Aprimo DAM Configuration Moving Tooling

~ Technical Guide ~

Contents

[Overview 3](#_Toc192848348)

[Use Cases & Capabilities 3](#_Toc192848349)

[Automated Migration Support 3](#_Toc192848350)

[Features of the tooling 4](#_Toc192848351)

[Primary Use Cases 4](#_Toc192848352)

[Technical Architecture & REST API Usage 4](#_Toc192848353)

[Key REST API Endpoints & Object Details 5](#_Toc192848354)

[Execution Order 7](#_Toc192848355)

[Authentication & Connection Setup 7](#_Toc192848356)

[Exporting Configuration Data 7](#_Toc192848357)

[Importing Data to Target Environment 8](#_Toc192848358)

[Limitations & Manual Interventions 9](#_Toc192848359)

[API Constraints 9](#_Toc192848360)

[Execution Constraints 9](#_Toc192848361)

[Enhancements & Future Considerations 9](#_Toc192848362)

[Summary & Next Steps 10](#_Toc192848363)

# Overview

The Aprimo DAM Configuration Moving Tooling is designed to streamline the process of migrating Digital Asset Management (DAM) configurations between environments. This set of tools automates the bulk transfer of key configuration elements while ensuring dependency resolution, security integrity, and validation processes.

This tooling primarily relies on **REST API-based automation** to extract, modify, and import configurations. Where APIs are unavailable, manual intervention is required to ensure a complete deployment.

# Use Cases & Capabilities

## Automated Migration Support

The tool automates moving only certain configuration objects within DAM:

* **Configuration object migration**:
  + Classifications,
  + Field Groups,
  + Field Definitions,
  + Settings,
  + Rules,
  + File Types,
  + Content Types
* **Security & permissions transfer**:
  + User Group creation with Domain Rights assigned,
  + Classification Permissions,
  + Functional DAM permissions

Configuration that cannot be automatically moved via this tooling will need to be addressed manually.

## Features of the tooling

* **Matching based on object names**: This tooling does not use object GUIDs/IDs, as these IDs are going to be different across Aprimo environments. Instead, it uses object names for matching the object to update.
* **Storage of configuration**: Tools export into data files (XML or Excel).
* **Logging & validation**: Detailed audit logs are provided during runtime to ensure traceability.
* **Delta deployments**: It is possible to do selective configuration move based on modified elements.
* **Error handling & rollback mechanisms**: As the tooling saves configuration in data files, it is possible to backup previous configuration and roll back to it, if deployment goes wrong.

## Primary Use Cases

* **Deployment after activation**: Moving all configurations from **sandbox to production**.
* **Incremental updates**: Deploying small batches of configuration changes in sprint cycles.
* **Security structure replication**: Duplicating user groups and permissions across environments.
* **Testing & validation**: Running exports and analyzing configurations before final deployment.
* **Disaster recovery scenarios**: Rebuilding configurations after data loss or migration issues.

## Technical Architecture & REST API Usage

The tool is built in **C# (.NET Framework)** and utilizes Aprimo’s **REST API endpoints** for migration operations. The core workflow involves:

1. **Extracting configuration data** from the source environment.
   * Transforming identifier into object names before saving into data files
2. **Importing configurations** into the target environment.
   * Transforming and validating the data before import.

## Key REST API Endpoints & Object Details

|  |  |  |  |
| --- | --- | --- | --- |
| Configuration Type | REST API Endpoint | Method | Dependencies |
| Field Groups | /api/core/fieldgroups | GET, PUT, POST | Standalone, no dependencies |
| Field Definitions | /api/core/fields | GET, PUT, POST | Requires Classifications and Field Groups to be created first |
| Classifications | /api/core/classifications | GET, PUT, POST | Requires parent classifications if hierarchical; may require Field Definitions and/or Field Groups for registration |
| Content Types | /api/core/contenttypes | GET, PUT, POST | Requires Field Definitions |
| File Types | /api/core/filetypes | GET, PUT, POST | Standalone, no dependencies |
| Rules | /api/core/rules | GET, PUT, POST | May depend on field definitions |
| Setting Categories | /api/core/settingcategories | GET, PUT, POST | Standalone, no dependencies |
| Setting Definitions | /api/core/settingdefinitions | GET, PUT, POST | Requires Setting Categories |
| Setting Values | /api/core/setting/{settingname} | GET, PUT | Requires Setting  Definitions |
| User Groups | /api/groups | GET, PUT, POST | Standalone, but permissions depend on them |
| Classification Permissions | /api/core/classification/  {classificationId}/{permissionType} | GET, PUT | Requires Classifications and User Groups |
| Functional Permissions | /api/core/permissions  /api/core/usergroup/{groupId}/permissions | GET, PUT | Requires User Groups |

**Example API Call – Exporting Fields**

GET https://{aprimodomain}/api/core/fields

Authorization: Bearer {access\_token}

**Sample Response**

[

{

"id": "12345",

"name": "Asset Title",

"type": "String",

"required": true,

"defaultValue": "Untitled Asset"

}

]

**Example API Call – Importing a New Field**

POST https://{aprimodomain}/api/core/fields

Authorization: Bearer {access\_token}

Content-Type: application/json

{

"name": "New Field Name",

"type": "String",

"required": false,

"defaultValue": "N/A"

}

**Sample Response**

{

"id": "67890",

"name": "New Field Name",

"status": "Created"

}

# Execution Order

## Authentication & Connection Setup

* Requires **Client ID and Access Token**, as well as **Registration** and **Username**
* Authentication uses **OAuth 2.0** for secure API access.

## Exporting Configuration Data

* The user selects objects to export.
* The tool calls the **relevant GET endpoints**.
* Exported data is saved in structured XML files (for non-security related items) or Excel file (for security related items).

**Validating & Cleaning Data**

During the export, dependent data is cleaned up so that the references are based on object names instead of object IDs that are environment specific.

# Importing Data to Target Environment

It is recommended to review exported files before running an import. If there are areas to clean up or adjust you can do so before importing.

The import process must follow a strict order to prevent dependency errors:

1. **Field Groups** (ensures fields have an assigned group)
2. **Classifications** (must exist before fields are imported)
3. **Field Definitions** (classifications and field groups should exists before field definitions are imported)
4. **Classifications** (import of classifications should be repeated for classification objects that have registrations on them, so that newly created field definitions can be registered)
5. **Content Types** (dependent on field definitions as well)
6. **File Types** (no dependencies)
7. **Rules** (should be imported after fields and classifications are in place; this tooling does not support copying watermarks, if rules are reliant on watermarks, those need to be created first manually)
8. **Setting Categories** (no dependencies)
9. **Setting Definitions** (custom Setting Categories must be imported, before importing related Setting Definitions)
10. **Setting Values** (for custom settings, Setting Definition and if applicable Setting Categories need to be imported prior to assigning setting values)
11. **User Groups** (imported before any other security related data to allow role-based assignments; it is recommend to only use User Group import for new groups that need to be created, and not for updating existing groups)
12. **Classification Permissions** (to import Classification Permissions, you need to have Classifications and User Groups imported first)
13. **Functional Permissions** (since Functional permissions are extended in DAM based on other objects being added to the system, you should run this import as last – dependencies: File Types, Content Types, Field Groups, User Groups)

This sequence ensures that all dependencies are resolved before proceeding with the next set of configurations.

It is also recommended to restart the tool before importing different types of objects, especially the ones that are dependent on previously imported ones.

For more information around tooling usage, please refer to [Documentation](https://github.com/Aprimo-Connect/DAM-Configuration-Moving-Tools/tree/main/Documentation)

# Limitations & Manual Interventions

## API Constraints

This tooling is constrained by currently available API routes, therefore portions of configuration not mentioned in this document will have to be moved/deployed manually. Examples:

* **Feature Flags**: Aprimo Service Now requests need to be submitted for Feature Flag enablement
* **Content Workspace UI Configurations**: Cannot be exported/imported (Spaces, Content Type layouts, menu option definitions and placement etc).
* **Languages and Translations**: Must be manually adjusted post-migration.

## Execution Constraints

* **Order of Import Matters**: Must process dependencies first.
* **Environment-Specific Adjustments**: URLs and environment variables may require modifications.
* **API Rate Limits**: Bulk migrations may require throttling to avoid hitting API limits.

# Enhancements & Future Considerations

* **Extending API support** for currently unsupported elements.
* **Automating UI Configurations** via potential scripting workarounds.
* **Enhancing validation mechanisms** for pre-deployment analysis.
* **Error recovery & rollback improvements** for failed deployments.
* **Batch execution optimizations** to reduce API overhead.

# Summary & Next Steps

The Aprimo DAM Configuration Moving Tooling significantly reduces **manual effort and deployment risks**, ensuring efficient environment synchronization. While **API limitations necessitate some manual steps**, ongoing improvements and custom enhancements can further streamline processes.