**Veeva Vault Connector User Guide for Mule 4**

**1.0.2**

**May 2019**

# Veeva Vault Connector

## Overview

The Anypoint Veeva Vault Connector lets you connect Mule flows to the Veeva Vault platform using Veeva REST API depending on the operation you configure.

This connector works with the [Veeva Vault REST, Bulk, and Streaming API](https://developer.veevavault.com/api/18.3/#authentication), depending on the operation you configure. Each API call uses an JSON/CSV request and returns JSON input stream response over an HTTPS connection. All required request headers, error handling, and HTTPS connection configurations are built into the connector.

The Veeva Vault connector can be use in mule application as an outbound connector with CRUD (Create, Retrieve, Update, Delete), Query, Download operations on following Vault Objects. These operations are discussed later in this document.

* Documents – Create, Retrieve, Update, Delete, Download, Export.
* Document Renditions – Create, Retrieve, Update, Delete.
* Vault Object – Create, Retrieve, Update, Delete.
* Picklists – Retrieve.
* Audit Trial – Retrieve.
* Query – VQL Query to query on Documents, Vault Object, Workflows.

You can configure the Veeva Vault Connector using Basic Authentication configuration with credentials to connect to a Veeva Vault instance and perform operations.

Use this guide to help you understand how you can set up and configure a basic flow using the connector.

MuleSoft maintains this connector under the Certified support policy.

## Prerequisites

To use this connector, you need:

* To be familiar with [Anypoint Connectors.](https://docs.mulesoft.com/release-notes/connector/anypoint-connector-release-notes#mule_4)
* An installation of [Anypoint Studio](https://docs.mulesoft.com/studio/7.3/to-download-and-install-studio).
* A MuleSoft Account.
* A Veeva Vault account.

You need to have basic knowledge of:

* [Mule Runtime and Concepts](https://docs.mulesoft.com/mule-runtime/4.2/)

## Setup the Environment

### Compatibility

| **Software** | **Version** |
| --- | --- |
| Mule Runtime | 4.0.0 or higher |
| Anypoint Studio | 7.0.0 or higher |
| Veeva Vault APIs | 18.0 or higher |

### Third Party Libraries Dependencies

* Apache Common Language – 2.6 or higher
* JSON Libraries – 20170516 or higher

### Using the Connector in a Mavenized Mule App

If you are coding a Mavenized Mule application, this XML snippet must be included in your pom.xml file.

<dependency>

<groupId>org.mule.extension</groupId>

<artifactId>mule-veevavault-connector</artifactId>

<version>RELEASE</version>

<classifier>mule-plugin</classifier>

</dependency>

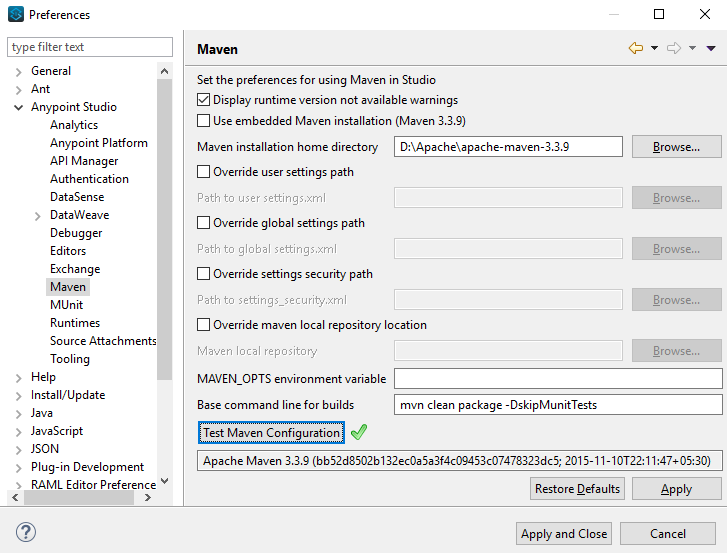
Inside the <version> tags, put the desired version number, the word RELEASE for the latest release, or SNAPSHOT for the latest available version.

### Installing and Configuring in Studio

1. Install these components:

* [Java 8.](https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
* [Apache Maven](https://maven.apache.org/download.cgi)

1. Download [Anypoint Studio](https://docs.mulesoft.com/studio/7.3/to-download-and-install-studio) and install it.
2. Configure Maven:
3. Under the **Anypoint Studio** (Mac or Linux) or **Window** (Windows) menu, click **Preferences > Anypoint Studio > Maven**.
4. Click **Browse** to locate the folder where you installed Maven.
5. Click **Test Maven Configuration**. When the green checkmark is shown, click **OK**.



1. Install the Connector from [Anypoint Exchange](https://docs.mulesoft.com/exchange/).
2. In Anypoint Studio, click the Exchange icon  in the taskbar.
3. Click **Login** in the Exchange popup and Sign in.

**Note:** To complete this step, you must have a Mulesoft account. You can create an account on the [Mulesoft Sign up](https://anypoint.mulesoft.com/login/#/signin) page.

1. Enter **Veeva Vault**in the search box and hit Enter.
2. Click **Veeva Vault** Connection and then click **Install**
3. Follow the remaining instructions to complete the installation of the connector.
4. In **Anypoint Studio** click **Help > Installation Details.**
5. Verify **Veeva Vault** connector appears in the list of installed software.

**Note:** The steps in the version of Anypoint Studio on your system might be different to the ones described here.

For information about installing connectors, see [Installing Connectors](https://docs.mulesoft.com/connectors/)

When Studio has an update, a message displays in the lower right corner, which you can click to install the update.

### Configuration:

You can configure the connector to connect to Veeva Vault using **Basic Authentication**

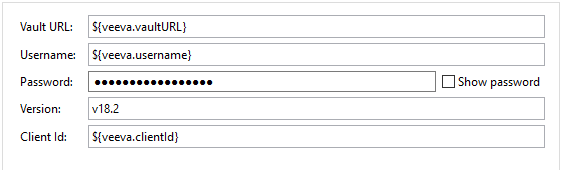
**Basic Authentication**

1. Drag and drop the any of connector operation to the Studio Canvas.
2. Click on icon  of Connector configuration to create global element for the connector configurations.



1. Set these fields:

Username Password Authentication:

****

| Field | Description |
| --- | --- |
| Name | Enter a name for this configuration to reference it later. |
| Vault URL | Enter Veeva Vault instance URL. |
| Username | Enter the Veeva Vault instance username. |
| Password | Enter the corresponding password. |
| Version | Enter the Veeva Vault instance version (e.g: v18.2) |
| Client Id | Enter Client Id in format {company}-{organisation}-{server}-{component} to send with each request in header for tracking purpose. |

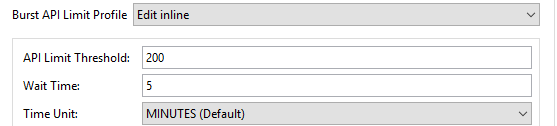
In the images above, the placeholder values refer to a configuration file placed in the folder (src/main/resources) of your project. You can either hardcode your credentials into the global configuration properties or reference a configuration file that contains these values. For simpler maintenance and better re-usability of your project, Mule recommends that you use a configuration file. Keeping these values in a separate file is useful if you need to deploy to different environments, such as production, development, and QA, where your access credentials differ.

The Connection Configuration will do Basic Authentication which will accept required parameters – Vault URL, Username, Password to perform connection with Veeva Vault instance to retrieve Session. The retrieved session will be used for each operation configured in Mule application for sending request.

**Burst API Limit Exceed Profile (Optional):**

The Burst API limit exceed profile configuration **(Connector Configuration > Advanced Tab)** to act upon if Burst API limit of Veeva Vault API is exceeded or reach threshold configure.

By default, this option is disabled. If Burst API Limit Exceed Profile is enabled, then connector will evaluate Veeva APIs response header **(X-VaultAPI-BurstLimitRemaining)** value with API Limit threshold specified value. At Vault API burst limit remaining value exceed or reach this threshold value, the connector will wait for specified Wait time to roll over the Burst Limit Remaining in 5 minutes window.



| Field | Description |
| --- | --- |
| Response Timeout | Enter timeout value for connector to wait for response from Veeva Vault instance. Default is 10 seconds. |
| API Limit Threshold | Enter API threshold limit that Veeva API Burst limit reached or exceed this threshold, connector will wait for specified wait time. Default is 50. |
| Wait Time | Enter Wait time for connector to wait if API Limit threshold is reached or exceeded. Default is 5 mins. |
| Time Unit | Select time unit for wait time. Default is MINUTES. |

**Daily API Limit Exceed Profile (Optional):**

The Daily API limit exceed profile configuration **(Connector Configuration > Advanced Tab)** to act upon if Daily API limit of Veeva Vault API is exceeded or reach threshold configure.

By default, this option is disabled. If Daily API Limit Exceed Profile is enabled, then connector will evaluate Veeva APIs response header **(X-VaultAPI-DailyLimitRemaining)** value with API Limit threshold specified value, the connector will throw an exception indicating the Daily API Limit is exceeded, the exception will continue for each an every jobs/request created until daily limit is reset or roll over next day.



| Field | Description |
| --- | --- |
| API Limit Threshold | Enter API threshold limit that Veeva API Burst limit reached or exceed this threshold, connector will wait for specified wait time. Default is 10000. |

**Response Timeout:**

The maximum time in milliseconds that the connector will wait for the Veeva Vault response. If time elapse will throw Timeout Exception. By default, is 10,000 milliseconds.



| Field | Description |
| --- | --- |
| Response Timeout | Enter timeout value for connector to wait for response from Veeva Vault instance. Default is 30 seconds. |

## Use the Connector in a Mule Flow:

Following are the list of operations with the connector:

* [Create Document Renditions](#_Create_Document_Renditions:)
* [Create Documents](#_Create_Documents)
* [Create Object Records](#_Create_Object_Records)
* [Delete Document Renditions](#_Delete_Document_Renditions)
* [Delete Documents](#_Delete_Documents)
* [Delete Object Records](#_Delete_Object_Records)
* [Download Document](#_Download_Document)
* [Export Documents](#_Export_Documents)
* [Get Audit Details](#_Get_Audit_Details)
* [Get Document Renditions](#_Get_Document_Renditions)
* [Get Documents](#_Get_Documents)
* [Get Object Records](#_Get_Object_Records)
* [Get Picklists](#_Get_Picklists)
* [Query](#_Query)
* [Update Documents](#_Update_Documents)
* [Update Object Records](#_Update_Object_Records)

### Create Document Renditions:

The Create Document Renditions operation allows to add document renditions in bulk. Your vault must be in Migration Mode before using this operation. It takes maximum CSV input data of size 1GB with value in standard UTF-8 encoded.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Metadata Format | Select CSV or JSON option to accept document metadata in specified format. |
| Document Metadata | Document Metadata payload with required metadata either in CSV or JSON format. Make sure payload would be in format as specified in Metadata format field. |
| Input | You can find following list of document metadata required in payload under Input tab of operation. |
| Output | You can find following operation output or response under Output tab. |

### Create Documents

The Create Documents operation allows to create single or multiple document(s) with provided document metadata in CSV or JSON format. Make sure you have uploaded document files on Vault FTP server location for document content to attach with created documents.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Metadata Format | Select CSV or JSON option to accept document metadata in specified format. |
| Document Metadata | Document Metadata payload with required metadata either in CSV or JSON format. Make sure payload would be in format as specified in Metadata format field. |
| Input | You can find following list of document metadata required to create document(s) on vault. Make sure payload should have mandatory metadata as per vault. |
| Output | You can find create documents operation output status or response from vault. |

### Create Object Records

The Create Object Records operation allows to create bulk object records for selected vault object with provided object metadata in CSV or JSON format.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Object Name | Click on icon  to retrieve vault object list. Select the object name from the list to create records for selected object. |
| Metadata Format | Select CSV or JSON option to accept object metadata in specified format. |
| Object Metadata | Object fields payload with required metadata either in CSV or JSON format. Make sure payload would be in format as specified in Metadata format field. |
| Input | You can find following list of object metadata required to create object records. |
| Output | You can find operation output or response retrieved from vault for an object records created. |

### Delete Document Renditions

The Delete Document Renditions operation allows to delete document renditions in bulk using CSV or JSON format input metadata.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**

****

|  |  |
| --- | --- |
| Field | Description |
| Metadata Format | Select CSV or JSON option to accept document metadata in specified format. |
| Document Metadata | Document Metadata payload with required metadata either in CSV or JSON format. |
| Input | You can find following list of document metadata required to create document(s) on vault. Make sure payload should have mandatory metadata as per vault. |
| Output | You can find delete document renditions operation output status or response from vault. |

### Delete Documents

The Delete Documents operation allows to delete document single or multiple document(s) with bulk API using CSV or JSON format input metadata.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**

****

|  |  |
| --- | --- |
| Field | Description |
| Metadata Format | Select CSV or JSON option to accept document metadata in specified format. |
| Document Metadata | Document Metadata payload with required metadata either in CSV or JSON format. |
| Input | You can find following list of document metadata required to delete document(s) on vault. |
| Output | You can find delete documents operation output status or response from vault. |

### Delete Object Records

The Delete Object Records operation allows to delete document single or multiple document(s) with bulk API using CSV or JSON format input metadata.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Object Name | Click on icon  to retrieve vault object list. Select the object name from the list to delete object records for selected object. |
| Metadata Format | Select CSV or JSON option to accept document metadata in specified format. |
| Object Metadata | Object fields payload with required metadata either in CSV or JSON format. Make sure payload would be in format as specified in Metadata format field. |
| Input | You can find following list of object metadata required to delete object records. |
| Output | You can find operation output or response retrieved from vault for an object records deleted. |

### Download Document

The Download Document allows to download latest or specified version of document file from vault and save to specified file system path.

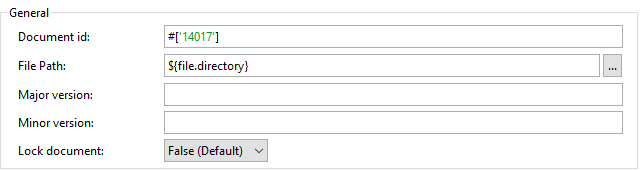
**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Document Id | Provide document Id whose document file need to download. |
| File Path | Provide an absolute directory path at which downloaded file will be save. |
| Major version | Optional, if not provided latest version is download otherwise specified version. |
| Minor version | Optional, if not provided latest version is download otherwise specified version. |
| Lock Document | Set to true to Check Out this document before retrieval. Default is false. |

### Export Documents

An Export Documents operation allows to query a set of documents for export to your vault’s FTP Staging server. It is recommended to use Export Documents operation is separate asynchronous flow using Async Scope and passing payload with document ids. Using asynchronously the operation executes in separate flow thread, will wait for specified polling interval to poll the Job status until it gets successful while the main flow continues its process.

The exported documents are stored on FTP Staging area in folder hierarchy structure – {root}/{userId}/{jobId}/{documentIds}/{versions}. For example, following is job status response:

{

"data": {

"run\_end\_date": "2019-05-22T10:48:57.000Z",

"run\_start\_date": "2019-05-22T10:48:56.000Z",

"method": "POST",

"links": [

{

"method": "GET",

"rel": "self",

"href": "/api/v18.2/services/jobs/34931",

"accept": "application/json"

}

],

"id": 34931,

"created\_date": "2019-05-22T10:48:55.000Z",

"created\_by": 2888511,

"status": "SUCCESS"

},

"responseStatus": "SUCCESS"

}

The above example shows **‘id’** (which is a JobId) and **‘created\_by’** (which is an userId), so folder path on staging server would be like **– {root}/34931/u2888511/{documentIds}/{versions}.**

**root** – Its root location of staging server.

**userId** – It’s a **created\_by** prefix with ‘**u**’. In above case is **u2888511.**

**jobId** – It’s an ‘**id’.** In above case is **34931.**

**documentIds –** The N number of folders with name as documentIds which are passed to export documents operation. The number of folders would be number of documentIds sent to export documents operation.

**versions –** A version folders with name as document versions number and can be multiple version folders associated with document.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Metadata Format | Select CSV or JSON option to accept metadata document Id in specified format. |
| Document Metadata | Document Metadata payload contains document Ids in either in CSV or JSON format. Make sure payload would be in format as specified in Metadata format field. |
| Source | Optional, to exclude source files set to false. Default is true. |
| Renditions | Optional, to include renditions set to true. Default is false. |
| All Versions | Optional, to include all versions or latest version set to true. Default is false. |
| Polling Interval | Optional, to polls vault at specified interval in seconds until job is successful. Default is 10 seconds. |
| Input | You can find following input document metadata required to export document(s) from vault to FTP staging server. Make sure payload should have mandatory metadata as per vault. |
| Output | You can find export documents job status response under output tab. |

### Get Audit Details

The Get Audit Detail operation allows to retrieve audit details for specified audit type.

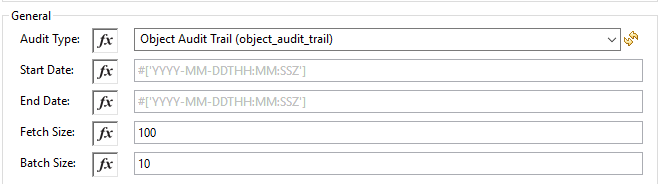
**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Audit Type | Click on icon  to retrieve audit types from vault. Select an audit type from the list whose audit details need to retrieve. |
| Start date | Optional, specify start date to retrieve audit information. This date cannot be more than 30 days ago. Dates must be YYYY-MM-DDTHH:MM:SSZ format. Dates and times are in UTC. If time is not specified, it will default to midnight (T00:00:00Z) on the specified date. |
| End date | Optional, specify end date to retrieve audit information. This date cannot be more than 30 days ago. Dates must be YYYY-MM- DDTHH:MM:SSZ format. Dates and times are in UTC. If time is not specified, it will default to midnight (T00:00:00Z) on the specified date. |
| Fetch Size | Provide number of records per page. Default is 100. |
| Batch Size | Provide number of pages per batch. Default is 10. |
| Input | N/A |
| Output | The following selected audit type details retrieved from vault in pagination format. Make sure to use For each / Splitter to fetch each record. |

### Get Document Renditions

The Get Document Renditions operation allows to retrieve document renditions detail.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Document Metadata | Document Id in payload either in CSV or JSON format. |
| Input | You can see document Id required in payload under Input tab of operation to retrieve document renditions details |
| Output | You can find following operation output or response under Output tab. |

### Get Documents

The Get Documents operation allows to retrieve document details based on document type, subtype and classification selected. Using selected type, subtype and classification it fetches document properties metadata and built VQL query dynamically. The VQL query is then executed on vault to retrieve document details. Place For-Each/Splitter element after this Operation to fetch each document data (page) sequentially.

**Note:** The document type, subtype and classification are optional, if none are selected then document properties metadata are fetched using API (/api/{version}/metadata/objects/documents/properties) and VQL Query is built and executed.

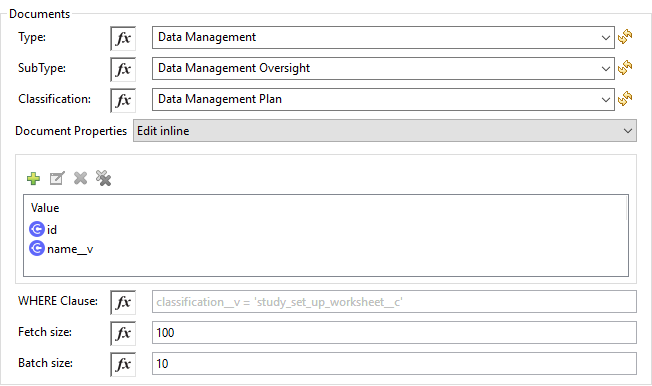
**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Fields | Description |
| Type (Optional) | Click on icon  to retrieve Document Types from vault. Select a document type from the list to retrieve document properties. |
| Subtype (Optional) | Click on icon  to retrieve Document Subtypes from vault. Select a document subtype from the list to retrieve document properties. |
| Classification (Optional) | Click on icon  to retrieve Document Classification from vault. Select a document classification from the list to retrieve document properties. |
| Document Properties (Optional) | Insert document properties as per business requirement and these properties will be used to build a VQL query to execute on vault. |
| WHERE Clause | Insert WHERE clause (without using keyword WHERE) conditions as per business requirement and it will be appended to dynamically to build VQL query to execute on vault. **E.g classification\_\_v = ‘study\_set\_up\_worksheet\_\_c’** |
| Fetch Size | Provide number of records per page. Default is 100. |
| Batch Size | Provide number of pages per batch. Default is 10. |
| Input | N/A |
| Output | You can find list of queryable document properties retrieved from vault. If document properties list is provided, then specified properties detail would retrieve and result in operation output. Default would be list of all queryable properties. |

### Get Object Records

The Get Object Records operation allows to retrieve object records detail using object metadata fields to build a VQL query dynamically and execute on vault to get Object details. Place For-Each/Splitter element after this Operation to fetch each object records (page) sequentially.

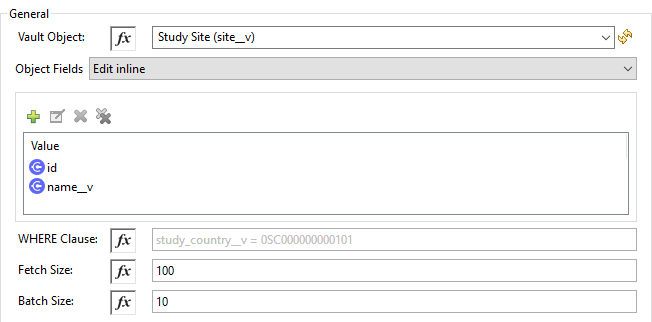
**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Fields | Description |
| Vault Object | Click on icon  to retrieve vault objects list from vault. Select an object from the list to retrieve object records detail. |
| Object Fields (Optional) | Insert object fields as per business requirement and these fields will be used to build a VQL query to execute on vault. |
| WHERE Clause | Insert WHERE clause (without using keyword WHERE) conditions as per business requirement and it will be appended to dynamically to build VQL query to execute on vault. **E.g study\_country\_\_v = ‘0SC000000000101’** |
| Fetch Size | Provide number of records per page. Default is 100. |
| Batch Size | Provide number of pages per batch. Default is 10. |
| Input | N/A |
| Output | You can find the list of object records fields to refer to add into Object fields list. If object fields list is provided, then specified fields detail comes as output. Default would be list of all fields. |

### Get Picklists

The Get Picklists operation allows to retrieve all available picklist values configured on a picklist.

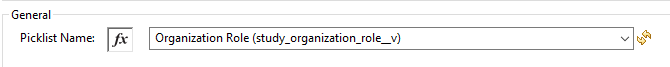
**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**

****

|  |  |
| --- | --- |
| Field | Description |
| Picklist Name (Optional) | Click on icon  to retrieve picklists from vault. Select a picklist name from the list to retrieve picklist values. |
| Input | N/A |
| Output | You can find picklist values with name and label as an output of Get Picklists operation. |

### Query

The Query Operation allows to execute specified VQL query on Vault and retrieve result in paginated data input stream. Place For-Each/Splitter after Query Operation to fetch each record in sequence order. Note - Do not specify LIMIT and OFFSET in VQL query. These parameters are incorporated internally.

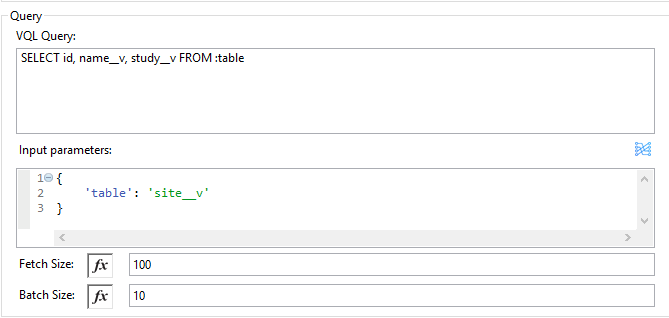
**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| VQL Query | Enter VQL Query to send it to Vault.  **Note -- Do not specify LIMIT and OFFSET in VQL query.** |
| Input Parameters | The input parameters are to pass parameter(s) to VQL query dynamically. The parameters should be name-value pair, payload or variables value can be pass. Eg.  #[{  'table': 'site\_\_v',  }]  Default parameter is **#[{}].** |
| Fetch Size | Provide number of records per page. Default is 100. |
| Batch Size | Provide number of pages per batch. Default is 10. |
| Input | N/A |
| Output | You can find the specified VQL query fields in output tab of operation. It returns specified VQL query result contains pagination data. |

**Use Input Parameters to Protect VQL Queries**

The Query operation is used to retrieve documents, object, workflow information from the Vault. The primary concept of this operation is to supply a VQL query and use DataWeave to supply the parameters:

In the above example, input parameters are supplied as key-value pairs, which you can create by embedding a DataWeave script. Those keys are used in conjunction with the semicolon character **(:)** to reference a parameter value by name. This is the recommended approach for using parameters in your VQL query.

The alternative is to directly write **` <veevavault:vql>SELECT id, name\_\_v, study\_\_v FROM documents WHERE name\_\_v = #[payload] </veevavault:vql>**`, but this is a very dangerous practice that is not recommended.

Advantages of using input parameters to configure the WHERE clause in a SELECT statement this way:

* The query becomes immune to VQL injection attacks.
* The connector can perform optimizations that are not possible otherwise, which improves the app’s overall performance.

### Update Documents

The Update Documents operation allows to update bulk document(s) with provided editable metadata in payload either in CSV or JSON format.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Metadata Format | Select CSV or JSON option to accept document metadata in specified format. |
| Document Metadata | Document Metadata payload with required editable metadata either in CSV or JSON format. Make sure payload would be in format as specified in Metadata format field. |
| Input | You can find following list of editable document metadata to update document(s) on vault. Make sure payload should have only editable metadata as per vault. |
| Output | You can find create documents operation output status or response from vault. |

### Update Object Records

The Update Object Records operation allows to update object records of specified vault object.

**Configurations:**

**Connector Configuration:**



Follow the instructions mentioned in [Connector Configuration](#_Configuration:)

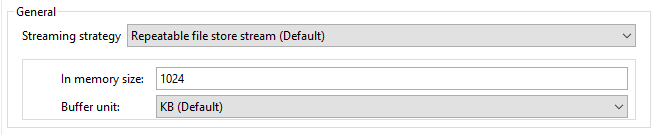
**Operation Configuration:**



|  |  |
| --- | --- |
| Field | Description |
| Object Name | Click on icon  to retrieve vault object list. Select the object name from the list to update object records of vault object. |
| Metadata Format | Select CSV or JSON option to accept object metadata in specified format. |
| Object Metadata | Object fields payload with required metadata either in CSV or JSON format. Make sure payload would be in format as specified in Metadata format field. |
| Input | You can find following list of object metadata required to create object records. |
| Output | You can find object status of an object updated as an output or response retrieved from vault under output tab of operation. |

## Streaming and Pagination

All the operations (except Download Document) in connector returns an InputStream as payload with respective results based on operation output. Because of an InputStream is returned as payload the Mule by default applies Streaming Strategies, please refer to [Mule Streaming Strategies](https://docs.mulesoft.com/mule-runtime/4.2/streaming-about) in more details. You can find streaming strategies configuration in **Advanced** tab of connector operations.



The following operations in connector provides pagination mechanism based on Mule standard pagination.

* [Get Audit Details](#_Get_Audit_Details)
* [Get Documents](#_Get_Documents)
* [Get Object Records](#_Get_Object_Records)
* [Query](#_Query)

While using these above paginated operations make sure to place For-Each/Splitter to retrieve each object (metadata’s in JSON format) at a time. The pagination operations have fields – **Fetch Size and Batch Size.**

* **Fetch Size:** The Fetch Size is a limit number of records that can be retrieve in a page. The operation would return the pages with fetch size number of JSON object records. Note, in some cases, Veeva API’s auto-calculate fetch size (number of records on each page) based on record size accumulated exceed the standard record size. The operation will return calculated records on each page.
* **Batch Size:** The Batch Size is number of pages it would return in each batch and each page will have fetch size number of records. The operation would return number of records (metadata’s in JSON format) per batch is calculated as below **E.**g:

Fetch Size set as **1000**

Batch Size set as **10**

Total records in vault say has **100,000** records then

Number of pages = Total records/Fetch Size

= 100000/1000

= 100 pages.

Number of pages per batch = Number of pages/Batch Size

= 100/10

= 10 pages per batch.

Number of Records per batch = Number of pages per batch \* Fetch Size

= 10 \* 1000

= 10,000 records.

Therefore, Number of records return per batch would be 10,000 records.

The repeatable streams measure the buffer size in byte measurements, when handling objects the runtime measures the buffer size using instance counts.

In non-repeatable stream connector operation would returns stream as number of records per batch and in repeatable stream would returns all records in once, so when calculating the in-memory buffer size for repeatable auto-paging, you need to estimate how much memory space each instance takes to avoid running out of memory.

## Demo Projects

The demo project we provide can be a starting point for your MuleSoft project. The demo project contains sample flows for these operations:

|  |  |
| --- | --- |
| Operations | Description |
| [Create Document Renditions](#_Create_Document_Renditions) | Create Document Renditions operation creates document renditions on vault. Use this option when you want to create document renditions with document metadata received from external source. The document renditions created status is written into file system. However, you can store into any other entities like – Database, Trigger to external source etc. |
| [Create Documents](#_Create_Documents_Examples) | Create Document operation creates document(s) on vault. Use this option when you want to create documents in bulk with document metadata received from external source. The documents created status is inserted into – Database. However, you can trigger to external source etc. |
| [Create Object Records](#_Create_Object_Records_1) | Create Object Records operation creates object records of specified object in vault. Use this option when you want to create object records in bulk using object metadata received from external source. The object records created status is inserted into – Database. However, you can trigger to external source etc. |
| [Update Documents](#_Update_Documents_Example) | Update Document operation update document(s) on vault. Use this option when you want to update documents in bulk with editable document metadata received from external source. The documents updates status is stored into file system. However, you can trigger to external source etc. |
| [Update Object Records](#_Update_Object_Records_1) | Update Object Records operation updates object records of specified object in vault. Use this option when you want to update object records in bulk using object metadata received from external source. The object records updated status is stored into file system. However, you can trigger to external source etc. |
| [Delete Documents](#_Delete_Documents_Example) | Delete Documents operation deletes documents from vault. Use this option when you want to delete documents in bulk using document Id received from external source. The documents deleted status is stored into file system. However, you can trigger/insert to external source. |
| [Get Documents](#_Get_Documents_Example) | Get Documents operations retrieves document details. Use this option when you want to retrieve document details based on type, subtype, classification document properties and VQL query. The document details retrieved in pagination and in order to fetch each record a For-Each element is added which will transform insert records into database. However, you can trigger to external source etc. |
| [Get Object Records](#_Get_Object_Records_1) | Get Object Records operation retrieves object details. Use this option when you want to retrieve object details based on object name. The object details retrieved in pagination and in order to fetch each records a For-Each element is added which will transform and insert records into database. However, you can trigger to external source etc. |
| [Query](#_Query_Example) | Query operation to execute VQL query on vault. Use this option when you want to retrieve document, object and workflow details using VQL query. The VQL query response retrieved in pagination and inorder to fetch each records a For-Each element is added which will transform and store data into file system. However, you can trigger to external sources etc. For more details on VQL please refer Veeva site ( <https://developer.veevavault.com/vql/#introduction-to-vault-queries>) |
| [Get Audit Details](#_Get_Audit_Details_1) | Get Audit Details operation retrieve audit details of specified audit type. Use this option when you want to retrieve audit details of an audit type. The Audit details response retrieved in pagination and in order to fetch each records a For-Each element is added which will transform and store data into file system. However, you can trigger to external sources etc. |

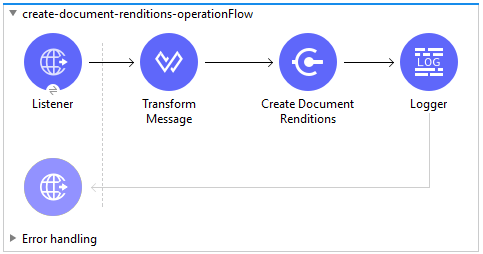
**To use the demo project:**

* You must know about basic concept of Veeva Vault.
* You must know about Veeva VQL query (<https://developer.veevavault.com/vql/#introduction-to-vault-queries>)
* You must know Mule concepts, flow Anypoint Studio and connectors – HTTP, Database, File. Refer to MuleSoft Documentation (<https://docs.mulesoft.com/general/>)
* You must have access to Veeva Vault instance for connector configurations:
* **Username**
* **Password**
* **Vault URL**
* **Vault version**
* **Vault Client Id (This is custom Id in format - {company}-{organization}-{server}-{component})**
* You must know about [Anypoint Studio](https://docs.mulesoft.com/studio/7.3/) on how to create Mule projects.

### Create Document Renditions Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag Transform Message component to transform document renditions metadata in CSV format and configure with mapping.
* Drag Create Document Renditions operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault Url, vault version, vault client Id.
* Select CSV from Metadata format drop down list.
* Drag Logger element to log information in console or log file.



**Example XML Code:**

<flow name=*"create-document-renditions-operationFlow"* doc:id=*"7822e654-0dc5-4a4e-9140-ad72b0a76107"* >

<http:listener doc:name=*"Listener"* doc:id=*"3ecffd1a-296a-4114-ab06-ee8ebcb126ae"* config-ref=*"HTTP\_Listener\_config"* path=*"/create-document-renditions"*/>

<ee:transform doc:name=*"Transform Message"* doc:id=*"4be516bc-20da-4445-8463-a530a7605d0d"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/csv headerLineNumber = 0 , header = true

---

payload]]></ee:set-payload>

</ee:message>

</ee:transform>

<veevavault:create-document-renditions doc:name=*"Create Document Renditions"* doc:id=*"3f35bbe9-0d5b-4dd2-99e9-3094a0da352b"* config-ref=*"Veeva\_Vault\_Config"* metadataFormat=*"CSV"*/>

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"c370c08d-afc4-461e-9b39-d998d48549f2"* message=*"Document Renditions Created :: #[payload]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"d261e967-dcfc-425b-99fb-79210c6ceb24"* type=*"ANY"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"f7669bc0-17f0-484d-9278-f3d7e16a578d"* message=*"Flow Exception Handled -- #[payload]"* />

</on-error-propagate>

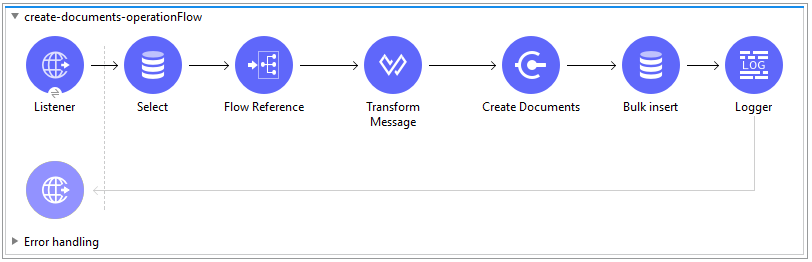
</error-handler>

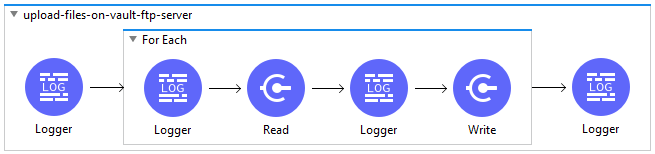
</flow>

### Create Documents Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag Database Select operation to fetch document metadata from table.
* Configure Database connector configuration with credentials details.
* Drag Flow Reference element to reference Sub Flow – upload-files-on-vault-ftp-server.
* Drag Transform Message component to transform document metadata in CSV format.
* Drag Create Documents operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault Url, vault version, vault client Id.
* Drag Database Bulk Insert operation to insert document created status data into table
* Drag Logger element to log information in console or log file.
* Create Sub-Flow with Logger to log information.
* For-Each element containing Read file operation and FTPS Write operation to write file on vault FTP server.
* Configure FTPS connector configuration with vault FTP server credentials details





**Example XML code:**

**create-documents-operationsFlow**

<flow name=*"create-documents-operationFlow"* doc:id=*"98fe428d-3961-41e9-afaf-1cbe2b38dee5"* >

<http:listener doc:name=*"Listener"* doc:id=*"0ea1fb76-ca56-4b78-b1ae-de8962e737b6"* config-ref=*"HTTP\_Listener\_config"* path=*"/create-documents"*/>

<db:select doc:name=*"Select"* doc:id=*"e5af2ee1-faf6-4397-aea8-7df1ca873abb"* config-ref=*"Database\_Config"*>

<db:sql >SELECT \* FROM createdocuments</db:sql>

</db:select>

<flow-ref doc:name=*"Flow Reference"* doc:id=*"3bf8061f-c536-463f-bb98-e0d722289c73"* name=*"upload-files-on-vault-ftp-server"*/>

<ee:transform doc:name=*"Transform Message"* doc:id=*"ac21fdf2-9117-4201-a595-907e4dd1ea12"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/csv headerLineNumber = 0 , header = true

---

payload map ( payload01 , indexOfPayload01 ) -> {

lifecycle\_\_v: payload01.lifecycle\_\_v,

study\_\_v: payload01.study\_\_v,

subtype\_\_v: payload01.subtype\_\_v,

name\_\_v: payload01.name\_\_v,

type\_\_v: payload01.type\_\_v,

classification\_\_v: payload01.classification\_\_v,

file: payload01.file

}]]></ee:set-payload>

</ee:message>

</ee:transform>

<veevavault:create-documents metadataFormat=*"CSV"* doc:name=*"Create Documents"* doc:id=*"5b98678c-1559-4827-80a9-289aad4a8873"* config-ref=*"Veeva\_Vault\_Config"*/>

<db:bulk-insert doc:name=*"Bulk insert"* doc:id=*"e8dd307e-40db-42bb-b82a-b72f6b235d00"* config-ref=*"Database\_Config"*>

<db:bulk-input-parameters ><![CDATA[#[output application/java

---

payload.data map () -> {

'id': $.id as String,

'external\_id\_\_v': $.external\_id\_\_v as String,

'responseStatus': $.responseStatus

}]]]></db:bulk-input-parameters>

<db:sql >INSERT into documents (id, external\_id\_\_v, responseStatus) VALUES (:id, :external\_id\_\_v, :responseStatus)</db:sql>

</db:bulk-insert>

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"a447ff60-8deb-471c-a1be-cc6c01dfcc29"* />

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"19d98427-1566-4f82-b694-f59cc6eb169a"* type=*"ANY"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"9c23d079-fbf2-42e7-afe2-b5186a7b9a3f"* message=*"Flow Exception Handled -- #[payload]"* />

</on-error-propagate>

</error-handler>

</flow>

**upload-files-on-vault-ftp-server (Subflow)**

<sub-flow name=*"upload-files-on-vault-ftp-server"* doc:id=*"b92629e4-69ee-4d9d-a24c-753ff54eff41"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"74d38eea-f86a-4482-a209-06b8fc9dcd8f"* message=*"Starting File Upload on Vault FTP server...."*/>

<foreach doc:name=*"For Each"* doc:id=*"1d1578a5-ea4b-40d6-8b87-b34ec778f232"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"44acecd2-a655-4eee-94cc-934ab7d32b92"* message=*"Uploading file #[payload.file] on Vault FTP server......."* />

<file:read doc:name=*"Read"* doc:id=*"4fad0c52-6d1c-452b-a51d-c5cfd3b2680b"* config-ref=*"File\_Config"* path=*"test-file.txt"*/>

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"76e29521-7854-449e-a7d6-c30a177ef8f9"* message=*"File Content :: #[payload]"*/>

<ftps:write doc:name=*"Write"* doc:id=*"fa077475-e89e-4618-a394-c01c7ac2efbc"* config-ref=*"FTPS\_Config"* path=*"#[attributes.fileName]"*/>

</foreach>

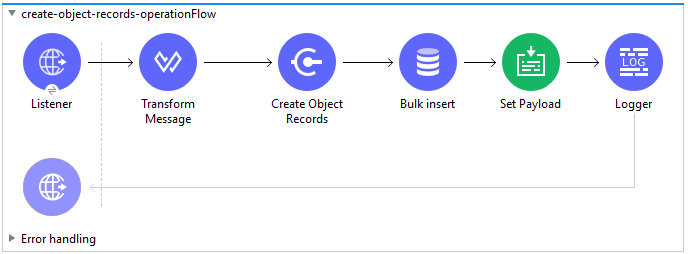
<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"4d8382e4-61b3-4104-ba89-5a3209a2bb9e"* message=*"Files are uploaded successfully on Vault FTP server :: #[payload]"*/>

</sub-flow>

### Create Object Records Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag Transform Message component to transform object metadata in CSV format with mapping.
* Drag Create Object Records operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault Url, vault version, vault client Id.
* Select Object Name from Object Name drop down list (Click refresh button to retrieve object name list).
* Select CSV from Metadata format drop down list.
* Drag Database Bulk Insert operation to insert document created status data into table. Make sure insert query is defined as per your requirement.
* Configure Database connector configuration with credentials details.
* Drag Set Payload element to set payload that will be return to triggering source.
* Logger to log information in console or log file.



**Example XML Code:**

<flow name=*"create-object-records-operationFlow"* doc:id=*"76d86207-916a-4be2-8928-fdd6c717c346"* >

<http:listener doc:name=*"Listener"* doc:id=*"2b5b8ff3-491c-4573-96db-a3b0326f667b"* config-ref=*"HTTP\_Listener\_config"* path=*"/create-object-records"*/>

<ee:transform doc:name=*"Transform Message"* doc:id=*"df08cf9f-3ab4-4f23-97e4-a2726ff78e03"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/csv headerLineNumber = 0 , header = true , separator = ","

---

payload map ( payload01 , indexOfPayload01 ) -> {

name\_\_v: payload01.name\_\_v,

location\_\_v: payload01.location\_\_v,

external\_id\_\_v: (payload01.external\_id\_\_v default "") ++ "|"++ payload01.name\_\_v,

study\_country\_\_v: payload01.study\_country\_\_v,

principal\_investigator\_\_v: payload01.principal\_investigator\_\_v,

actual\_siv\_\_v: payload01.actual\_siv\_\_v as Date,

site\_selected\_date\_\_v: payload01.site\_selected\_date\_\_v as Date

}]]></ee:set-payload>

</ee:message>

</ee:transform>

<veevavault:create-object-records doc:name=*"Create Object Records"* doc:id=*"75b80a1e-0338-4f9e-b9e5-81c944381ddb"* metadataFormat=*"CSV"* objectName=*"site\_\_v"* target=*"objectStatus"* config-ref=*"Veeva\_Vault\_Config"*/>

<db:bulk-insert doc:name=*"Bulk insert"* doc:id=*"14110393-ec5e-4d69-b9e0-3b52b039da35"* config-ref=*"Database\_Config"*>

<db:bulk-input-parameters ><![CDATA[#[output application/java

---

vars.objectStatus.data map () -> {

'id': $.data.id,

'url': $.data.url,

'responseStatus': $.responseStatus

}]]]></db:bulk-input-parameters>

<db:sql >INSERT INTO deletesiteobject (id, url, responseStatus) VALUES (:id, :url, :responseStatus)</db:sql>

</db:bulk-insert>

<set-payload value=*"#[vars.objectStatus]"* doc:name=*"Set Payload"* doc:id=*"e883c1b5-6790-43ba-9d7e-b058793c4a4b"* />

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"b4816f4f-f199-43ee-b00b-c9b389d4343a"* message=*"Flow completed with object records created :: #[vars.objectStatus]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"e681ed06-c458-42ef-bf24-217a42c85866"* type=*"ANY"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"a47171c6-6b6e-47e5-97cd-8a7d0695410d"* message=*"Flow Exception Handled -- #[payload]"* />

</on-error-propagate>

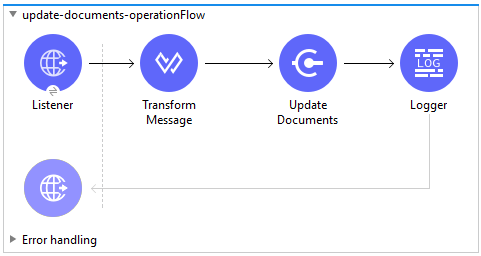
</error-handler>

</flow>

### Update Documents Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag Transform Message component to transform document metadata in CSV format and configure with mapping.
* Drag Update Documents operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault url, vault version, vault client Id.
* Select CSV from Metadata format drop down list.
* Drag Logger element to log information in console or log file.



**Example XML Code:**

<flow name=*"update-documents-operationFlow"* doc:id=*"9745bc16-f9f4-45c4-8ead-0e05a5d1d798"* >

<http:listener doc:name=*"Listener"* doc:id=*"f0bc92a7-8c86-41ce-bcb2-d727290a0216"* config-ref=*"HTTP\_Listener\_config"* path=*"/update-documents"*/>

<ee:transform doc:name=*"Transform Message"* doc:id=*"125f4490-bce5-4379-ac49-6bc680309e20"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/csv headerLineNumber = 0 , header = true , separator = ","

---

payload map ( payload01 , indexOfPayload01 ) -> {

id: payload01.id,

notes\_\_v: (payload01.notes\_\_v default "") ++ payload01.product\_\_v,

name\_\_v: payload01.name\_\_v,

product\_\_v: payload01.product\_\_v

}]]></ee:set-payload>

</ee:message>

</ee:transform>

<veevavault:update-documents metadataFormat=*"CSV"* doc:name=*"Update Documents"* doc:id=*"8a13cdda-6ec8-47a9-9117-44a0584ed4d1"* config-ref=*"Veeva\_Vault\_Config"*/>

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"4bc22f37-365f-4f8a-a4a0-6ecf6f939a45"* message=*"Document Updated :: #[payload]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"4b13626b-2dc7-4bcc-a79f-b1ed523a540c"* type=*"ANY"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"bee5def9-acb2-4d83-96ab-e20c60018182"* message=*"Flow Exception Handled -- #[payload]"* />

</on-error-propagate>

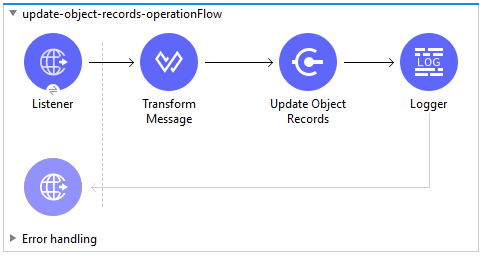
</error-handler>

</flow>

### Update Object Records Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag Transform Message component to transform object metadata in CSV format and configure with mapping.
* Drag Update Object Records operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault url, vault version, vault client Id.
* Select Object Name from Object Name drop down list (Click refresh button to retrieve object name list).
* Select CSV from Metadata format drop down list.
* Drag Logger element to log information in console or log file.



**Example XML Code:**

<flow name=*"update-object-records-operationFlow"* doc:id=*"a51bda88-f358-40ad-9986-8e6aea559e3c"* >

<http:listener doc:name=*"Listener"* doc:id=*"cc2fb1da-1f14-4cef-91af-12ea9a7e1beb"* config-ref=*"HTTP\_Listener\_config"* path=*"/update-object-records"*/>

<ee:transform doc:name=*"Transform Message"* doc:id=*"90ec9543-b70d-485d-b510-802c21d45ff5"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/csv headerLineNumber = 0 , header = true , separator = ","

---

payload map ( payload01 , indexOfPayload01 ) -> {

id: payload01.id,

name\_\_v: payload01.name\_\_v,

location\_\_v: payload01.location\_\_v,

external\_id\_\_v: payload01.external\_id\_\_v,

study\_country\_\_v: payload01.study\_country\_\_v,

principal\_investigator\_\_v: payload01.principal\_investigator\_\_v,

actual\_siv\_\_v: payload01.actual\_siv\_\_v as Date,

site\_selected\_date\_\_v: payload01.site\_selected\_date\_\_v as Date

}]]></ee:set-payload>

</ee:message>

</ee:transform>

<veevavault:update-object-records doc:name=*"Update Object Records"* doc:id=*"5a16fb75-f150-44c9-bf33-3a25878cf197"* config-ref=*"Veeva\_Vault\_Config"* metadataFormat=*"CSV"* objectName=*"site\_\_v"*/>

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"507ee77d-7f6c-429d-81ae-17359bb3afc1"* message=*"Vault Object Updated :: #[payload]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"68925d3d-4e42-4662-84ed-c9ba581e333b"* type=*"ANY"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"d0ea5c85-c456-4ab9-89c7-7da2a51c929b"* message=*"Flow Exception Handled -- #[payload]"* />

</on-error-propagate>

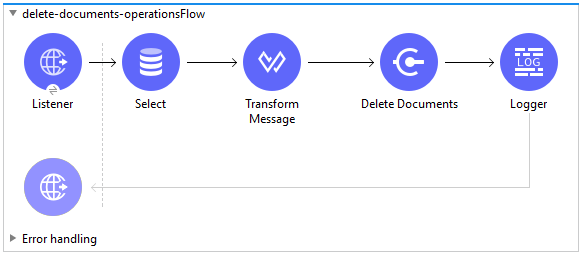
</error-handler>

</flow>

### Delete Documents Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag Database Select operation to fetch document Ids from table.
* Configure Database connector configuration with credentials details.
* Drag Transform Message component to transform document Ids in CSV format.
* Drag Delete Documents operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault Url, vault version, vault client Id.
* Drag Logger element to log information in console or log file.



**Example XML Code:**

<flow name=*"delete-documents-operationsFlow"* doc:id=*"21f3d675-909f-43f9-a778-0a4ffa0691de"* >

<http:listener doc:name=*"Listener"* doc:id=*"4a0c72cc-d442-4c0f-b254-b1eaae48729e"* config-ref=*"HTTP\_Listener\_config"* path=*"/delete-documents"*/>

<db:select doc:name=*"Select"* doc:id=*"060ea275-765b-49c2-b243-c96238cc1a85"* config-ref=*"Database\_Config"*>

<db:sql >select id from deletedocuments</db:sql>

</db:select>

<ee:transform doc:name=*"Transform Message"* doc:id=*"8d1e28bb-2f6a-4835-9a68-bb5abf7e3ffc"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/csv headerLineNumber = 0 , header = true , separator = ","

---

payload map ( payload01 , indexOfPayload01 ) -> {

id: payload01.id as String

}]]></ee:set-payload>

</ee:message>

</ee:transform>

<veevavault:delete-documents metadataFormat=*"CSV"* doc:name=*"Delete Documents"* doc:id=*"4dde94f1-c0d5-4ea2-8f64-fc95642b823d"* config-ref=*"Veeva\_Vault\_Config"*/>

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"66db562e-80c1-4db6-993e-addb3f013613"* message=*"Document Deleted :: #[payload]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"4dc356f1-5618-429e-9fc2-8a6eaee96798"* type=*"ANY"* >

<set-payload value=*"#[payload]"* doc:name=*"Set Payload"* doc:id=*"c9e6b85c-bb64-4de4-827c-02449a9f7566"* />

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"c7ed420f-2acc-438a-85f7-01a6505cbb0d"* message=*"Flow Exception Handled -- #[payload]"* />

</on-error-propagate>

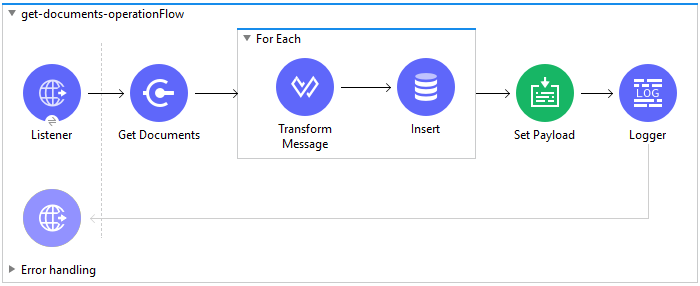
</error-handler>

</flow>

### Get Documents Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag and drop Get Documents operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault Url, vault version, vault client Id.
* Click respective refresh button to fetch document type, subtype, classification list.
* Select Type, Subtype, Classification from drop down list each.
* Add document properties in document properties list as per business requirement to incorporate properties name in VQL query build dynamically.
* Drag For-Each element after Get Documents operation.
* Drop Transform Message and Database – Insert operation from respective palette.
* Configure transformation using Dataweave as per your business need.
* Configure Insert query to insert data into table.
* Drag Set Payload element after For-Each to set payload that can be returned to triggered source.
* Drag Logger element to log information in console or log file.



**Example XML code:**

<flow name=*"get-documents-operationFlow"* doc:id=*"facc29d2-8025-4b58-a3ec-322ded221705"* >

<http:listener doc:name=*"Listener"* doc:id=*"93298153-7da4-4fcc-b028-00e692b0d19e"* config-ref=*"HTTP\_Listener\_config"* path=*"/get-documents"*/>

<veevavault:get-documents doc:name=*"Get Documents"* doc:id=*"c9e9f482-91ce-40d2-a7f7-92b39fc7e8a3"* config-ref=*"Veeva\_Vault\_Config"*>

<veevavault:documents type=*"site\_management\_\_c"*>

<veevavault:document-properties >

<veevavault:document-property value=*"name\_\_v"* />

<veevavault:document-property value=*"type\_\_v"* />

<veevavault:document-property value=*"subtype\_\_v"* />

<veevavault:document-property value=*"classification\_\_v"* />

<veevavault:document-property value=*"lifecycle\_\_v"* />

<veevavault:document-property value=*"status\_\_v"* />

<veevavault:document-property value=*"study\_\_v"* />

<veevavault:document-property value=*"product\_\_v"* />

</veevavault:document-properties>

</veevavault:documents>

</veevavault:get-documents>

<foreach doc:name=*"For Each"* doc:id=*"a05bd532-48eb-42a8-af56-7c9830a4433d"* >

<ee:transform doc:name=*"Transform Message"* doc:id=*"d5bf0da9-e544-4728-a3e9-57dc05a02fa9"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/json

---

payload]]></ee:set-payload>

</ee:message>

</ee:transform>

<db:insert doc:name=*"Insert"* doc:id=*"bf5a1288-8f37-456d-b26c-d18b8ea411db"* config-ref=*"Database\_Config"*>

<db:sql >INSERT INTO createdocuments (name\_\_v, type\_\_v, subtype\_\_v, classification\_\_v, lifecycle\_\_v,

product\_\_v, status\_\_v, study\_\_v, file) VALUES (:name\_\_v, :type\_\_v, :subtype\_\_v, :classification\_\_v, :lifecycle\_\_v, :product\_\_v, :status\_\_v, :study\_\_v, :file)</db:sql>

<db:input-parameters ><![CDATA[#[%dw 2.0

output application/json

---

{

'name\_\_v': payload.name\_\_v as String,

'type\_\_v': payload.type\_\_v as String,

'subtype\_\_v': payload.subtype\_\_v as String,

'classification\_\_v': payload.classification\_\_v as String,

'lifecycle\_\_v': payload.lifecycle\_\_v as String,

'product\_\_v': payload.product\_\_v[0],

'status\_\_v': payload.status\_\_v as String,

'study\_\_v': payload.study\_\_v[0],

'file': "CDA.docx"

}]]]></db:input-parameters>

</db:insert>

</foreach>

<set-payload value=*"#[output application/json --- payload.payload]"* doc:name=*"Set Payload"* doc:id=*"abcb3160-55a5-49f8-9b31-d7e61dca3d2e"* />

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"ab49834e-389e-483e-84a4-2b61752bba72"* message=*"Document Details :: #[payload]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"Error Propagate"* doc:id=*"a7ac2cf2-ec74-408b-8a02-ee1a0d24e8f4"* type=*"ANY"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"2b932042-57f3-45d8-9e6b-3cd9fcda6509"* message=*"Flow Exception Handled -- #[payload]"* />

</on-error-propagate>

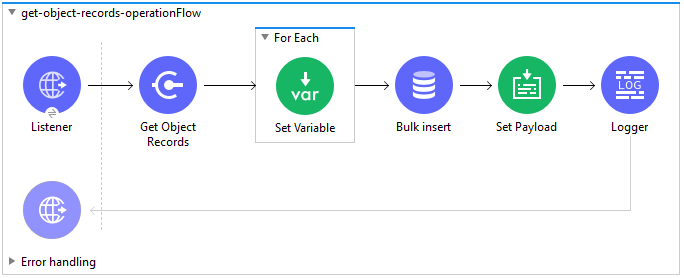
</error-handler>

</flow>

### Get Object Records Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag and drop Get Object Records operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault Url, vault version, vault client Id.
* Click refresh button to correspond to Object Name to retrieve object name list.
* Select Object Name from Object Name drop down list.
* Add Object fields in object fields list as per business requirement to incorporate fields name in VQL query build dynamically.
* Drag For-Each element after Get Documents operation.
* Drag Set Variable element to insert each object records into array variable
* Drag Database Bulk Insert operation to insert object records data into table
* Drag Set Payload element to set payload that can be returned to triggered source.
* Drag Logger element to log information in console or log file.



**Example XML Code:**

<flow name=*"get-object-records-operationFlow"* doc:id=*"53818dc8-f0fd-4ccd-967f-e83dadcc1525"* >

<http:listener doc:name=*"Listener"* doc:id=*"190a5ad6-d379-4ff8-a8c4-b2b9cddbed2b"* config-ref=*"HTTP\_Listener\_config"* path=*"/get-object-records"*/>

<veevavault:get-object-records doc:name=*"Get Object Records"* doc:id=*"0ee4a657-c261-43c4-acce-00577bebb6c1"* config-ref=*"Veeva\_Vault\_Config"* vaultObject=*"site\_\_v"*>

<veevavault:object-fields >

<veevavault:object-field value=*"name\_\_v"* />

<veevavault:object-field value=*"location\_\_v"* />

<veevavault:object-field value=*"study\_country\_\_v"* />

<veevavault:object-field value=*"external\_id\_\_v"* />

<veevavault:object-field value=*"actual\_siv\_\_v"* />

<veevavault:object-field value=*"site\_selected\_date\_\_v"* />

<veevavault:object-field value=*"principal\_investigator\_\_v"* />

</veevavault:object-fields>

</veevavault:get-object-records>

<foreach doc:name=*"For Each"* doc:id=*"ab811c9e-2595-487e-a9ab-05aa5e4dc1d8"* >

<set-variable value=*"#[output application/java --- vars.records default[] ++ [payload]]"* doc:name=*"Set Variable"* doc:id=*"060d7249-9251-4d5c-9b5d-1a63102cf6ae"* variableName=*"records"*/>

</foreach>

<db:bulk-insert doc:name=*"Bulk insert"* doc:id=*"3a3f03cd-afc2-48e8-9049-38b3058457f4"* config-ref=*"Database\_Config"*>

<db:bulk-input-parameters><![CDATA[#[%dw 2.0

output application/java

---

vars.records map {

'name\_\_v': $.name\_\_v,

'location\_\_v': $.location\_\_v,

'study\_country\_\_v': $.study\_country\_\_v,

'external\_id\_\_v': $.external\_id\_\_v,

'actual\_siv\_\_v': $.actual\_siv\_\_v,

'site\_selected\_date\_\_v': $.site\_selected\_date\_\_v,

'principal\_investigator\_\_v': $.principal\_investigator\_\_v

}]]]></db:bulk-input-parameters>

<db:sql>INSERT INTO siteobjectrecords (name\_\_v, location\_\_v, study\_country\_\_v, external\_id\_\_v, actual\_siv\_\_v,

site\_selected\_date\_\_v, principal\_investigator\_\_v) VALUES (:name\_\_v, :location\_\_v, :study\_country\_\_v, :external\_id\_\_v,

:actual\_siv\_\_v, :site\_selected\_date\_\_v, :principal\_investigator\_\_v)</db:sql>

</db:bulk-insert>

<set-payload value=*"#[output application/json --- vars.records]"* doc:name=*"Set Payload"* doc:id=*"4bb1eb6d-f703-4871-8b5e-0a46a771a59e"* />

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"b9e95b86-ac51-4846-aa06-e8fcbeee4a46"* message=*"Vault Object retrieved :: #[payload]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"54095f77-47cc-4768-ad37-113a2d1481ec"* type=*"ANY"* >

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"b14cc42b-9c2a-4e36-bbb3-30c834e8e3f0"* message=*"Flow Exception Handled -- #[payload]"* />

</on-error-propagate>

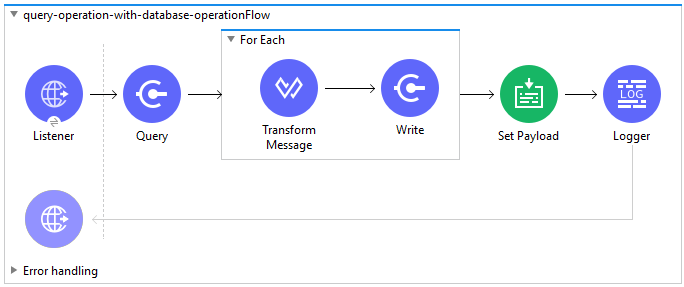
</error-handler>

</flow>

### Query Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag Query operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault Url, vault version, vault client Id.
* Define VQL query as per business requirement.
* Add input placeholder if used in VQL query to incorporate dynamically with VQL at runtime.
* Drag For-Each element after Query operation.
* Drop Transform Message and File - Write operation from respective palette.
* Configure transformation using Dataweave as per your business need for transforming payload.
* Configure File connector configuration with working file path and Write operation to write payload data into specified file.
* Drag Set Payload element after For-Each to set payload that can be returned to triggered source.
* Drag Logger element to log information in console or log file.



**Example XML Code:**

<flow name=*"query-operation-with-database-operationFlow"* doc:id=*"451f0f78-1e2e-4cc4-b56d-4f427f01e30a"* >

<http:listener doc:name=*"Listener"* doc:id=*"454e6602-2c2f-4e98-8569-b3e928bf3da5"* config-ref=*"HTTP\_Listener\_config"* path=*"/query"*/>

<veevavault:query doc:name=*"Query"* doc:id=*"e4310f17-b0f5-49c0-8662-bc2d798fd9de"* config-ref=*"Veeva\_Vault\_Config"*>

<veevavault:vql >SELECT id, name\_\_v, type\_\_v, subtype\_\_v, classification\_\_v, lifecycle\_\_v, status\_\_v, study\_\_v, product\_\_v from :table</veevavault:vql>

<veevavault:input-parameters ><![CDATA[#[{

'table': 'documents'

}]]]></veevavault:input-parameters>

</veevavault:query>

<foreach doc:name=*"For Each"* doc:id=*"55a3f3e9-b0ad-4c6d-8ccf-96f2d11f7bc8"* >

<ee:transform doc:name=*"Transform Message"* doc:id=*"f4875262-fe32-42e6-8e21-a21e592a729d"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/json

---

payload]]></ee:set-payload>

</ee:message>

</ee:transform>

<file:write doc:name=*"Write"* doc:id=*"cb88f995-b31f-4dcf-b508-ea8340ae84fb"* path=*"query.json"* config-ref=*"File\_Config"* mode=*"APPEND"*>

<file:content><![CDATA[#[output application/json

---

payload]]]></file:content>

</file:write>

</foreach>

<set-payload value=*"#['Flow Completed']"* doc:name=*"Set Payload"* doc:id=*"66f96a83-dfc9-4855-a65c-c35327cfec2a"* />

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"016dc7a0-7014-44d9-bcbd-d2f540c2fc2e"* message=*"Query Operation flow Completed - #[payload]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"6a7998b6-4384-41fb-bb5a-625f5410003a"* type=*"ANY"*>

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"46ee422c-4430-4b63-b1c3-6c993ad732fa"* message=*"Flow Exception Handled -- #[payload]"*/>

</on-error-propagate>

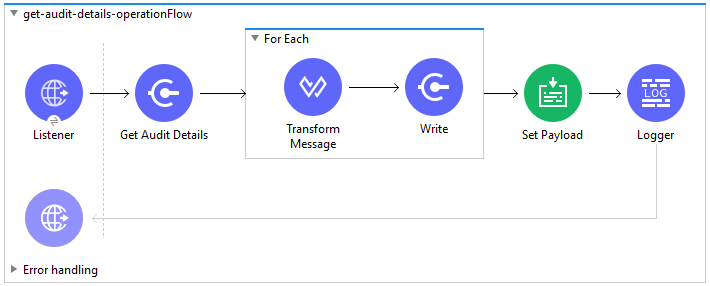
</error-handler>

</flow>

### Get Audit Details Example

To create Mule flow, follow these instructions:

* Drag and drop HTTP Listener from Mule palette to project canvas.
* Configure HTTP Listener as per your environment.
* Drag Get Audit Details operation from Veeva Vault Connector palette.
* Configure Veeva Vault connection configuration with username, password, vault Url, vault version, vault client Id.
* Click refresh button to fetch audit types list from vault.
* Select Audit Type from drop down list.
* Drag For-Each element after Get Audit Details operation.
* Drop Transform Message and File - Write operation from respective palette.
* Configure transformation using Dataweave as per your business need for transforming payload.
* Configure File connector configuration with working file path and Write operation to write payload data into specified file.
* Drag Set Payload element after For-Each to set payload that can be returned to triggered source.
* Drag Logger element to log information in console or log file.



**Example XML Code:**

<flow name=*"get-audit-details-operationFlow"* doc:id=*"33cb5d09-419b-435e-8744-e93a318a7ccb"* >

<http:listener doc:name=*"Listener"* doc:id=*"3880815e-a45e-45dc-bd93-163adea84db6"* config-ref=*"HTTP\_Listener\_config"* path=*"/get-audit-details"*/>

<veevavault:get-audit-details doc:name=*"Get Audit Details"* doc:id=*"e05e19d4-1422-43f4-a14e-f4c94660a838"* config-ref=*"Veeva\_Vault\_Config"* auditType=*"object\_audit\_trail"*/>

<foreach doc:name=*"For Each"* doc:id=*"40573cba-b238-44f9-bf71-9a1fa7b23197"* >

<ee:transform doc:name=*"Transform Message"* doc:id=*"6a17ec35-1ec0-4b83-9bbd-1dbc59feacb5"* >

<ee:message >

<ee:set-payload ><![CDATA[%dw 2.0

output application/json

---

payload]]></ee:set-payload>

</ee:message>

</ee:transform>

<file:write doc:name=*"Write"* doc:id=*"09e842a6-6cee-42ee-8fce-4aeb3eb4898e"* path=*"get-audit-details.json"* config-ref=*"File\_Config"* mode=*"APPEND"*>

<file:content ><![CDATA[#[%dw 2.0

output application/json

---

payload]]]></file:content>

</file:write>

</foreach>

<set-payload value=*"#[output application/json --- payload.payload]"* doc:name=*"Set Payload"* doc:id=*"a48d5e43-1377-4d3e-82ba-984b18c9cde5"* />

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"51f56870-73ee-4b41-9d3e-078e64bd370a"* message=*"Audit Details :: #[payload]"*/>

<error-handler >

<on-error-propagate enableNotifications=*"true"* logException=*"true"* doc:name=*"On Error Propagate"* doc:id=*"1933ca25-531a-4e43-baac-fc145ad79c93"* type=*"ANY"*>

<logger level=*"INFO"* doc:name=*"Logger"* doc:id=*"e02a58e6-2028-4c59-a649-e112443a3431"* message=*"Flow Exception Captured #[payload]"*/>

</on-error-propagate>

</error-handler>

</flow>

## Connector Namespace and Schema

If you are creating Veeva Vault connector XML by hand, add the namespace for the connector. By contrast, Anypoint Studio creates the namespace for a connector automatically when you create a project for a Mule app.

xsi:schemaLocation="

...

http://www.mulesoft.org/schema/mule/veevavault

http://www.mulesoft.org/schema/mule/veevavault/current/mule-veevavault.xsd"

Example:

<mule xmlns="http://www.mulesoft.org/schema/mule/core"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:veevavault="http://www.mulesoft.org/schema/mule/veevavault"

xsi:schemaLocation="

http://www.mulesoft.org/schema/mule/core

http://www.mulesoft.org/schema/mule/core/current/mule.xsd

http://www.mulesoft.org/schema/mule/veevavault

http://www.mulesoft.org/schema/mule/veevavault/current/mule-veevavault.xsd">

<!-- Insert your configuration elements and your flow here -->

</mule>

## Maven Dependencies Information

When creating an app manually from the XML, you need to set up the pom.xml for your project:

<build>

<plugins>

<plugin>

<groupId>org.mule.tools.maven</groupId>

<artifactId>mule-maven-plugin</artifactId>

<version>${mule.maven.plugin.version}</version>

<extensions>true</extensions>

<configuration>

<sharedLibraries> <sharedLibrary>

<groupId>org.json</groupId>

<artifactId>json</artifactId>

</sharedLibrary>

</sharedLibraries>

</configuration>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>anypoint-exchange-v2</id>

<name>Anypoint Exchange</name>

<url>https://maven.anypoint.mulesoft.com/api/v2/maven</url>

<layout>default</layout>

</repository>

<repository>

<id>mulesoft-releases</id>

<name>MuleSoft Releases Repository</name>

<url>https://repository.mulesoft.org/releases/</url>

<layout>default</layout>

</repository>

</repositories>

This snippet specifies a project dependency for a specific release version:

<dependency>

<groupId>org.mule.extension</groupId>

<artifactId>mule-veevavault-connector</artifactId>

<version>RELEASE</version>

<classifier>mule-plugin</classifier>

</dependency>