Copying Vault Data Using Direct Data API and Migration Mode

Executive Summary

As Veeva Vault data grows in size and complexity, moving complete or partial datasets between Vaults is critical for advanced use cases, such as executing migration projects, performing extended testing on high-risk data-dependent features, or establishing a production-like baseline in a Sandbox environment.

This white paper outlines how administrators can leverage **Direct Data API** and **Migration Mode** to copy full or partial datasets between Vaults.

> **Note:** This method is **not** intended to replace *Sandbox* or *Snapshot* refresh operations or standard production-to-production migrations. **Data Snapshots** remain the recommended best practice for setting up *Sandbox Vaults* for repeatable testing. Only use the method described in this white paper when *Snapshots* or *Test Data Packages* cannot support the required data volume or complexity.

Limitation: *This method is not applicable to Clinical Data Vaults, which utilize Study File Format rather than Direct Data API.*

Overview

This process leverages **Direct Data API** for high-speed data extraction and uses **Vault Loader** with **Migration Mode** for efficient data insertion.

The core process steps include:

1. **Object Record Create Pass:** Creates all object records in the target Vault *without* relationships.
2. **Object Record Update Pass:** Updates the records to establish relationships via object reference fields
3. **Document Create Pass**: Creates documents and document versions *without* relationships.
4. **Document Relationship Pass**: Updates the records and creates the relationship between documents.

Best Practices

We recommend that you follow the below best practices when adopting this process or when creating an automation to ensure a consistent approach with minimal errors and risks.

Controlled Data Cloning

Setting up new environments by cloning data without restriction is **not recommended**, especially indiscriminate cloning of *Production Data*. This process works best when creating a new *Configuration Clone* of a Vault.

Configuration Mode

It is also recommended to turn on **Configuration Mode** while you are importing the data to the target Vault to avoid potential issues caused by user activity.

Transformation, Obfuscation, and Filtering

Sandboxes can have fundamentally different users—often with fewer or no restrictions. This allows Sandbox users to access data that would otherwise not be visible, creating serious risks to data integrity and confidentiality.

Therefore, while **Transformation**, **Obfuscation**, and **Filtering (Hierarchical and Value Based)** may appear as *optional* steps from a *technical perspective*, we recommend they be treated as *mandatory* to ensure *Production Data* is not exposed or misused.

When defining automation requirements, include capabilities needed for **Transformation**, **Obfuscation**, and **Filtering (Hierarchical and Value Based)** of the source data in the Direct Data API extract in order to be loaded into the target environment.

Filtering (Hierarchical and Value-Based)

Filtering (hierarchical and value-based) is important for managing the **size** of the data that is copied to another environment. Just because the process enables you to copy all the data, it does not mean that you should or is the best approach.

Environments with too much data will prevent you from being agile with your sandboxes and snapshots. For example, you cannot use a Snapshot larger than the license of the available Vaults. If you load a sandbox with data beyond the limits, it will prevent you from creating new records in the Sandbox until you downsize the data below the limit.

Keeping the Sandboxes and Snapshots within data limits ensures that you can easily refresh the environments and perform configuration and data changes without having to go through a downsizing process first.

If you believe you need "all the data" for validation reasons, we recommend the following white paper: Leveraging Snapshots for Compliant Validation in Veeva Vault

Automating Data Transformation and Load

While this guide explains the step-by-step logic, executing these transformations *manually* carries a high risk of error for large datasets and requires significant effort. We strongly recommend executing the transformation and loading steps using a script or program (e.g., Python, Java) to ensure data integrity and efficiency.

Target Vault Setup Recommendation

For best results, we recommend starting the process by setting up a Target Vault as a fresh Configuration Copy of the Source Vault where the data will be extracted from.

Why? While the process works for "out-of-sync" copies as well, this method assumes that the *Configuration Data* between the two Vaults are the same and have the same Record IDs. You can only ensure this by using a fresh Configuration Copy.

Prerequisites

Before initiating the copy process, ensure the following requirements are met in both Source and Target Vaults for the user trying to perform the action:

- **Permissions**
  - **Vault Loader** access
  - **Record Migration** permission
  - **API Access** to be able to execute API calls
- **Document Field Security**: You will need to Configure Field-Level Security for *Document Fields* to be editable for the user performing the copy

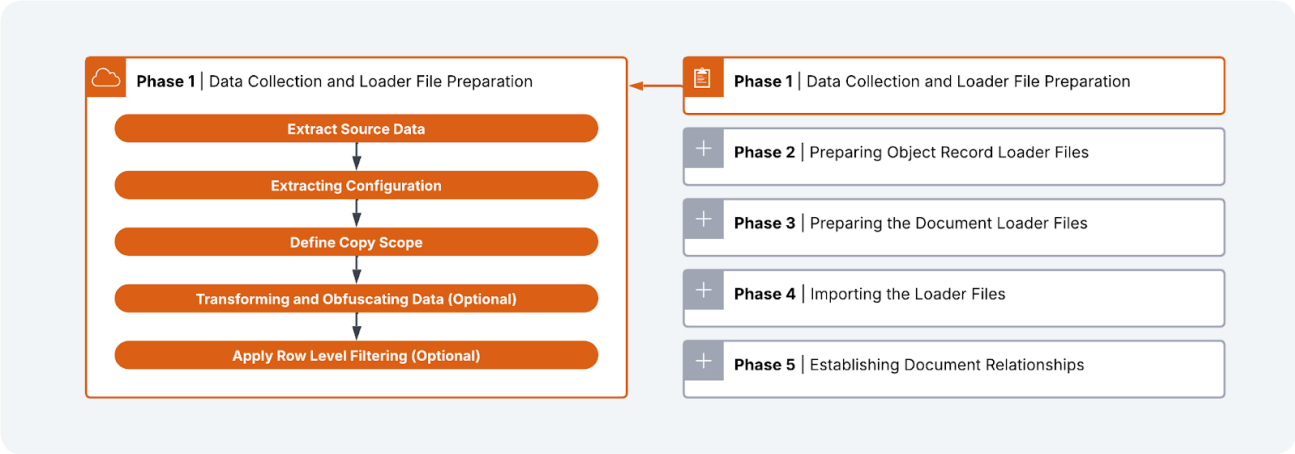
You need to make the major_version_number__v (Major Version Number) field **Editable** for the user performing the action, to be able to create *Documents* and *Document Versions* with a specific version number.

  - You need to make any other fields **Editable** that you wish to set at the time of import that may be *Read-only* or *Hidden* to the user by default.
  - Some system fields like stage__sys (Lifecycle Stage) cannot be set to Editable, and as such will be set by the system at import.

- **Direct Data API**
  - Must be enabled in the **Source Vault**
  - Navigate to **Admin > Settings > General Settings**
  - Select **Enable Direct Data API**
    Note: After enabling the feature, it requires time for the building of the initial file, it is not instantaneous.
- **File Staging Access**
  - Access to the Vault File Staging (via Loader CLI, API, or Vault Toolbox) for uploading large CSV files.
  - Create the following folder in the File Staging of the Target Vault:
    - OBJECT_RECORD_CREATE_PASS

- OBJECT_RECORD_UPDATE_PASS
- DOCUMENT_PASS
- DOCUMENT_RELATIONSHIP_PASS
- OBJECT_RECORD_DOCUMENT_FIELD_PASS

Phase 1: Data Collection and Loader File Preparation



Phase 1.1: Extract Source Data

The **Direct Data API** generates a daily *Full* file containing the complete dataset of the Vault.

1. **Download** the daily *Full* extract from the Source Vault.
   Note: You can use Vault Toolbox (Chrome or Edge) or the dedicated API endpoints to download the files.
2. **Unzip** the contents to a local dedicated working directory.
3. **Organize** the working directory with subdirectories for each processing stage:
   - Object Record Create Pass
   - Object Record Update Pass
   - Document Create Pass
   - Object Record Document Field Pass
   - Document Relationship Pass

Phase 1.2: Extracting Configuration

Use the Configuration Report or information retrieved from the *Vault API* to determine:

- Which fields are *Object Fields or Document Fields*
- Which objects are *Configuration Data* or use *Auto-Naming*
- If fields are *Editable*.

These tools can also be used to programmatically automate the process.

When preparing the files for data import, you will need to evaluate the configuration to determine if a specific column can be or should be imported. Without performing this evaluation, you will run into import errors when using Vault Loader to create Object Records, Documents, and Document Versions.

Phase 1.3: Define Copy Scope

Analyze the Target Vault configuration metadata to determine which objects can be **excluded**. You do not need to import every object in the extract.

- **System-Managed Objects:** Exclude transactional system objects (e.g., workflow envelopes) as these are generated automatically during Vault operations.
- **Configuration Data:** Objects that define configuration are automatically included if the target is a *Configuration Clone*. However, keep in mind that configuration records may reference non-configuration data and, as such, will require an update.
- **Exclusion List:** Identify business-specific objects that are irrelevant to the target environment's purpose. You can also exclude documents if they are not required for your setup.

Phase 1.4: Transforming and Obfuscating Data

If a column should not be completely excluded, you should replace sensitive values with dummy data — where necessary — to comply with data privacy and security standards.

If implementing this process automatically via a script, consider what is the best way to define the needed transformation rules and impacted fields based on your use case and included objects.

Phase 1.5: Apply Row Level Filtering

You may wish to exclude *Object Records* and *Documents* from the copy process for various reasons:

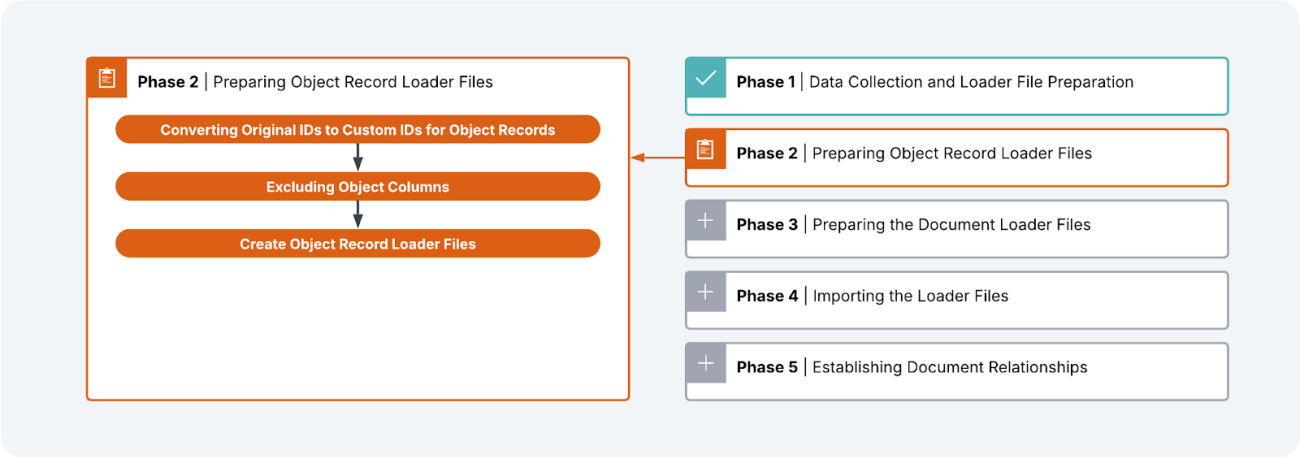
- Reduce size by only copying select hierarchies
- Remove irrelevant or confidential information
- Remove records with in-flight workflows

This kind of *Row Level Filtering* will need to be done consistently across all files. It is best to automate this process, as depending on the size of the data, it could be an unnecessarily time-consuming task.

Hierarchical Filtering
One way to implement **hierarchical filtering** is to *recursively* iterate through the *inbound relationships* of an object and do the same for every object identified in the *inbound relationship hierarchy*. You can then use the obtained **hierarchy map** to copy only a set of "top level" records and their children, while all other records are "discarded" from the process.

Any objects **not** in the **hierarchy map** are treated as *reference data* and all records of that object are copied over (except for the objects already excluded).

Phase 2: Preparing Object Record Loader Files



The Direct Data Extract will contain a separate CSV File for each Object in Vault. These files need to be processed and transformed to work with Record Migration Mode.

Phase 2.1: Converting Original IDs to Custom IDs for Object Records

To successfully link records in a target Vault, you must control the Record IDs. Vault Loader's Record Migration Mode allows you to define *Custom IDs* during creation.

To prevent conflicts with future Vault-generated IDs, Veeva requires a specific format for *Custom IDs*: **the fourth character of the ID must be 'Z'.**

- **Transformation Logic:** Transform the **id** column and **all object reference columns** in every CSV file. Example: V580000000001001 → V58Z00000001001
- **Exception - Configuration Data:** Do *not* transform IDs for records that represent *Configuration Data*, or Object and Document fields referencing *Configuration Data,* as these IDs must match the records already present in the Target Vault, provided the Target Vault is a fresh Configuration Copy.
- **Exception - Users:** User IDs are consistent across Vaults in the same domain. If you are moving data between Vaults of different domains, exclude user fields or map them to a valid user in the target.

Converting IDs from Source Vaults that have Custom IDs

It is possible that the Source Vault will already have records created with Custom IDs that would match the transformed ID of another record.

For example, there could be a *V580000000001001* and *V58Z00000001001* both present in the source data. This would lead to ID duplication on import and incorrect linking from other records referencing the IDs.

To avoid this issue, create a mapping table with **Source Record ID** and **Transformed Record ID** to check for duplicates and resolve conflicts. You can use the mapping table to perform the replacement of the **Source Record ID** on other CSV files which reference the Object.

Custom Record IDs are only prescriptive about the first four characters of a Record ID (Object Prefix + "Z"). You can replace more characters if needed: *V580000000001001 → V58ZZ0000001001.*

Phase 2.2: Excluding Object Columns

Record Migration Mode is a powerful feature. However, there may be cases where you need to exclude columns, and these columns either cannot be set with *Record Migration Mode,* or it does not make sense to set them. In this case, these columns should be excluded from the Loading process.

There are many ways you can exclude columns in the loading process. One way is to provide a prefix or suffix to the column name. The Loader will simply ignore these columns when importing the data (e.g., EXCLUDE_global_id__sys). This is required so that you do not run into errors throughout the import process.

Fields to be excluded:

- **Formula Fields** must be excluded as they perform calculations, and do not store data.
- global_id__sys
- state_stage_id__sys
- stage__sys

Contextually Non-Editable field to be excluded:

- status__v for objects that use Lifecycles
- lifecycle__v for object that use Lifecycles

Conditionally excluded fields:

- **Lookup Fields** can be excluded as they will be set by the system
- **Roll-up Fields** can be excluded as they can be re-initialized after the import
- name__v should be excluded for objects that use *Sequence-Based Auto-Naming* (i.e., auto-numbering) **unless** the starting number in the target Vault is updated for the object to be higher than the largest number being imported.
- created_by__v should be excluded if copying data from another domain (e.g., PROD -> SBX)
- modified_by__v should be excluded if copying data from another domain (e.g., PROD -> SBX)

Please note: The fields are mentioned from a platform perspective. Each Vault application may have additional fields that cannot or should not be edited even in migration mode.

Phase 2.3: Create Object Record Loader Files

Object Record Create Pass File Preparation:

1. **Header Modification:** Vault Loader ignores CSV headers that do not match object field names. We can use this to temporarily exclude relationship fields.

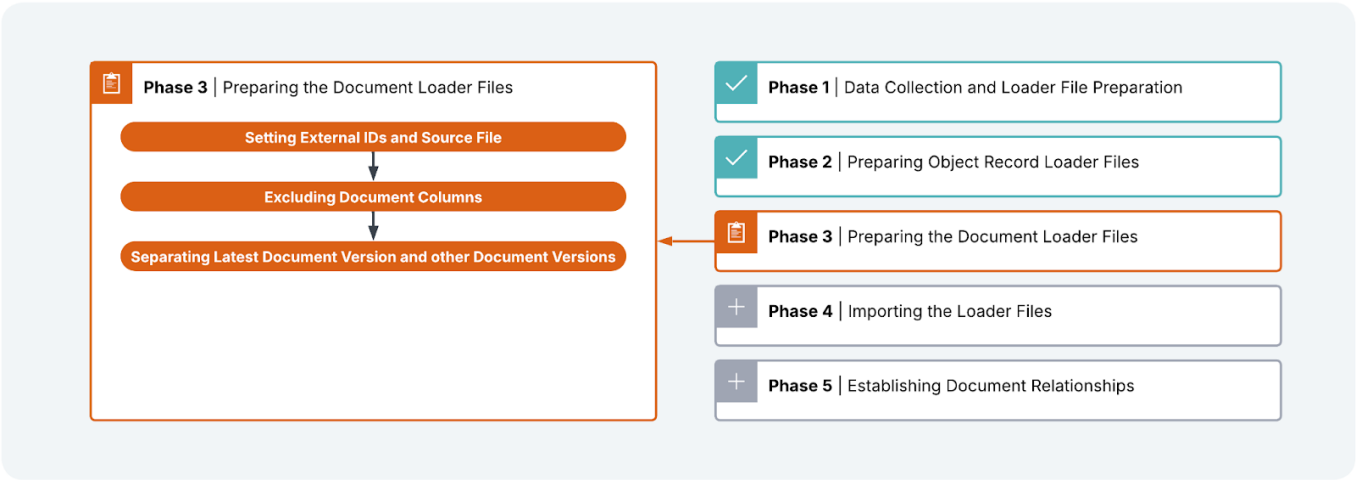
a. Temporarily prefix object reference columns to be used in the Object Record Update Pass (e.g.,: UPDATEPASS_object_field__c)

b. Temporarily prefix document reference columns to be used in the Object-to-Document Linking (e.g.,: OBJECTDOCFIELDPASS_document_field__c)

Object Record Update Pass File Preparation:

1. **Copy Loader CSVs**: Create a copy of the files used in the Object Record Create Pass.
2. **Column Cleanup:** Remove all columns *except* the id and the *Object Reference Columns*.
3. **Header Restoration:** Remove the temporary prefix from the *Object Reference Columns* so Vault Loader recognizes them.
4. **Configuration Data References:** Ensure you include files for *Configuration Data* objects if they reference *Non-Configuration Data* objects.

Phase 3: Preparing the Document Loader Files



The Direct Data Extract will contain a single CSV file for all *Document Versions* in Vault. These files need to be processed and transformed to work with Document Migration Mode.

Phase 3.1: Setting External IDs and Source File

Migrating documents is more complex because Document Migration Mode enforces stricter constraints than *Record Migration Mode*. Specifically, **you cannot define a custom ID (id) for documents.**

Use the **document_version__sys.csv** file from the *Direct Data Extract*.

1. **External ID Mapping:** Copy the source doc_id into the external_id__v column. This is crucial for linking versions together and for mapping relationships later.

2. **Source Files:** Add a file column pointing to the source file path in File Staging.
    - *Recommendation:* For sandboxes, upload a single dummy file (e.g., placeholder.pdf) to File Staging and use this path for all rows to simplify processing.

Phase 3.2: Excluding Document Columns

To work according to the restrictions of Document Migration Mode the following steps need to be performed for the Document Loader Files:

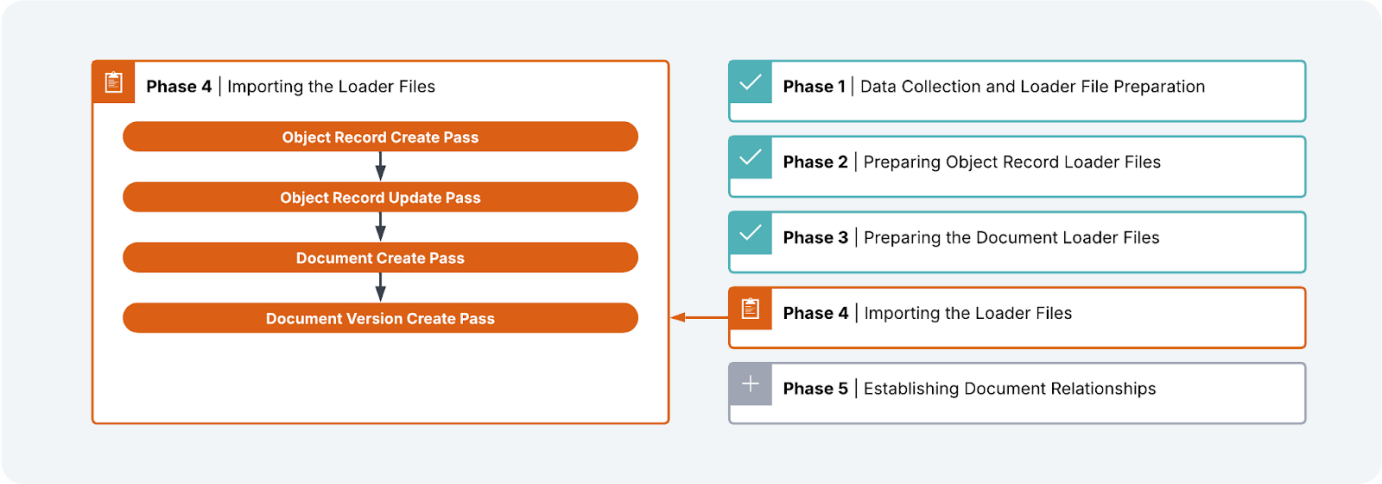
- **Required Fields**: You will not be able to create *Documents* and *Document Versions* if there are missing required fields. For this reason, any missing required fields will need to be populated for all *Documents* and *Document Versions*.
- **Non-Editable Fields**: Fields that are *Default Non-Editable* and have *No Editable Overrides*, must be excluded. There are many ways you can exclude fields. One way is to provide a prefix or suffix to the column name. The Loader will simply ignore these columns when importing the data (e.g., EXCLUDE_global_id__sys). This is required so that you do not run into errors throughout the import process.

Phase 3.3: Separating Latest Document Version and Other Document Versions

To import *Documents* effectively, you first need to import the *Latest Document Version* and then load in all the *Non-Latest Document Versions* later.

- Create Latest Version Loader File
    - Filter the CSV for rows where latest_version__v = true
    - Save as **document_create_latest_versions.csv** in the *"Document Create Pass" working directory*
- Create Non-Latest Version Loader File
    - Filter the CSV for rows where latest_version__v = true
    - Save as **document_create_non_latest_versions.csv** in the *"Document Create Pass" working directory*

Phase 4: Importing the Loader Files



Phase 4.1: Object Record Create Pass

The goal of this phase is to create records in the Target Vault *without* any relationships (i.e., Object and Document fields) because during this step the related records may not exist yet.

UI Execution:

1. **CSV file**: Choose the prepared CSV file from the *"Object Record Create Pass"* working directory
2. **Entity Type**: Select the Object for which the CSV file contains the Object Records
3. **Action Type**: Upsert
4. **Key Field**: ID (id)
5. **Record Migration Mode**: True
6. **No Triggers**: True

API Execution:

1. Upload the prepared CSV files from the *"Object Record Create Pass" working directory* to File Staging to the OBJECT_RECORD_CREATE_PASS folder.
2. Create **Vault Loader** jobs for each object with the following Body Parameter:

```
[{
  "entity_type": "vobjects__v",
  "object": "object_name__v",
  "action": "upsert",
  "file": "OBJECT_RECORD_CREATE_PASS/file_name.csv",
  "idparam": "id",
  "recordmigrationmode": true,
  "notriggers": true
}]
```

Phase 4.2: Object Record Update Pass

The goal of this pass is to establish relationships between the records created in the previous step.

UI Execution:

1. **CSV file**: Choose the prepared CSV file from the *"Object Record Update Pass"* working directory.
2. **Entity Type**: Select the Object for which the CSV file contains the Object Records
3. **Action Type**: Upsert
4. **Key Field**: ID (id)
5. **Record Migration Mode**: True
6. **No Triggers**: True

API Execution:

1. Upload the updated CSV files from the *"Object Record Update Pass" working directory* to File Staging to the OBJECT_RECORD_UPDATE_PASS folder.
2. Create **Vault Loader** jobs for each object with the following Body Parameter:

```
[{
  "entity_type": "vobjects__v",
  "object": "object_name__v",
  "action": "upsert",
  "file": "OBJECT_RECORD_UPDATE_PASS/file_name.csv",
  "idparam": "id",
  "recordmigrationmode": true,
  "notriggers": true
}]
```

Phase 4.3: Document Create Pass

Documents must be loaded in a specific order to ensure version lineage is preserved. This means starting with creating the **Latest Version** of a *Document Version Tree* first.

UI Execution:

1. **CSV file**: Choose the **document_create_latest_versions.csv** CSV file from the *"Document Create Pass"* working directory
2. **Entity Type**: Documents
3. **Action Type**: Create
4. **Document Migration Mode**: True

API Execution:

1. Upload the **document_create_latest_versions.csv** to File Staging to the DOCUMENT_PASS folder
2. Create a **Vault Loader** job with the following Body Parameter:

```
[{
  "entity_type": "documents__v",
  "action": "create",
  "file": "DOCUMENT_PASS/document_create_latest_versions.csv",
  "documentmigrationmode": true
}]
```

Phase 4.4: Document Version Create Pass

Once the **Latest Version** of a *Document Version Tree* is created, we can proceed to load the rest of the *Non-Latest Document Versions*.

UI Execution:

1. **CSV file**: Choose the **document_create_non_latest_versions.csv** CSV file from the *"Document Create Pass"* working directory

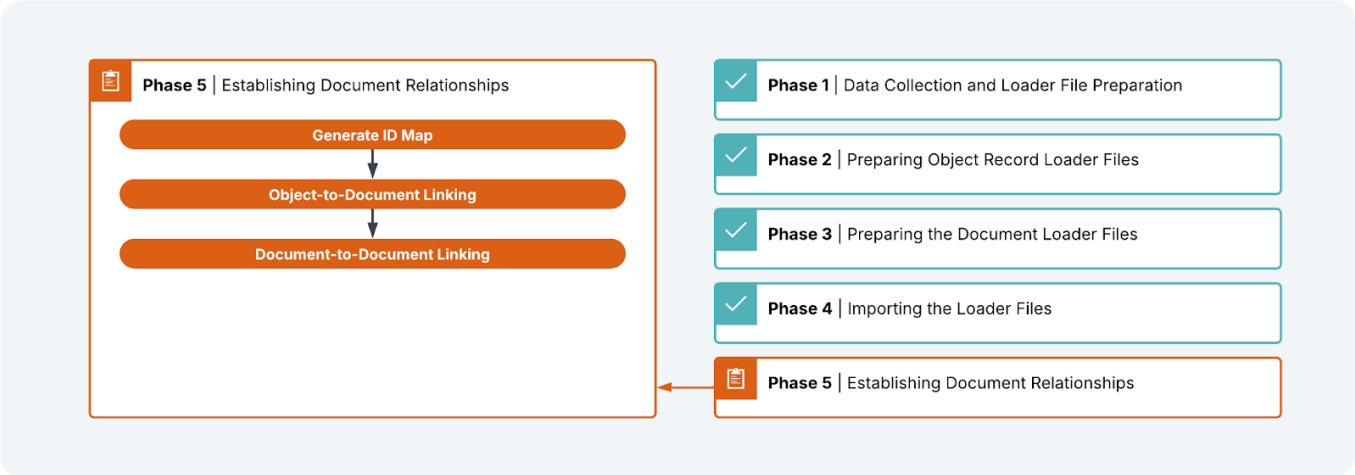
2. **Entity Type**: Document Versions
3. **Action Type**: Create
4. **Document Migration Mode**: True

API Execution:

1. Upload the **document_create_non_latest_versions.csv** to File Staging to the DOCUMENT_PASS folder
2. Create a **Vault Loader** job with the following Body Parameter:

```
[{
  "entity_type": "document_versions__v",
  "action": "create",
  "file": "DOCUMENT_PASS/document_create_non_latest_versions.csv",
  "documentmigrationmode": true
}]
```

Phase 5: Establishing Document Relationships



Since the documents in the Target Vault were generated with new system IDs (not the Custom "Z" IDs), you must remap any references to these documents.

Phase 5.1: Generate ID Map

To set the accurate references, you must map the *Target Vault Document IDs* to the *Source Vault Document IDs*. Achieve this by extracting the id and external_id__v columns from the *Target Vault* using a Loader, VQL query, or a Report.

When we created the Documents and Document Versions, we used the *Source Vault Document IDs* as the external_id__v column. This is why these two columns are the IDs map themselves.

Phase 5.2: Object-to-Document Linking

To establish the link between Object Records and Documents, you need to update the Object Records that have a *Document Field* by replacing the *Source Vault Document IDs* with *Target Vault Document IDs.*

The CSV files in the *"Object Record Create Pass" working directory* will have the *Document Field* columns prefixed with a specific prefix marking them for this process (e.g., OBJECTDOCFIELDPASS_).

Object Record Document Field File Preparation

1. **Locate Files**: Open each CSV file in the *"Object Record Create Pass"* working directory that has a *Document Field*.
2. **Remove Columns**: Delete all columns except the id and any column with the dedicated prefix.
3. **Remap Values:** Replace the *Source Vault Document IDs* in these columns with the new *Target Vault Document IDs* using the ID map generated in the previous step.

    a. For "Version Specific" references (e.g., 123_1_0), replace only the ID portion before the first underscore.

1. **Restore Headers:** Remove the dedicated prefix.
2. **Execute Load:** Run a Vault Loader Upsert job with **Record Migration Mode** enabled. You can use the same method as described for the *Object Record Update Pass*.

Phase 5.3: Document-to-Document Linking

The Direct Data Extract also contains the Document Relationships. If you wish to establish these relationships you can use the ID map generated in the previous step.

1. **Locate Files**: Open the **document_relationships.csv** file from the Direct Data API extract.
2. **Remap Values:** Use the ID map to replace values in source_doc_id__v and target_doc_id__v columns.

    a. For "Version Specific" references (e.g., 123_1_0), replace only the ID portion before the first underscore.

3. Save as **document_relationships_create.csv** in the *"Document Relationship Pass"* working directory.

Phase 5.3.1: Document Relationship Pass

Once you create all of the *Documents* and *Document Versions* it is now possible to set up the relationships between them.

UI Execution:

1. **CSV file**: Choose the **document_relationships_create.csv** CSV file from the *"Document Relationship Pass"* working directory
2. **Entity Type**: Document Relationships
3. **Action Type**: Create

API Execution:

1. Upload the **document_relationships_create.csv** to File Staging to the DOCUMENT_RELATIONSHIP_PASS folder
2. Create a **Vault Loader** job with the following Body Parameter:

```
[{
  "entity_type": "document_relationships__v",
  "action": "create",
  "file": "DOCUMENT_RELATIONSHIP_PASS/document_relationships_create.csv",
}]
```

Conclusion

Administrators can use this approach to successfully clone complex, interrelated datasets between Vaults. This method ensures that even with Vault's strict validation rules, users can achieve high-fidelity data copies for critical testing and migration activities.

Veeva Systems
Global Headquarters
Pleasanton, California, USA
4280 Hacienda Drive
Pleasanton, California 94588