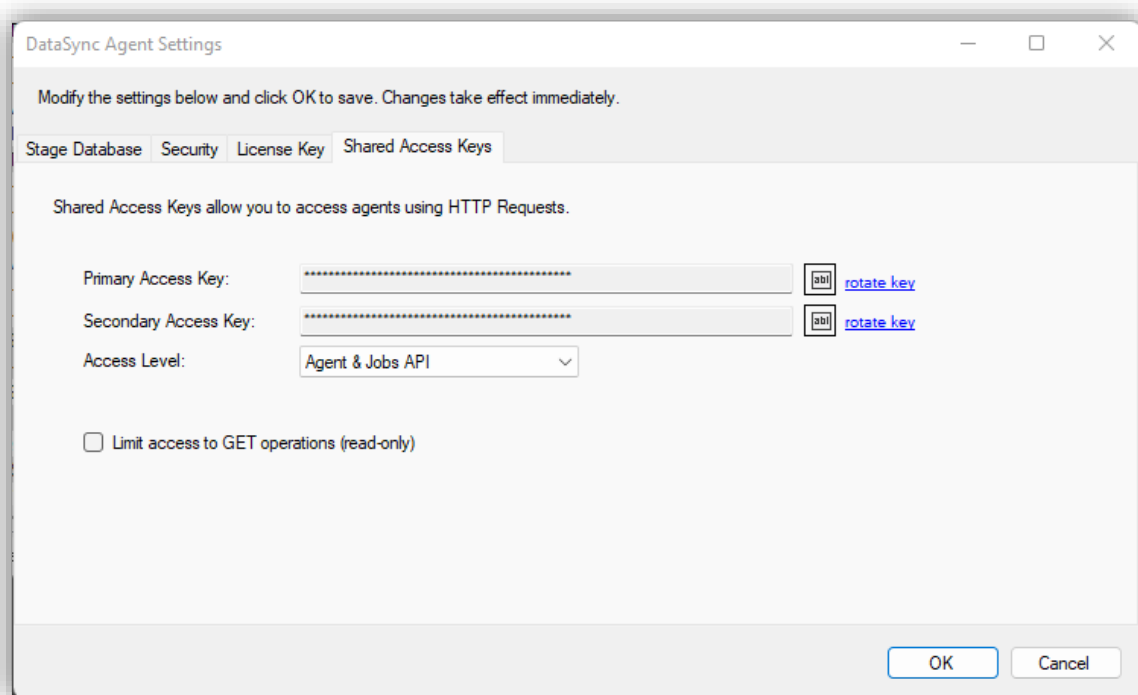
returns a JSON document of the tweets for the given **userid**; if this function is used in a **Dynamic Column** (in a Data Pipeline), the new column value will contain the JSON document returned by this HTTP request.

```
http_get(@conn(TWITTER), /users/{{userid}}/tweets)
```

Sync Agent REST API

You can use HTTP/S REST commands against a DataZen Sync Agent to perform most of the operations available in DataZen Manager, with a few exceptions for security reasons.

To enable REST access to an agent, select the agent in DataZen Manager and choose DataZen Agent Settings from the menu. Select the desired Access Level and whether to limit access to GET operations only.



The complete documentation for the Sync Agent REST API can be found online:

<https://www.enzounified.com/blog/DataZenAPIDocumentation>

Appendix

Limitations

Target DDL Script

- Building target DDL scripts is only available for SQL Server, MySQL, Oracle and Teradata databases.
- Not all data types may be accurate; some data types may be modified from the original source system to ensure transportability of the data.
- When the source system is MySQL, the schema information for string length and other data types may be inaccurate due to MySQL driver issues. The script that is generated to create the target table may need to be modified manually.
- When the source system is an ODBC driver or Enzo, the precision or decimal data types and string length sizes of the source system may not be available.

Upsert Script

- Upsert scripts generated may need to be modified to comply with the database engine
- MySQL Notes
 - The generated Upsert script depends on the ON DUPLICATE KEY UPDATE clause, which implies the table should have a key or unique index defined on the Key Columns for the script to work
- Oracle Notes
 - DataZen stores binary data types as Base64 string, so the Upsert command should contain the following functions applied:
TO_BLOB(utl_raw.cast_to_raw('{{fieldname}}'))
 - When the script is generated, date/time types fields are automatically converted as follows:
TO_TIMESTAMP('{{fieldname}}', 'YYYY-MM-DD HH24:MI:SS.FF7')

Data Replication

- Geospatial information is replicated as a binary field in the target system.

Transactional Consistency Considerations

Because DataZen is a loosely coupled replication engine, replicating data with eventual consistency, tables selected for replication are not guaranteed to be transactionally consistent by design. However, two strategies can be used when transactional consistency is required for a few tables:

- Target Logical Access
Reports and applications running against the target system should implement logic that does not display or use orphaned records (such as missing line items for an order) when possible
- Source Joined Data Sets
Implement a more complex source SELECT JOIN statement against multiple tables, so that the

data returned by the Sync Job represents all the records; use a custom stored procedure at the target to split each record and perform the necessary Upsert/Delete operations.

Data Type Conversions

The following data type mapping tables provide an overview of the supported data types and their correspondence across multiple database platforms. The direction of the replication may change the data type conversion logic.

Oracle to SQL Server Data Mapping

ORACLE DATA TYPE	SQL SERVER DATA TYPE
INT	DECIMAL(38,0)
RAW(16)	VARBINARY(16)
TIMESTAMP(4)	DATETIME
CLOB(JSON)	NVARCHAR(max)
VARCHAR2(4 BYTE)	NVARCHAR(4)
NUMBER(9)	DECIMAL(9,0)
VARCHAR2(20 BYTE)	NVARCHAR(20)
DATE	DATETIME
NUMBER(8,2)	DECIMAL(8,2)
NUMBER(*,2)	DECIMAL(38,2)
NUMBER(*,-2)	DECIMAL(38,4)
BINARY_FLOAT	REAL
BINARY_DOUBLE	FLOAT
NUMBER	FLOAT
INT	DECIMAL(38,0)
VARCHAR2(20 BYTE)	NVARCHAR(20)
NCHAR(25)	NVARCHAR(25)
VARCHAR2(80 BYTE)	NVARCHAR(80)
TIMESTAMP	DATETIME
LONG RAW	VARBINARY(max)
CLOB	NVARCHAR(max)
BLOB	VARBINARY(max)

MySQL to SQL Server Data Mapping

MYSQL DATA TYPE	SQL SERVER DATA TYPE
INT	INT
INT UNSIGNED	BIGINT
VARCHAR(100)	NVARCHAR(400)
DECIMAL(16,2)	DECIMAL(16,2)
DATE	DATETIME
CHAR(1)	NVARCHAR(4)
TINYINT	SMALLINT
SMALLINT	SMALLINT
MEDIUMINT	INT
INT	INT
INT(4)	INT
BIGINT	BIGINT
FLOAT	REAL
FLOAT(4)	REAL
FLOAT(8)	FLOAT
REAL	FLOAT

BIT	BIT
BIT(8)	VARBINARY(1)
DATE	DATETIME
DATETIME	DATETIME
DATETIME(6)	DATETIME
TIMESTAMP	DATETIME
TIME	TIME(7)
YEAR	INT
VARCHAR(20) CHARACTER SET UTF8	NVARCHAR(60)
TEXT CHARACTER SET latin1 COLLATE	NVARCHAR(max)
ENUM('a','b','c')	NVARCHAR(4)
SET('a','b','c')	NVARCHAR(20)
CHAR(1)	NVARCHAR(4)
TINYTEXT	NVARCHAR(255)
LONGTEXT	NVARCHAR(max)
BLOB	VARBINARY(max)
BINARY(10)	VARBINARY(10)
VARBINARY(10)	VARBINARY(10)
TINYBLOB	VARBINARY(255)
LOB	VARBINARY(max)