UPPER SPIRE | the summit of Salesforce cloud apps

CARTULARIUS

DOCUMENT MANAGEMENT

Getting Started with Cartularius for Developers

Getting Started with Cartularius for Developers

Cartularius Version 3.48

Copyright © 2020-2025 Upper Spire B.V. All rights reserved.

Disclaimer

Upper Spire B.V. (Upper Spire) makes no representation or warranty concerning the adequacy or accuracy of this documentation or the software which it describes. In no event will Upper Spire or its direct or indirect suppliers be liable for any damages whatsoever including, but not limited to, direct, indirect, incidental, or consequential damages of any character including, without limitation, loss of business profits, data, business information, or any other commercial damages or losses, or for any damages.

Upper Spire B.V.

Website: www.upperspire.com
E-mail: support@upperspire.com

Table of Contents

Chapter 1: Introduction *1*
What is Cartularius? 1
Scope of this document 2
Chapter 2: Auto Folder Creation *3*
What is Auto Folder Creation? 3
SObject Configuration 3
Add Custom Objects to CDM SObjects (Global Value Set) 4
Create Triggers for Custom Objects 5
Chapter 3: Invocable Actions *6*
Overview and Use Cases 6
Common User Cases 6
Key Advantages 6
Available Invocable Actions 7
CDM Copy File 7
CDM Create Content Document 7
CDM Link Content Document 8
CDM Create Folder 8
CDM Delete Items 9
CDM Get File Categories for Folder 9
CDM Get Folder Id 10
CDM Get Folder Link 10
CDM Get Folder Resource Path 11
CDM Import Content Document 11
CDM Is Create Folder Allowed 12
CDM Is File Category Restricted 12
CDM Is Folder Locked by File Category Restriction 13
CDM Move File 13
CDM Purge Items 14
CDM Set Current Version 14
CDM Get Folder Key (deprecated) 14
Using Invocable Actions in Salesforce Flow 15
Adding an Invocable Action to a Flow 15
Common Automation Scenarios 16
Example: Automatically Create a Folder When a Record Is Created 16
Example: Move a File to a “Closed” Folder When a Record Is Updated 16
Tips and Best Practices 16

Chapter 4: External Integration *17*

Salesforce Files **17**

Salesforce File Limits 17

Connecting External Software to Salesforce 18

Uploading Files to Salesforce 20

Downloading Files from Salesforce 21

Amazon S3 ..... **23**

Connecting External Software to Amazon S3 23

Uploading Files to Amazon S3 25

Downloading Files from Amazon S3 28

Data Control and Security 28

FEEDBACK *29*

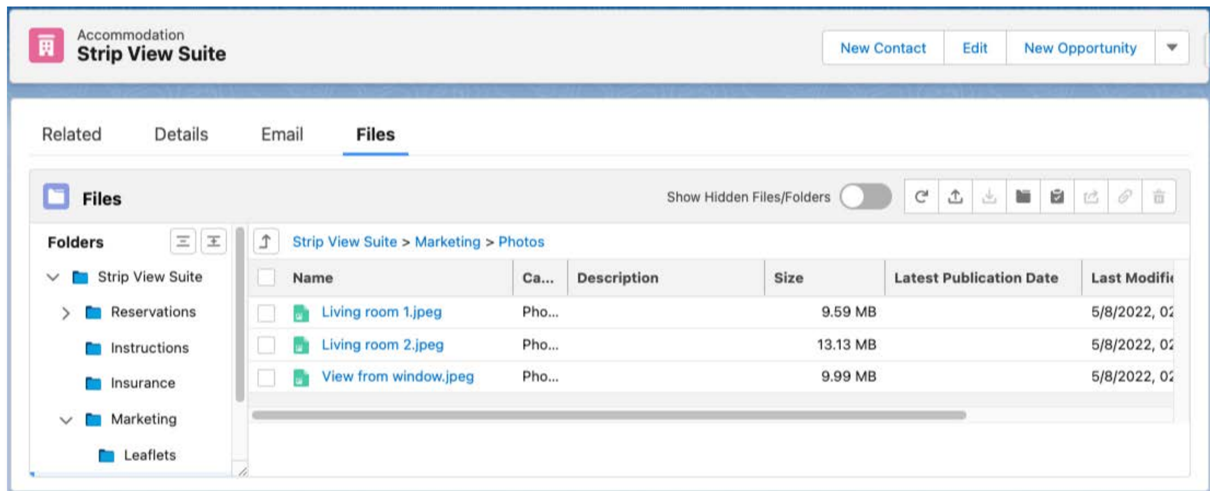
MORE RESOURCES *29*

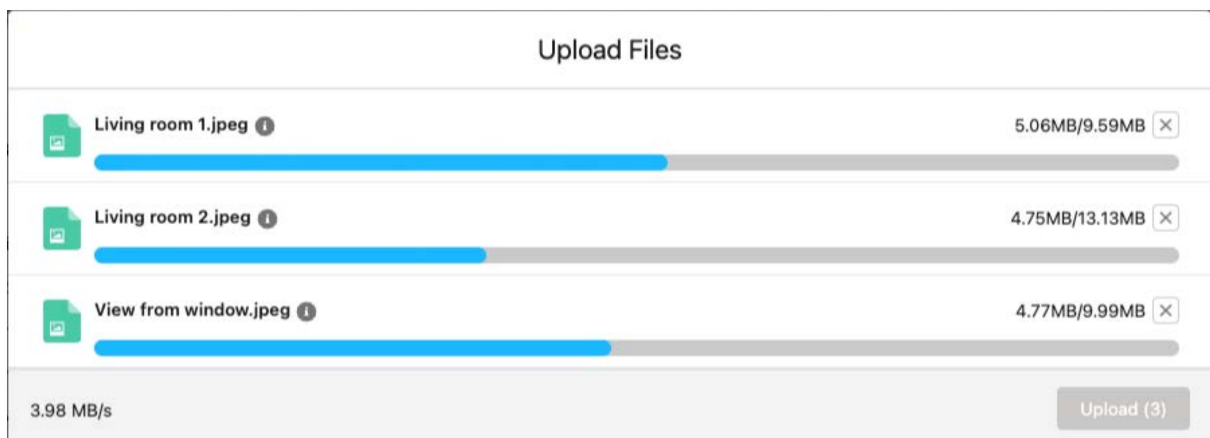
Chapter 1: Introduction

Welcome to the *Getting Started with Cartularius for Developers* guide. This guide is designed to provide documentation for a technical audience, specifically developers responsible for performing development tasks within the Salesforce ecosystem.

What is Cartularius?

Cartularius (CDM) is an app that can be installed from the Salesforce AppExchange as an add-on to your existing Salesforce organization. CDM is a document management solution that relates your Salesforce object model to an advanced folder hierarchy. The software also serves as an interface between Salesforce and Amazon S3, which securely stores your files. CDM is a cost-effective solution that enables you to store gigabytes of files and access them directly from your Salesforce records.





Scope of this document

This guide is intended for developers who are getting started with Cartularius Document Management (CDM). It focuses specifically on development tasks such as automating file handling, integrating with external systems, and using CDM's invocable actions and APIs.

While this guide covers many common scenarios in detail, it is **not** intended to serve as a complete reference manual. For information on how to use Cartularius as an end user, refer to the *Getting Started with Cartularius for Users* guide. For installation, setup, and system configuration, see the *Getting Started with Cartularius for Admins* guide.

Chapter 2: Auto Folder Creation

What is Auto Folder Creation?

The Auto Folder Creation (AFC) feature is a great automation tool for structuring files and folders based on the relationship model and the state of records in your Salesforce instance.

In short, when a record is created in Salesforce, a folder hierarchy will be generated based on the AFC configuration.

Most of the configuration is stored in the following custom objects:

Object	Description
AFC Object	Basic object configuration per SObject and Record Type (Base Folder, Key Field, etc.).
AFC Folder	Basic folder configuration per SObject and Record Type (Folder Name, Parent Folder etc.).
AFC Folder File Category	List of File Categories that are allowed per AFC Folder.
Related Folder	Creates a folder relationship between two direct related records within a parent child relationship.
Indirect Related Folder	Creates a folder relationship between two records based on a junction record.

More information on AFC is available in the *Auto Folder Creation* chapter of the *Getting Started with Cartularius for Admins* guide.

SObject Configuration

For many standard objects, there is a trigger handler that starts the AFC process when the specific object is enabled in the setup.

When a record is created or updated, the trigger handler executes code to create or update the folder hierarchy configured for the corresponding *AFC Objects*, *AFC Folders,* and *AFC Folder File Categories*.

To have the same functionality for a custom object, the following steps must be taken.

- Add the custom object to the *CDM SObjects* Global Value Set
- Create a trigger for the custom object

Add Custom Objects to CDM SObjects (Global Value Set)

Take the following steps to add one or more Custom Objects to the *CDM SObjects* Global Value Set:

1. From Setup, enter *Picklist Value Sets* in the Quick Find box, then select **Picklist Value Sets**.

2. Open the *CDM SObjects* Global Value Set by clicking on its label.

3. Click **New**.

4. Enter the name of one or more custom objects (including the __c postfix).

5. Click **Save**.

After the Custom Objects have been added to the CDM SObjects Global Value Set, they are available when creating the AFC Objects and AFC Folders configuration.

Create Triggers for Custom Objects

The following example code shows how to enable the AFC feature for a custom object.

```
trigger AccountTrigger on Account (after insert, after update, after delete) {

    // After inserting the new record, create the folders according to the configuration.
    if (Trigger.isInsert) {
        try {
            GlobalDocumentManager.autoFolderCreation(Trigger.new);
        } catch (Exception e) {
            for (SObject record : Trigger.new) {
                record.addError('An error occurred during folder creation: ' +
e.getMessage());
            }
        }
    }

    // When the name of a record is updated, all corresponding folder resource path labels
should be updated accordingly.
    if (Trigger.isUpdate) {
        try {
            GlobalDocumentManager.renameRecords(Trigger.new, Trigger.oldMap);
        } catch (Exception e) {
            for (SObject record : Trigger.new) {
                record.addError('An error occurred while renaming folders for ' +
record.getSObjectType() + ': ' + e.getMessage());
            }
        }
    }

    // When a record is deleted, some maintenance needs to be performed.
    if (Trigger.isDelete) {
        try {
            GlobalDocumentManager.sfRecordDeletion(Trigger.old);
        } catch (Exception e) {
            for (SObject record : Trigger.old) {
                record.addError('An error occurred during the Salesforce record deletion
maintenance of Cartularius: ' + e.getMessage());
            }
        }
    }
}
```

Replace [TriggerName] and [CustomObject] with your values, and feel free to add any modifications to the code.

Chapter 3: Invocable Actions

Overview and Use Cases

Cartularius Document Management (CDM) includes a set of Apex **invocable actions** that can be used in Salesforce Flow or triggered programmatically to automate file and folder operations. These actions serve as a declarative interface to key CDM functionality, allowing developers and admins to build powerful, custom file workflows without writing low-level Apex.

Each invocable action encapsulates a specific task, such as creating a folder, moving a file, linking an existing ContentDocument to CDM, or updating metadata. They are designed to work with both **Salesforce Files** and **Amazon S3**, with differences in support noted per action.

Common User Cases

- Automatically create folders when a new record (e.g., Opportunity or Case) is inserted.
- Move or copy files between folders based on record status changes.
- Link files uploaded outside of CDM (e.g., email attachments) into the appropriate Cartularius folder.
- Organize files using Flow-driven logic (e.g., assign to a file category or version description).
- Integrate external software using Flow + invocable actions for simplified file injection.

Key Advantages

- Declarative: Easily used in Salesforce Flow without Apex code.
- Flexible: Can be combined with standard Flow actions, Apex classes, or REST API calls.
- Extensible: Enable advanced file workflows, custom record-file linking, and batch operations.
- Storage-aware: Handle storage differences transparently (e.g., S3 vs ContentDocument).

In the following sections, each invocable action is documented with its:

- Purpose and behavior
- Input parameters
- Output values
- Supported storage provider(s)

Available Invocable Actions

CDM Copy File

ApiName: CopyFile

Description:
Copies a file (including all its versions) from a source folder to a destination folder. The file will retain the same filename; ensure the `Filename` input matches the name you want in the destination path.

Inputs:
- **File Id** (Id) – The Id of the file to be copied.
- **Filename** (String) – The new filename for the copied file.
- **Source Resource Path** (String) – The resource path (identifier) of the source folder from which to copy.
- **Destination Resource Path** (String) – The resource path of the destination folder where the file will be copied.

Output: None (the action does not return a value).

Supported Storage Providers: Salesforce Files and Amazon S3 (works for files stored in either provider).

CDM Create Content Document

ApiName: CreateContentDocument

Description:
Creates a Salesforce **ContentDocument** (Salesforce File) from an existing Cartularius-managed file. In other words, it exports a file from the external storage into Salesforce Files.

Inputs:
- **File Id** (Id) – The Id of the Cartularius file to copy into Salesforce Files. This is the file record Id from the Cartularius system (often stored in Amazon S3).
- **Publish Location Id** (Id) – *(Optional.)* The Id of a Salesforce record (e.g., an Account, Case, etc.) where the new ContentDocument should be published/associated. If provided, the file will appear under that record's attachments/files.

Output: ContentDocument Id (Id) – The Id of the newly created Salesforce ContentDocument.

Supported Storage Providers: Requires Amazon S3 (source storage for the file) and Salesforce Files (target). This action is only available when both storage providers are enabled.

CDM Link Content Document

ApiName: CreateFileFromContentDocument

Description:
Creates a Cartularius file record in a specified folder by linking an existing Salesforce ContentDocument (Salesforce File). This **does not copy** the file data to S3; instead, it registers the Salesforce File in the Cartularius system (the file's storage provider will be marked as Salesforce Files).

Inputs:

- **Content Document Id** (Id) – The Id of the Salesforce ContentDocument (or ContentVersion) to link into Cartularius.
- **Folder Id** (Id) – The Id of the Cartularius folder where this file should be linked/stored logically.

Output: File Id (Id) – The Id of the new Cartularius file record that was created to represent the linked ContentDocument.

Supported Storage Providers: Salesforce Files (this action involves a Salesforce File as the source). *Note:* Typically used in environments where Salesforce Files is enabled as a storage provider. No Amazon S3 operation is performed (the file remains in Salesforce).

CDM Create Folder

ApiName: CreateFolder

Description:
Creates a new folder as a subfolder of the specified parent folder, with the provided name. This action calls the FolderManager service to perform the creation.

Inputs:

- **Id** (Id) – The Id of the parent folder in which to create the new folder. (This is the Cartularius folder record Id for the parent.)
- **Folder Name** (String) – The name for the new folder.

Output: None (the folder is created, but no value is returned to the flow).

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Delete Items

ApiName: DeleteItemsInvocable

Description:
Deletes the specified items (files, versions, or folders) by their Ids, respecting the system's deletion settings. Depending on the configuration, this may move items to a recycle bin or immediately purge them. In contrast to **Purge Items**, this action follows the standard deletion flow (soft-delete if enabled).

Inputs:

- **Record Ids** (List<Id>) – The list of file, file version, or folder record Ids to delete.

Output: None.

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Get File Categories for Folder

ApiName: GetFileCategoriesForFolder

Description:
Returns the list of **file categories** configured for a given folder. File categories typically define allowed file types or classifications for that folder.

Inputs:

- **Folder Resource Path** (String) – The resource path of the folder whose allowed file categories you want to retrieve.

Output: List of Strings – A list of file category names (or identifiers) that are configured for the folder. If no specific categories are set, this list may be empty.

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Get Folder Id

ApiName: GetFolderId

Description:
Retrieves a folder's Id given its resource path. Use this to look up the Salesforce record Id of a folder by providing its path in the Cartularius system.

Inputs:
- **Folder Resource Path** (String) – The resource path of the folder whose Id is needed.

Output: The **Folder Id** (Salesforce Id) corresponding to the given resource path, or null if no such folder exists.

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Get Folder Link

ApiName: GetFolderLink

Description:
Retrieves the **Folder Link** record for a given folder. The Folder Link (`Folder_Link__c`) is a custom object that represents a link or reference to the folder in the Cartularius system.

Inputs:
- **Folder Resource Path** (String) – The resource path of the folder for which to get the link record.

Output: Folder_Link__c record – Returns the Folder_Link__c object associated with the given folder. This record contains information linking the Salesforce context to the root folder of a record.

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Get Folder Resource Path

ApiName: GetFolderResourcePath

Description:
Retrieves the folder's **resource path** (unique path key) based on a given folder Id. This is essentially the inverse of *Get Folder Id*.

Inputs:
- **Folder Id** (Id) – The Id of the folder whose resource path should be retrieved.

Output: String – The resource path corresponding to the folder Id (or null if no such folder exists).

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Import Content Document

ApiName: ImportContentDocument

Description:
Imports a Salesforce ContentDocument (or a specific ContentVersion) into a Cartularius folder as a new **Amazon S3 file**. In effect, this copies the file data from Salesforce into the external S3 storage and creates a corresponding Cartularius file record.

Inputs:
- **Content Document/Version Id** (Id) – The Id of the Salesforce ContentDocument **or** ContentVersion to import from Salesforce. You can provide a ContentDocument Id (to import its latest version) or a specific ContentVersion Id.
- **Folder Id** (Id) – The Id of the Cartularius folder (in S3) where the file should be created.

Output: None (the file is imported to S3, and the operation does not return a value to the flow).

Supported Storage Providers: Requires both Salesforce Files (as the source) and Amazon S3 (as the target) to be enabled. This action will retrieve the file from Salesforce and upload it to S3.

CDM Is Create Folder Allowed

ApiName: IsCreateFolderAllowed

Description:
Checks if the current user is permitted to create a new subfolder within a given folder. This can be used to enforce folder-specific permissions.

Inputs:

- **Folder Resource Path** (String) – The resource path of the folder in question (the folder in which you want to create a subfolder).

Output: Boolean – Returns `true` if the user is allowed to create a folder in the specified location; `false` if not.

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Is File Category Restricted

ApiName: IsFileCategoryRestricted

Description:
Determines whether a given folder has an active *file category restriction*. A file category restriction means the folder is configured to allow only certain file types.

Inputs:

- **Folder Resource Path** (String) – The resource path of the folder to check for file category restrictions.

Output: Boolean – `true` if the folder has a file category restriction in place; `false` otherwise.

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Is Folder Locked by File Category Restriction

ApiName: IsFolderLockedByFileCategoryRestriction

Description:
Checks if a folder is effectively **locked** due to a file category restriction that has no allowed categories configured. In such a case, no files can be added to the folder because the restriction is present but empty.

Inputs:

- **Folder Resource Path** (String) – The resource path (or Id) of the folder to check.

Output: Boolean – `true` if the folder has a file category restriction defined but no file categories listed (meaning the folder is locked for uploads); `false` otherwise.

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Move File

ApiName: MoveFile

Description:
Moves a file (and all its versions) from one folder to another. Unlike copying, moving will remove the file from the source location after placing it in the target location.

Inputs:

- **File Id** (Id) – The Id of the file to be moved.
- **Source Resource Path** (String) – The resource path of the folder where the file currently resides (source folder).
- **Destination Resource Path** (String) – The resource path of the target folder where the file should be moved.

Output: None.

Supported Storage Providers: Salesforce Files and Amazon S3.

CDM Purge Items

ApiName: PurgeItemsInvocable

Description:
Permanently deletes the specified items (files, file versions, or folders) from the system. This "purge" action bypasses the recycle bin and cannot be undone.

Inputs:

- **Record Ids** (List<Id>) – A list of Ids for the file, file version, or folder records to purge permanently.

Output: None (items are deleted permanently with no return value).

Supported Storage Providers: Salesforce Files and Amazon S3 (the action will purge items from whichever storage provider they reside in).

CDM Set Current Version

ApiName: SetCurrentVersion

Description: Sets a specific file version as the current version of a file. This allows you to revert or promote an older version of a file to become the active version users see by default.

Inputs:

- **File Version Id** (Id) – The Id of the file version record that should become the current version. (The file version Id can typically be obtained from a prior query or action.)

Output: None.

Supported Storage Providers: Salesforce Files and Amazon S3. (This action applies to versioned files in the Cartularius system regardless of storage backend.)

CDM Get Folder Key (deprecated)

ApiName: GetFolderKey

This action was used to retrieve a folder's resource path based on a given folder Id. It has been replaced by **CDM Get Folder Resource Path**, which provides the same functionality with updated naming.

Using Invocable Actions in Salesforce Flow

Cartularius invocable actions are designed to be directly usable within **Salesforce Flow**, enabling developers and admins to build custom document workflows without Apex code. Each invocable action appears as a **Flow action** once the Cartularius managed package is installed.

This section outlines how to incorporate these actions into a Flow.

Adding an Invocable Action to a Flow

1. **Open Flow Builder**

   Navigate to *Setup* → *Flows* and create or edit a flow.

2. **Add an Action Element**

   In the Flow canvas, click **"+" → Action**.

3. **Search for CDM Actions**

   In the Action search, type keywords like `"CDM"`, `"Cartularius"`, or a specific operation such as `"Copy File"` or `"Create Folder"`.

4. **Configure Action Inputs**

   Provide the required input variables (e.g., File Id, Folder Resource Path). Most input fields are **text** or **record Ids**, so you can reference them from prior steps (such as a triggered record or variable).

5. **Capture Outputs (Optional)**

   Some actions return output values (e.g., Folder Id, File Id). You can store these in Flow variables for use in later steps.

Common Automation Scenarios

Example: Automatically Create a Folder When a Record Is Created

Use case: When a new `Case` is created, create a folder under `/Cases/{CaseNumber}`.

1. Create a **Record-Triggered Flow** on the `Case` object (triggered on create).
2. Use **Get Folder Id** to find the Id of the `/Cases` root folder.
3. Use **Create Folder** with:
    a. `Id = Folder Id` from step 2
    b. `Folder Name = {!$Record.CaseNumber}`

Example: Move a File to a "Closed" Folder When a Record Is Updated

Use case: When a Case is closed, move all its linked files to a `/Cases/Closed` folder.

1. Create a **Record-Triggered Flow** on `Case`, entry condition `Status = Closed`.
2. Use **Get Folder Id** for `/Cases/Closed`.
3. Loop over linked File__c records.
4. For each file:
    a. Use **Move File** with:
        i. `File Id = {!File__c.Id}`
        ii. `Source Path = {!File__c.Resource_Path__c}`
        iii. `Destination Path = "/Cases/Closed"`

Tips and Best Practices

- **Match Input Types:** Use Text or Id Flow variables where appropriate.
- **Chain Actions:** Combine folder creation, file linking, and metadata updates in sequence.
- **Check Folder Existence:** Use *Get Folder Id* before creating or moving into a folder.
- **Version Control:** Use *Set Current Version* to manage file version visibility when needed.
- **Handle Failures Gracefully:** Wrap actions in **fault paths** to log or alert on errors.

Chapter 4: External Integration

In this chapter, we cover how to integrate external systems with our document management solution via **Salesforce Files** and **Amazon S3**. We will explain how to connect external software to Salesforce and Amazon S3, how to upload and download files, and highlight the steps and code snippets from our Document Handler app. We also discuss Salesforce file limitations and how using Amazon S3 helps overcome some of them.

Salesforce Files

Salesforce Files (Content Documents) can be used as a storage backend for our app. In this section, we describe how an external application connects to Salesforce, uploads files into Salesforce, and downloads files from Salesforce. We also note the limitations of using Salesforce Files.

Salesforce File Limits

When using Salesforce Files, be aware of certain limits:

1. **File Size**
   Each file (ContentVersion) can be up to **2 GB** via the REST API. Larger files cannot be stored in Salesforce in a single piece. (If absolutely needed, one could implement client-side chunking with the REST API's chunked upload, but that's complex and beyond our scope.)

2. **Storage Limits**
   Files in Salesforce consume your org's file storage quota. Salesforce orgs have a limited file storage allocation (e.g., a per-user allotment or purchased storage). Storing many or large files may incur costs or hit limits in your org.

3. **Bandwidth/Throughput**
   Downloading or uploading many large files through Salesforce can be slower or constrained by API throughput limits. Salesforce also has a daily bandwidth limit for content deliveries.

4. **Versions and Metadata**
   Salesforce allows multiple versions per file (ContentDocument), which is useful for version history. There isn't a strict limit on the number of versions documented, but an extremely large number may impact performance or storage. Our app, by default, uses Salesforce Files for storing files, depending on configuration. When Salesforce Files are enabled and Amazon S3 is disabled, all file versions reside in Salesforce.

Connecting External Software to Salesforce

To allow an external service to communicate with Salesforce, you can use a number of methods. In this section, we outline the steps we took using our own external Document Handler web service. We set up an **External Client Application** – essentially a Salesforce Connected App configured for our integration. This Connected App is configured with OAuth settings (Consumer Key and a digital certificate) to support the **JWT Bearer OAuth 2.0 flow** for server-to-server authentication. Using the JWT flow allows our external Document Handler service to obtain an access token on behalf of a Salesforce user **without interactive login**, provided the user or profile is pre-authorized for the Connected App.

Steps to set up the connection:

1. **Create a Connected App in Salesforce**
   Enable OAuth and configure the required **OAuth scopes** (e.g., full access or API access). For JWT, upload an **X.509 certificate** (the public key) in the Connected App. Salesforce will use this to trust the JWT signed by the external app. Note the **Consumer Key** (Client ID) for the app.

2. **Authorize the Connected App**
   Set the policy so the integration user is **pre-authorized** (e.g., "Admin approved users are pre-authorized"), and assign the Connected App to the integration user's profile or permission set. This avoids the need for interactive user approval.

3. **Store credentials securely**
   In our implementation, the Consumer Key and the certificate's private key are stored securely (e.g., in Azure Key Vault). The external service (Document Handler) can retrieve these when needed.

4. **Construct a JWT and request an access token**
   The external app uses the stored credentials to create a signed JWT assertion, then posts it to Salesforce's OAuth token endpoint to get an access token. We use the **grant type** `urn:ietf:params:oauth:grant-type:jwt-bearer` with the JWT as the assertion. The JWT includes the Connected App's **Client ID** as the issuer, the Salesforce user's username as the subject, and is signed with the private key:

```
// Simplified example: create a JWT and exchange for an access token
var tokenDescriptor = new SecurityTokenDescriptor {
    Issuer = CLIENT_ID,                         // Connected App Consumer Key
    Audience = instanceUrl,                     // Salesforce instance URL (login audience)
    Subject = new ClaimsIdentity(new[] { new Claim("sub", USERNAME) }), // Salesforce username
    Expires = DateTime.UtcNow.AddMinutes(3),
    SigningCredentials = new SigningCredentials(new RsaSecurityKey(rsa), SecurityAlgorithms.RsaSha256)
};
string jwt = new JwtSecurityTokenHandler().CreateEncodedJwt(tokenDescriptor);

// Send JWT to Salesforce token endpoint
var request = new HttpRequestMessage(HttpMethod.Post, instanceUrl + "/services/oauth2/token") {
    Content = new FormUrlEncodedContent(new Dictionary<string, string> {
        {"grant_type", "urn:ietf:params:oauth:grant-type:jwt-bearer"},
        {"assertion", jwt}
    })
};
HttpResponseMessage response = await httpClient.SendAsync(request);
```

In the above code, `CLIENT_ID` is the Connected App consumer key, and `rsa` is the RSA private key for the certificate. Salesforce will validate the JWT and return an access token if the user and app are authorized. This access token is then used to authenticate subsequent Salesforce API calls (via an HTTP Authorization header).

Note: Instead of the JWT flow, one could use other OAuth flows (e.g., OAuth 2.0 Web Flow with a redirect), but our integration uses JWT for a seamless backend connection. Using the JWT Bearer flow requires setting up the Connected App with a certificate and pre-authorizing the user.

Uploading Files to Salesforce

Once the external app has an OAuth access token, it can upload files to Salesforce via the **REST API**. In Salesforce, files are stored as Content Documents (ContentDocument/ContentVersion records). Our Document Handler app uses the following process to upload a file to Salesforce:

Prepare a ContentVersion record

We create a new **ContentVersion** with the file's metadata and binary data. Salesforce REST API allows uploading a ContentVersion in a single multipart request.

In our implementation, we construct a multipart/form-data request with two parts:

- **Metadata part:** JSON content for the ContentVersion fields (e.g., *Title*, *PathOnClient*, etc.).
- **Binary part:** the file content stream, labeled as *VersionData*.

For example, when creating a **new file**, our JSON metadata includes `Title` (the file name without the extension) and `PathOnClient` (the full file name). If adding a **new version to an existing file**, we include the `ContentDocumentId` instead of the Title (to attach the version to that document). Below is a simplified snippet illustrating the request construction:

```
// Assume accessToken is obtained and fileStream is the file content stream
var content = new MultipartFormDataContent();
content.Add(new StringContent(jsonMetadata, Encoding.UTF8, "application/json"),
"entity_content");
content.Add(new StreamContent(fileStream) {
    Headers = { ContentType = new MediaTypeHeaderValue("application/octet-stream") }
}, "VersionData", fileName);

var uploadRequest = new HttpRequestMessage(HttpMethod.Post, instanceUrl +
"/services/data/v61.0/sobjects/ContentVersion");
uploadRequest.Headers.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);
uploadRequest.Content = content;
HttpResponseMessage uploadResponse = await httpClient.SendAsync(uploadRequest);
```

This multipart format is required for files > 50 MB (because non-multipart uploads are limited to ~37 MB of base64 data). Using multipart, Salesforce supports files up to **2 GB** for ContentVersion uploads. (By contrast, attachments in the old API were limited to 25 MB.) Our Document Handler sets a 2GB limit on request size to align with Salesforce's maximum. Keep in mind that extremely large file uploads may be slow and could require chunking for reliability, but Salesforce's limit is 2 GB per file.

Check the upload result

If the ContentVersion upload is successful, Salesforce returns a JSON with an `id` (the new ContentVersion Id). If this were the **first version** of a new file, we need the *ContentDocument ID* that Salesforce generated. We retrieve it by querying the ContentVersion record's `ContentDocumentId`. (This step is not needed if we were adding a version to an existing ContentDocument, since we already know the ContentDocumentId.)

Create ContentDocumentLink

Finally, we link the Content Document to the relevant Salesforce record (such as an Account, Case, or a custom record) by creating a **ContentDocumentLink**. This link object specifies the `ContentDocumentId` and the target record's Id (the *LinkedEntityId*), and we typically set share type and visibility. In our case, we use `ShareType = V` (Viewer permission) and `Visibility = AllUsers` so that the file is available to all users with access to the record. If needed, we can create multiple ContentDocumentLink records to link the same file to multiple records (our implementation allows an optional second LinkedEntityId).

After these steps, the file is successfully stored in Salesforce Files and linked to the appropriate record(s). The Salesforce UI will show the file in the record's related files, and the ContentDocument/ContentVersion metadata is in Salesforce.

Downloading Files from Salesforce

Downloading a file that is stored as a Salesforce Content Document can be done via the REST API (or SOAP API). In our integration, the external Document Handler service obtains the file and streams it back to the user. The process is:

1. **Identify the file/version to download**
   If you have a specific ContentVersion Id (Salesforce file version ID), you can download that version directly. If you only know the ContentDocument Id (the file's common Id), you may need to find the latest ContentVersion Id (or a specific version) via a query. For example, our service will query `SELECT Id FROM ContentVersion WHERE ContentDocumentId = '<docId>' ORDER BY VersionNumber DESC LIMIT 1` to get the latest version Id.

2. **Retrieve file metadata (optional)**
   We often fetch the ContentVersion metadata (by making a GET request to `/sobjects/ContentVersion/<Id>` with the access token) to retrieve fields like Title and FileExtension, which help us name the file and determine its type. This step is optional but user-friendly.

3. **Download the binary content**
   To get the actual file data, we perform a **GET request to the ContentVersion's VersionData** via the REST API. The endpoint format is:
   ```
   GET
   /services/data/vXX.X/sobjects/ContentVersion/<ContentVersionId>/VersionData
   ```

   with the Authorization header `Bearer <accessToken>`. This call returns the file's raw binary stream. In our .NET code, we simply stream this response back to the client:

   ```
   var dataRequest = new HttpRequestMessage(HttpMethod.Get,
       instanceUrl +
   $"/services/data/v61.0/sobjects/ContentVersion/{contentVersionId}/VersionData");
   dataRequest.Headers.Authorization = new AuthenticationHeaderValue("Bearer",
   accessToken);
   HttpResponseMessage dataResponse = await httpClient.SendAsync(dataRequest);
   Stream fileStream = await dataResponse.Content.ReadAsStreamAsync();
   ```

   The `fileStream` contains the file bytes, which we then return to the user (with an appropriate content type and filename).

With this approach, the external app handles the download by proxying the Salesforce file. Alternatively, if you have a front-end running in Salesforce (e.g., a Lightning component), you could use the native file download URL or the Lightning Data Service to fetch the file. But through an external integration, using the REST API as above is straightforward.

Note on Salesforce Files vs Attachments: Our solution uses Salesforce ContentDocuments (the modern "Files" feature), which offer higher size limits and the ability to link to multiple records. This differs from old Attachments (which were limited to 25 MB via the UI or 5 MB via the API). Salesforce Files is the recommended approach and what we support.

Limitations Recap: Relying solely on Salesforce Files means adhering to Salesforce storage limits and file-size caps. For organizations with extremely large or numerous files, this can become a bottleneck or cost issue. That's where our Amazon S3 integration comes in, which we discuss next.

Amazon S3

Amazon S3 can be used as an external file storage, integrated with our Salesforce app. In this configuration, files are not stored in Salesforce; instead, they reside in the customer's **own Amazon S3 environment**. This has several advantages: the customer has full control over their data, storage can scale virtually without limit, and it can alleviate storage and size constraints imposed by Salesforce.

In this section, we explain how to connect external software to the Amazon S3 environment, how file uploads to S3 work in our system, and how to download or access those files.

Connecting External Software to Amazon S3

When integrating external applications with Amazon S3, the application must authenticate securely, determine the correct file storage structure (key naming), and interact with S3 using supported APIs or SDKs.

Authentication and Access

To connect an external application to Amazon S3, AWS provides multiple authentication mechanisms. The most common include:

- **Access Key ID and Secret Access Key**
  These credentials identify and authorize the external client to interact with S3. They can be configured with granular IAM (Identity and Access Management) policies to restrict actions and resources.

- **Temporary Security Credentials (STS)**
  Using AWS Security Token Service, you can generate short-lived credentials, often through assumed roles. This is useful for enhanced security and delegation.

- **IAM Roles (for AWS-hosted applications)**
  If your external software runs on AWS (e.g., EC2, Lambda), it can assume roles with pre-attached policies, avoiding the need for hardcoded keys.

Uploading Files

To upload files to S3, external software can:

- Use the **S3 REST API** with properly signed requests (AWS Signature Version 4).
- Use an **AWS SDK** (available in many languages like Python, Java, .NET, JavaScript), which handles authentication, retries, and serialization.
- Use **multipart uploads** for large files (recommended for files over 100 MB).

Important considerations:

- Specify the bucket name and full key (path) for each upload.
- Optionally set metadata such as `Content-Type`, custom tags, or server-side encryption.
- Capture response values like `ETag` (checksum) and `VersionId` if versioning is enabled.

Downloading Files

To retrieve files:

- Perform a **GET Object** request using the bucket and key.
- Optionally generate a **pre-signed URL** for temporary public access without exposing credentials. Pre-signed URLs are useful when you want to allow downloads from a browser or third party without giving full access.

Example using AWS SDK (JavaScript):

```
const AWS = require('aws-sdk');
const s3 = new AWS.S3();

const params = {
  Bucket: 'your-bucket-name',
  Key: 'path/to/file.pdf',
};

s3.getObject(params, (err, data) => {
  if (err) console.error(err);
  else {
    // data.Body contains the file content
  }
});
```

Security Considerations

- Always use HTTPS when accessing S3.
- Limit credentials to the least privilege required (read-only vs full access).
- Use **server-side encryption** (SSE-S3 or SSE-KMS) for data-at-rest protection.
- Enable **logging** and **object versioning** for traceability and recovery.

Uploading Files to Amazon S3

Using Amazon S3 addresses many of the limitations in Salesforce file storage. Amazon S3 can handle *very large* files (single objects up to **5 TB** in size and essentially unlimited storage.

Our solution, **Cartularius Document Management (CDM)** stores files in S3 with a specific key naming scheme called the **resource path**. Understanding and using the resource path is crucial for external integrations.

Resource Path Structure: In CDM, every file's S3 key (path) encodes its folder location. The resource path is a string consisting of the hierarchy of folder IDs (Salesforce record IDs of Folder__c records), each followed by a forward slash, and ending with the file name. For example, a file in folder X (which might be under parent folder Y) could have a resource path like `Y/X/SomeFile.pdf`. In practice, our Folder__c records have a field, `Resource_Path__c`, that contains the path prefix for that folder (including the trailing `/`). When creating a new file, we take the folder's Resource_Path__c and append the file's name to form the full key. This ensures the file is stored in S3 in a folder that corresponds to its Salesforce folder structure.

Note: If you plan to upload files to S3 *outside of* the Salesforce UI (e.g., via an external script or application) and still have them recognized in Salesforce, you **must follow this resource path convention**. Otherwise, Salesforce (CDM) will not know where to find the file or how to organize it.

Procedure to upload a file to S3 and register it in Salesforce:

- **Obtain the Resource Path from Salesforce:** The external system should first determine where in the folder hierarchy the file will reside. Typically, you would have a target **Folder__c** record in Salesforce (or you create one via the API if needed). Using Salesforce API (REST or SOAP), retrieve that Folder__c's `Resource_Path__c` field. This gives you the key prefix. Then append the file's name to it to form the full `resourcePath` for the file. For example: if Folder__c with Id `a012...` has Resource_Path__c = `"a0120000001ABCxAAA/"` and your file is *Report.pdf*, then the resource path for the file becomes `"a0120000001ABCxAAA/Report.pdf"`. Ensure file names are S3-safe by replacing or encoding special characters; Also, we convert slashes in the name with underscores.

- **Upload the file to Amazon S3:** Use the AWS S3 API or SDK to upload your file bytes to the S3 bucket at the key equal to the resource path determined above. You can do this via a PUT object call. If using AWS CLI or SDK, it's straightforward (specify bucket name, key, and file). If implementing directly, ensure you include the correct headers and authentication signature. If your bucket has versioning enabled, AWS will generate a `VersionId` for the new object—capture this value, as we will store it in Salesforce. Also note the ETag returned by S3 (which is usually an MD5 hash of the file content, or a composite for multipart uploads).

Example: Using AWS CLI, an upload command might look like:

```
aws s3 cp /path/to/Report.pdf s3://<YourBucket>/<resourcePath>.
```

Using AWS SDK (pseudo-code):

```
s3Client.putObject(new PutObjectRequest(bucketName, resourcePath,
file));
String eTag = putObjectResult.getETag();
String versionId = putObjectResult.getVersionId(); // if versioning
is enabled
```

Ensure that the upload is performed with the correct AWS credentials and region. On success, the file is now stored in S3, and the client has the key (path), ETag, and possibly VersionId.

- **Create a File__c record in Salesforce:** The final step is to notify Salesforce of this new file so it appears in our app's UI and metadata. This is done by creating a new **File__c** record (and a related File_Version__c record) via the Salesforce API. Our app provides an Apex method `FileManager.upsertFile` (and a bulk variant) that we use internally, but as an external integrator, you can perform an equivalent via the REST API (sObject insert) or Apex REST if exposed. The File__c record should be populated with the file's metadata. **All of the following fields need to be set** to properly register the file:
    - **Name:** (Standard field) The Name of the File__c record. We set this to the file's name (truncated to 80 characters if longer).
    - **File_Name__c:** The full file name (including extension).
    - **Extension__c:** The file's extension (e.g. "pdf", "xlsx"), in lowercase.
    - **Resource_Path__c:** The exact resource path used as the S3 object key (full folder path + file name).
    - **Size__c:** File size in bytes (this should match the size of the object in S3).
    - **Storage_Class__c:** S3 storage class of the object. By default, this is "STANDARD" for new uploads (this field is a placeholder for future functionality).
    - **Folder__c:** Lookup to the Folder__c record in which this file resides (the folder Id you used to get the resource path).
    - **Last_Modified_On_Server__c:** The timestamp when the file was last modified on the server. We usually take the S3 `LastModified` timestamp of the object and convert it to a Salesforce datetime. In our code, we make an S3 HEAD request to retrieve the last modified time. Ensure this field is set (or you can use the current datetime if nothing else).
    - **ETag__c:** The ETag returned by S3 for the object. This acts as an identifier for the file content (and helps with change detection).
    - **Current_File_Version_Version_Id__c:** If your bucket has versioning and this upload generated a Version ID, set this field to that S3 VersionId. If versioning is off, you can leave this blank or set it to null.

  - **Bucket__c:** Reference to the Bucket__c record that represents the S3 bucket used. (We prepared the use of multiple buckets for future use, but for now use the Id of the default bucket)
  - **Hidden__c:** A checkbox indicating if the file is hidden. This should be **false** (we use it for internal filtering).
  - **SObject__c:** Every folder is associated with a particular Salesforce object type, set this to the API name of that object. In our code, we derive it from the folder via an internal function.
  - **Storage_Provider__c:** Indicate the storage provider as "Amazon S3" (as opposed to "Salesforce Files" in the other mode). This helps our app know where the file is stored.
  - **Content_Document__c:** (Optional, used only if integrating with Salesforce Files directly). In S3 mode, we **leave this null** because there is no Salesforce ContentDocument for the file (the file is external).
  - **Description__c:** (Optional) Any description text for the file.
  - **File_Category__c:** (Optional) A categorization if your implementation uses it (it can be made required by configuration. If required, make sure to add a file category that is valid for the corresponding folder)

  After setting these fields, insert the File__c record via API. If done correctly, a related **File_Version__c** record will typically be auto-created (via trigger or by calling a similar Apex method) to log this specific version upload. In our implementation, after upserting File__c, we explicitly insert a File_Version__c with details of this version (Version_Id__c = S3 VersionId, Size, ETag, etc.). The File_Version__c is a child object tracking each version's metadata.

- **Link the File to a record:** In our S3 approach, File__c records themselves represent files and are organized under Folder__c hierarchy, which in turn is tied to a specific Salesforce record (e.g., an Account's folder). If your integration created the file under the correct Folder__c, it's effectively "linked" to that record by hierarchy.

Once the File__c is created, our app will treat it like any other file. Users will see it in the UI under the corresponding folder, and they can preview or download it. The actual content is fetched from S3 when needed.

Downloading Files from Amazon S3

For files stored in Amazon S3, downloading (or accessing) them involves retrieving the object from the S3 bucket. There are two main scenarios to consider:

- **Within the Salesforce app (user clicks download in UI):** Our app will handle this by making an Apex callout to S3 to get the file, then streaming it to the user. The user experience is seamless – they click "Download" and behind the scenes, the app knows the file's resource path and uses the stored AWS credentials to fetch the object.

- **External integration (direct S3 access):** If you are integrating outside of Salesforce (e.g., a separate application or a script) and want to download a file uploaded via the above process, you can use the AWS S3 API/SDK directly. You will need the **bucket name**, the object's **resource path (key)**, and valid AWS credentials. With those, you can perform a GET Object request.

  For example, using AWS CLI:
  ```
  aws s3 cp s3://<YourBucket>/<folderIds>/Report.pdf ./Report.pdf
  ```
  Or using an SDK, `s3Client.getObject(new GetObjectRequest(bucketName, resourcePath), destinationFile)`. This will retrieve the file from S3. The `resourcePath` here must exactly match what was stored in Salesforce (which you can obtain via a query to File__c.Resource_Path__c if needed).

If you prefer to generate a **presigned URL** (a time-limited public URL to download the file without credentials), you can do so using AWS SDKs (e.g., `GeneratePresignedUrlRequest` in AWS SDK for Java or `S3Client.generate_presigned_url` in AWS SDK for Python).

You'd provide the key and an expiration time to get a URL that anyone with the link can use to download. Since our app doesn't currently provide a one-click presigned URL, an advanced integration could handle this on its own if needed.

Data Control and Security

Because the files in S3 are in the customer's AWS account, the customer has **full control**. They can revoke the keys or use IAM policies to control which operations our Salesforce integration can perform. All data in the S3 bucket can be encrypted at rest (SSE-S3 or SSE-KMS keys). Our app supports both Amazon S3-managed keys and KMS keys as configurable options. When downloading, our integration will also honor those settings (for example, including the proper headers for KMS decryption if required). From a compliance standpoint, this means no file data is stored in Salesforce's cloud. It resides in the customer's cloud, which can simplify certain regulatory requirements.

FEEDBACK

Thank you for using Cartularius. If you encounter any problems, irregularities, or anything else while using our software or reading our documentation, do not hesitate to contact us at support@upperspire.com. Your feedback would help us improve our products and services for you and other customers.

If you like CDM, we would be grateful if you could write a review in the AppExchange listing's review section. This would help us reach a larger audience and allow our company to grow and develop better products and services.

MORE RESOURCES

CDM AppExchange listing
https://appexchange.salesforce.com/appxListingDetail?listingId=a0N4V00000GYhhhUAD

CDM website
https://www.cartularius.com/

Upper Spire website
https://www.upperspire.com

Amazon S3 user guide
https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

Amazon S3 console URL
https://s3.console.aws.amazon.com/

Best Practices on Amazon AWS security
https://aws.amazon.com/premiumsupport/knowledge-center/security-best-practices/

Setup MFA for Amazon AWS
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_virtual.html#enable-virt-mfa-for-root

UPPER SPIRE | the summit of Salesforce cloud apps