d.velop

d.velop connect for salesforce
CRM API

Table of Contents

1. d.velop connect for Salesforce CRM API

1.1. Apex development guide

This section gives you an overview of the various API functions for development in Salesforce.

1.1.1. Uploading Salesforce files to the DMS

This chapter shows you how to upload Salesforce files (**ContentDocuments**, **Attachments** and **EmailMessages**) to the DMS as documents. You can upload the files and records individually or migrate all original record's attachments directly to the DMS.

Requesting a valid user session for the process

To ensure that the upload is performed in the included **queueable** process with a valid user session, the user session must be requested first in d.velop documents. You can either request the user session manually or have it performed by the process.

The following sections show you how to get a valid user session:

Contents

Pre-authentication (recommended)

The DocumentUploader class provides several methods to request a valid user session in advance. For example, you can directly retrieve a session for the logged-in user or service user. Alternatively, you can also use a session based on a previously obtained cookie (e.g. using the OpenAPI class methods).

The following shows you how to use the various methods.

Pre-authentication – Logged-in user

```
// Create the document uploader - This can happen anywhere in your code
dvelop_docs_dev.DocumentUploader docUploader = new
dvelop_docs_dev.DocumentUploader();

// Authenticate as the currently logged in user
docUploader.authenticateInCurrentUserContext();

// Define upload request(s)
// ...

// Start the process
docUploader.startUpload(requests);
```

Pre-authentication – Service user

```
// Create the document uploader - This can happen anywhere in your code
dvelop_docs_dev.DocumentUploader docUploader = new
dvelop_docs_dev.DocumentUploader();

// Authenticate as the service user
docUploader.authenticateInServiceUserContext();

// Define upload request(s)
// ...

// Start the process
docUploader.startUpload(requests);
```

Pre-authentication – With cookie

```
// Create the document uploader - This can happen anywhere in your code
dvelop_docs_dev.DocumentUploader docUploader = new
dvelop_docs_dev.DocumentUploader();

// Fetch a cookie (cached or completely fresh) and pass it to the uploader
that starts the process
String cookie = dvelop_docs_dev.OpenAPI.getValidCookieForConfigUser();
docUploader.setAuthentication(cookie);

// Define upload request(s)
// ...

// Start the process
docUploader.startUpload(requests);
```

Authentication during the process

In the event that a valid user session has not been obtained in advance, the upload process requests a session in d.velop documents once using the internal **queueable**class **AuthenticationJob**. Either a session already cached by the **Salesforce Platform Cache** is used or a completely new session is requested in **Identity Provider**.

You can control the behavior of **AuthenticationJob** using the DocumentUploadOptions class and, for example, run the process using the service user's credentials.

> **Note**
> If a session was not requested in advance and **DocumentUploadOptions** was not transferred to the process, the **Fallback Options** (**useConfigUser = false** – **skipValidate = false**) are used.

The following example shows you how you can customize the process parameters using the class.

Authentication during the process – Controlling the options

```
// Create the document uploader - This can happen anywhere in your code
dvelop_docs_dev.DocumentUploader docUploader = new
dvelop_docs_dev.DocumentUploader();

// Define your custom options
dvelop_docs_dev.DocumentUploadOptions options = new
```

```
dvelop_docs_dev.DocumentUploadOptions(true, false);

// Define upload request(s)
// ...

// Start the process - Use the overloaded method signature
"startUpload(requests, options)"
docUploader.startUpload(requests, options);
```

See also

- Using the class “DocumentUploader”
- Using the “SubscriberInterface” class
- Using the class “OpenAPI”
- Using the “DocumentUploadOptions” class

Uploading “ContentDocument” and “Attachment” files

This section shows you how to upload single or multiple **ContentDocument** or **Attachment** files from Salesforce to the DMS.

Contents

Uploading a ContentDocument/Attachment file

In the context of a record, you can use the **ContentDocumentUploadRequest** class and the **startUpload(requests)** method to upload a **ContentDocument** or **Attachment** file.

The following example shows you how to use the class.

```
// Create a new uploader instance
dvelop_docs_dev.DocumentUploader docUploader = new
dvelop_docs_dev.DocumentUploader();

// Authenticate against connected DMS
docUploader.authenticateInCurrentUserContext();

// Define upload request(s)
dvelop_docs_dev.ContentDocumentUploadRequest request = new
dvelop_docs_dev.ContentDocumentUploadRequest();
request.relatedEntityId = '0019Z00000I7cpRQAR';
request.contentDocumentId = '0699Z000001ygrqQAA';
request.documenttypeKey = 'Schriftverkehr_Kunde';
request.preserveFileAfterUpload = true;

List<dvelop_docs_dev.IDocumentUploadRequest> uploadRequests = new
List<dvelop_docs_dev.IDocumentUploadRequest>{
    request
};

// Perform the action - Start the upload process
docUploader.startUpload(uploadRequests);
```

To upload multiple **ContentDocument/Attachment** files or other items one at a time, expand the list to include additional **IDocumentUploadRequest** instances.

See also

- Using the class “DocumentUploader”
- Using the “ContentDocumentUploadRequest” class

Uploading “EmailMessage” items

This chapter shows you how to upload single or multiple **EmailMessage** items to the DMS.

Contents

Uploading an EmailMessage item

In the context of a record, you can use the **EmailMessageUploadRequest** class and the **startUpload(requests)** method to upload a **EmailMessage** item.

The following example shows you how to use the class.

```
// Create a new uploader instance
dvelop_docs_dev.DocumentUploader docUploader = new
dvelop_docs_dev.DocumentUploader();

// Authenticate against connected DMS
docUploader.authenticateInCurrentUserContext();

// Define upload request
dvelop_docs_dev.EmailMessageUploadRequest request = new
dvelop_docs_dev.EmailMessageUploadRequest();
request.relatedEntityId = '0019Z00000I7cpRQAR';
request.emailMessageIds = new List<Id>{ '02s9Z000003G7TeQAK' };
request.documenttypeKey = 'Schriftverkehr_Kunde';

List<dvelop_docs_dev.IDocumentUploadRequest> uploadRequests = new
List<dvelop_docs_dev.IDocumentUploadRequest>{
    request
};

// Perform the action - Start the upload process
docUploader.startUpload(uploadRequests);
```

You can upload multiple **EmailMessage** or other items one at a time by expanding the list to include additional **IDocumentUploadRequest** instances.

See also

- Using the class “DocumentUploader”
- Using the “EmailMessageUploadRequest” class

Uploading new versions

This chapter shows you how to upload **ContentDocument** or **Attachment** files as new versions of DMS documents.

Contents

Uploading ContentDocument/Attachment files as new versions

You can upload **ContentDocument** and **Attachment** files as a new version of a DMS document using the **UploadNewVersionRequest** class and the **startUpload(requests)** method.

The following example shows you how to use the class.

```
// Create a new uploader instance
dvelop_docs_dev.DocumentUploader docUploader = new
dvelop_docs_dev.DocumentUploader();

// Authenticate against connected DMS
docUploader.authenticateInCurrentUserContext();

// Define upload request
dvelop_docs_dev.UploadNewVersionRequest request = new
dvelop_docs_dev.UploadNewVersionRequest();
request.contentDocumentId = '0699Z000001ygrqQAA';
request.dmsDocumentId = 'XH00020875';

List<dvelop_docs_dev.IDocumentUploadRequest> uploadRequests = new
List<dvelop_docs_dev.IDocumentUploadRequest>{
    request
};

// Perform the action - Start the upload process
docUploader.startUpload(uploadRequests);
```

You can upload multiple new versions or items one at a time by expanding the list to include additional **IDocumentUploadRequest** instances.

See also

- Using the class “DocumentUploader”
- Using the “UploadNewVersionRequest” class

Migrating Salesforce files and e-mails connected to records

This chapter shows you how to migrate **ContentDocument** and **Attachment** files connected to records, as well as **EmailMessage** items, to the DMS.

Contents

Migrating Salesforce files and EmailMessage items

You can unpin and migrate files and **EmailMessage** items from a specific record using the **AttachmentMigrationRequest** class and the **startUpload(requests)** method. You can either keep the migrated items in the appropriate location or delete them from Salesforce to save storage space.

The following example shows you how to use the class.

```
// Create a new uploader instance
dvelop_docs_dev.DocumentUploader docUploader = new
dvelop_docs_dev.DocumentUploader();

// Authenticate against connected DMS
docUploader.authenticateInCurrentUserContext();
```

```
// Define upload request
dvelop_docs_dev.AttachmentMigrationRequest request = new
dvelop_docs_dev.AttachmentMigrationRequest();
request.relatedEntityId = '0019Z00000I7cpVQAR';
request.docTypeKey = 'Schriftverkehr_Kunde';
request.fileTypesToProcess = '*';
request.transferAndRemoveEmails = true;
request.preserveFilesAfterTransfer = true;
request.preserveEmailsAfterTransfer = true;

List<dvelop_docs_dev.IDocumentUploadRequest> uploadRequests = new
List<dvelop_docs_dev.IDocumentUploadRequest>{
    request
};

// Perform the action - Start the upload process
docUploader.startUpload(uploadRequests);
```

You can process multiple records one at a time or upload items by expanding the list to include additional **IDocumentUploadRequest** instances.

See also

- Using the class “DocumentUploader”
- Using the “AttachmentMigrationRequest” class

1.1.2. Managing and processing DMS documents

This section shows you how to work with DMS documents in Apex and manage and process DMS documents in a simple and efficient way.

Authentication in d.velop documents

This chapter shows you how to authenticate the **DocumentManager** class in d.velop documents and explains why this is advisable.

Contents

Background

Pre-authentication is not absolutely necessary to use the **DocumentManager** class methods in your custom code. However, we do recommend that you add a valid user session to an instance once using the different authentication methods.

This ensures that a valid session exists and that all outgoing HTTP requests to d.velop documents are performed in the context of the respective user.

It also avoids having to re-validate a user session cached in the Salesforce Platform Cache in the Identity Provider app. In this case, you may need to re-request the user session, potentially leading to a **Mixed DML Operation** error in Salesforce.

Class authentication

The following example shows you how to authenticate the **DocumentManager** class quickly and easily.

```
// Create the document manager
dvelop_docs_dev.DocumentManager documentManager = new
```

```
dvelop_docs_dev.DocumentManager();

// Pick one of the authentication methods and authenticate
documentManager.authenticateInCurrentUserContext();

// Perform an operation
// ...
```

See also

- Using the class “DocumentManager”
- Using the “SubscriberInterface” class

Updating a DMS document

This chapter shows you how to update a DMS document’s properties and categories.

Contents

Updating a document

AttributYou can update a DMS document using the DocumentManager class **updateDocument(documentId, updatedCategory, updatedAttributes)** method.

The following example shows you how to use the method.

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();

// Define the updated details
List<dvelop_docs_dev.DocumentAttribute> updatedAttributes = new
List<dvelop_docs_dev.DocumentAttribute>{
    new dvelop_docs_dev.DocumentAttribute('objecttitle', 'Ja'),
    new dvelop_docs_dev.DocumentAttribute('multivalue_property__c', new
List<String>{ 'A', 'B' })
};

// Perform the action - Update the document
documentManager.updateDocument('XH00014213', 'correspondence__c',
updatedAttributes);
```

Error trapping

When you update a document, you may receive the following error message: “A mapping is not possible because several values were transferred to the same destination”.

The following shows you the possible error causes and solutions:

An individual attribute appears in the attribute list multiple times.

Ensure that the transferred list contains each attribute just once. You can remove identical attributes by converting the list into a set. You can then convert the set back into a list.

Multiple source properties are assigned to the same d.3 target field

Check the mappings and remove any duplicates.

See also

- Using the class “DocumentManager”
- Using the class “DocumentAttribute”

Retrieving a DMS document

This chapter shows you how to determine a DMS document’s properties and categories from the connected DMS.

Contents

Retrieving a document

You can retrieve an individual document from the DMS using the DocumentManager class **getDocument(documentId)** method.

The following example shows you how to use the method.

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();

// Perform the action - Get the document and work with ouput
dvelop_docs_dev.Document dmsDocument =
documentManager.getDocument('XH00014213');
System.debug(dmsDocument);
```

See also

- Using the class “DocumentManager”
- Using the class "Document"

Creating a folder

This chapter shows you how to create a folder.

Contents

Creating a folder

You can create a folder in the DMS using the DocumentManager class **createFolder(category, attributes)** method.

The following example shows you how to use the method.

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();
```

```
// Define the details
List<dvelop_docs_dev.DocumentAttribute> attributes = new
List<dvelop_docs_dev.DocumentAttribute>{
    new dvelop_docs_dev.DocumentAttribute('objecttitle', 'Account4711'),
    new dvelop_docs_dev.DocumentAttribute('objectkey',
'001AW00000B53c5YAB'),
    new dvelop_docs_dev.DocumentAttribute('account', 'Account4711'),
    new dvelop_docs_dev.DocumentAttribute('accountnumber',
'001AW00000B53c5YAB'),
    new dvelop_docs_dev.DocumentAttribute('foldertype', 'Account')
};

// Perform the action - Create the folder
documentManager.createFolder('akte_safo__c',  attributes);
```

See also

- Using the class “DocumentManager”
- Using the class “DocumentAttribute”

Downloading DMS documents

This chapter shows you how to download documents from the DMS and attach them to a record.

Contents

Technical limitations

As with all Apex transactions, you also have to keep the Salesforce governor limits in mind when downloading documents. The following technical limitations with regard to the size of HTTP responses apply:

- In a synchronous transaction, only documents with a maximum file size of 6 MB can be downloaded.
- In an asynchronous transaction, only documents with a maximum file size of 12 MB can be downloaded.

Downloading a document

You can download an individual document from the DMS using the DocumentManager class **downloadDocument(documentId)** and **downloadDocumentToRecord(recordId, documentId)** methods.

The following examples show you how to use the two methods.

Downloading a document – Processing the document

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
```

```
documentManager.authenticateInCurrentUserContext();

// Perform the action - Download the document and work with ouput
dvelop_docs_dev.DocumentDownloadResult downloadResult =
documentManager.downloadDocument('XH00014213');
System.debug(downloadResult.getFilename());
System.debug(downloadResult.getBody());
```

Downloading a document – Saving the document to the record

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();

// Perform the action - Download the document and attach it to a record
Id createdContentDocumentId =
documentManager.downloadDocumentToRecord('001Aa0000059kyPIAQ',
'XH00014213');
System.debug(createdContentDocumentId);
```

Downloading multiple documents

You can download multiple documents with an action from the DMS using the DocumentManager class **downloadDocuments(documentIds,finisher, proceedOnFailure)** method.

Since the method downloads multiple potentially large files for the DMS documents, the downloads are performed in a queue in an asynchronous process. To ensure you can respond to the completion of the queue and access the downloaded documents, you must transfer a **queueable** job of the type dvelop_docs_dev.AsyncProcessFinisher when calling the method. This job is added to the queue following the completion of the last download. The output (the downloaded documents) is transferred from dvelop_docs_dev.DocumentDownloadResults as a list.

The following example shows you how to define a finish job to download multiple documents asynchronously:

Downloading multiple documents – Defining the finish job

```
public class DownloadProcessFinishJob extends
dvelop_docs_dev.AsyncProcessFinisher {
    private static final String DEFAULT_PARAM_NAME = 'downloadResults';

    public DownloadProcessFinishJob() {
        // Call of parent constructor with an identifier for the default
parameter name
        super(DEFAULT_PARAM_NAME);
    }

    public override void execute(Map<String, Object> processOutput) {
        // Read the default parameter and cast to correct type
        Object defaultParamOutput = processOutput.get(DEFAULT_PARAM_NAME);
        List<dvelop_docs_dev.DocumentDownloadResult> downloadResults =
(List<dvelop_docs_dev.DocumentDownloadResult>) defaultParamOutput;

        // Process the results
        for (dvelop_docs_dev.DocumentDownloadResult result :
```

```
downloadResults) {
            System.debug(result.getFilename());
            System.debug(result.getBody());
        }
    }
}
```

Downloading multiple documents – Starting the process

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();

// Defining download params
List<String> documentIds = new List<String>{ 'XH00014226', 'XH00014222',
'XH00014225' };
DownloadProcessFinishJob finishJob = new DownloadProcessFinishJob();

// Perform the action - Download the documents in a queue
documentManager.downloadDocuments(documentIds, finishJob, false);
```

See also

- Using the class “DocumentManager”
- Using the class “DocumentDownloadResult”
- Using the class “AsyncProcessFinisher”

Deleting DMS documents

This chapter shows you how to delete documents and folders from the DMS.

Contents

Deleting a DMS item

You can delete a document or item from the DMS using the DocumentManager class **deleteDocument(documentId, ?reason)** method.

The following example shows you how to use the method.

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();

// Perform the action - Delete the document
documentManager.deleteDocument('XH00014213', 'My custom reason');
```

See also

- Using the class “DocumentManager”

Searching for DMS documents

This chapter shows you how to search for and find documents for a specific Salesforce record or Salesforce item in the DMS.

Contents

Searching for DMS documents

This example shows you how to perform a simple search for documents on a specific record using the DocumentManager class **search(options)** method.

By default, the search criteria used to restrict the documents are determined based on the settings configured in the d.velop documents configuration.

> **Note**
>
> The **search(options)** method requires you to enter a **DocumentSearchOptions** class instance.
>
> The Using “DocumentSearchOptions.Builder” class section shows you how to create simplified search parameters.

Searching for DMS documents – Simple search

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();

// Define the search options
dvelop_docs_dev.DocumentSearchOptions searchOptions = new
dvelop_docs_dev.DocumentSearchOptions();
searchOptions.useRecordContext('00011100001tBgaxAAC');

// Perform the action - Search for documents using the defined options
List<dvelop_docs_dev.Document> documents =
documentManager.search(searchOptions);
```

Searching for DMS documents – Identifying all the documents in a Salesforce record

```
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();
documentManager.authenticateInCurrentUserContext();
```

```
dvelop_docs_dev.DocumentSearchOptions searchOptions = new
dvelop_docs_dev.DocumentSearchOptions();
searchOptions.useRecordContext('00011100001tBgaxAAC');

List<dvelop_docs_dev.Document> documents =
documentManager.search(searchOptions);
```

Searching for DMS documents – Identifying all the documents in a Salesforce item

```
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();
documentManager.authenticateInCurrentUserContext();

dvelop_docs_dev.DocumentSearchOptions searchOptions = new
dvelop_docs_dev.DocumentSearchOptions();
searchOptions.useObjectContext('Account');

List<dvelop_docs_dev.Document> documents =
documentManager.search(searchOptions);
```

Searching for DMS documents – Identifying all the documents in a Salesforce item with overridden search criteria

```
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();
documentManager.authenticateInCurrentUserContext();

dvelop_docs_dev.DocumentSearchOptions searchOptions = new
dvelop_docs_dev.DocumentSearchOptions();
searchOptions.useObjectContext('Account');
searchOptions.addSearchAttribute('objectkey', '0100X00QGA');
searchOptions.addSearchAttribute('string1', 'Max Mustermann');

List<dvelop_docs_dev.Document> documents =
documentManager.search(searchOptions);
```

Searching for documents using IDs

You can also identify multiple documents based on their IDs. You can do this using the DocumentManager class **searchDocumentsWithIds(searchAttributeKey, documentIds)** method.

```
dvelop_docs_dev.DocumentManager docManager = new
dvelop_docs_dev.DocumentManager();
docManager.authenticateInCurrentUserContext();

List<String> documentIds = new List<String>{ 'C200000024', 'C200000023' };

List<dvelop_docs_dev.Document> documents =
docManager.searchDocumentsWithIds('externaldocumentnumber', documentIds);
```

See also

- Using the class “DocumentManager”
- Using the class “DocumentSearchOptions”
- Using the class "Document"

Sharing DMS documents

This chapter shows you how to share DMS documents with other people.

Contents

Creating links for sharing documents

You can create links for sharing documents using the DocumentManager class **shareDocuments(downloadUrls)** method.

The following example shows you how to use the method.

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();

// Get a document
dvelop_docs_dev.Document dmsDocument =
documentManager.getDocument('FP00000010');

// Define the urls
Set<String> downloadUrls = new Set<String>{
    dmsDocument.downloadUrl
};

// Perform the action - Create the temporary links
Map<String, String> linksByDownloadUrl =
documentManager.shareDocuments(downloadUrls);
System.debug(linksByDownloadUrl.get(dmsDocument.downloadUrl));
```

See also

- Using the class “DocumentManager”
- Using the class "Document"

Sending e-mails with attachments

This chapter shows you how to send e-mails with Salesforce and DMS documents as attachments.

Contents

Technical limitations

As with all Apex transactions, you also have to keep the Salesforce governor limits in mind when sending e-mails with documents. Due to the maximum heap size (magnitude of dynamic data storage) for e-mail services, the following technical limitations apply:

- The maximum permitted size of the e-mail attachments together is 36 MB.
- Since DMS documents must be temporarily downloaded from the DMS before sending them, the same technical limitations as described in the Downloading DMS documents chapter also apply.

Sending an e-mail

You can send documents from Salesforce and the DMS via e-mail using the DocumentManager class **sendEmail(options)** method.

The following example shows you how to use the method.

```
// Create a new manager instance
dvelop_docs_dev.DocumentManager documentManager = new
dvelop_docs_dev.DocumentManager();

// Authenticate against connected DMS
documentManager.authenticateInCurrentUserContext();

// Define the email options
dvelop_docs_dev.DocumentEmailOptions emailOptions = new
dvelop_docs_dev.DocumentEmailOptions();
emailOptions.setSubject('Important Message');
emailOptions.setBody('Have a look at this.');
emailOptions.addRecipient('astro@salesforce.com');
emailOptions.addContentDocumentId('069AP0000006hQbYAI');
emailOptions.addDmsDocumentId('XH00014846');

// Perform the action - Send an email with documents from Salesforce and
the DMS
documentManager.sendEmail(emailOptions);
```

See also

- Using the class “DocumentManager”
- Using the class “DocumentEmailOptions”

1.1.3. Dynamically determining values with Apex

In the d.velop documents configuration, you can create mappings between Salesforce objects and data record types. This creates a direct connection between Salesforce fields and the properties of DMS documents. This no-code approach makes it easy to implement many application scenarios without any programming knowledge. For more complex application scenarios in which values cannot be determined from data records and other Salesforce sources, you can add your own Apex logic to the d.velop documents configuration. You can then determine the values from a data record dynamically.

The following example shows you how to create a dynamic value determination using the class **dvelop_docs_dev.RecordValueProvider**.

Example

```
// Extend the provider parent class to be able to configure your class in
the d.velop documents configuration
global class CustomRecordValueProvider extends
dvelop_docs_dev.RecordValueProvider {

    // Implement the "provide" method, which is then called by the internal
process you want to integrate in
    global override dvelop_docs_dev.RecordValueProviderOutput
provide(dvelop_docs_dev.RecordValueProviderInput input) {
        // Read info from the given input
        Id recordId = input.getRecordId();

        // Work with the input data / perform further logic / call an
```

```
endpoint
        List<System.SelectOption> selectOptions = new
List<System.SelectOption>{
            new System.SelectOption('valueA', 'Label A'),
            new System.SelectOption('valueB', 'Label B'),
            new System.SelectOption(recordId, 'Record ID')
        };

        // Use one of the factory methods to return an appropriate output
object for the internal process
        return
dvelop_docs_dev.RecordValueProviderOutput.picklist(selectOptions);
    }

}
```

1.1.4. Testing d.velop Apex code

In Apex, you can write unit and integration tests in a number of different ways. Salesforce provides several options for designing tests efficiently, securely and reliably.

This chapter shows you how to record your Apex classes in your tests. You can also learn what to keep in mind while doing so to avoid problems.

Generating the test environment

Assembling and adding test data is often time-consuming and inefficient. Avoid this effort with unit tests as poorly structured unit tests are usually costly and confusing.

> **Note**
> You can find out how to avoid inefficiently generated test data and isolated standalone unit tests in chapter Using mocks for isolated unit tests.

For integration tests or unit tests with access to test data, you can use the dvelop_docs_dev.OpenAPI class. The **initializeTestEnvironment()** method ensures that the assembly of test data and HTTP callout mocks is kept to the absolute minimum. This allows you to successfully call the d.velop classes and methods in your tests.

The following example shows you how to use the method.

```
@IsTest
private class MyTestClass {
    @TestSetup // Notice the "TestSetup" annotation
    static void makeData() {
        dvelop_docs_dev.OpenAPI.initializeTestEnvironment();
    }

    @IsTest
    static void myUnitTest() {
        // TODO: implement unit test
    }
}
```

Using mocks for isolated unit tests

“Mocking” is a software engineering model that is frequently used for writing efficient, isolated and reliable unit tests. Mocking frameworks such as Mockito for Java have been developed with this in mind.

With Salesforce too, you can simulate dependencies between Apex classes using the stub API. This allows you to perform independent tests without adding test data beforehand. ApexMocks is one of the leading mocking frameworks. It is heavily based on Mockito and the concept of “dependency injection”.

Even without such frameworks, you can simulate dependencies between classes in Salesforce with the stub API, for example, d.velop classes such as dvelop_docs_dev.DocumentManager.

> **Note**
> When simulating an Apex class with the stub API, you can only simulate non-static methods. Due to their exclusively static structure, classes such as dvelop_docs_dev.OpenAPI cannot be simulated effectively using the stub API or any frameworks.

The following shows you an example using the stub API with no framework together with d.velop classes.

Using the stub API with d.velop classes

In this example, unit tests are to be written for the class **MyConsumerClass**. This class uses the **DocumentManager** class **getDocument(documentId)** method and is therefore dependent on the method. Here, a unit test without mocking would have to ensure that all the required test data and HTTP mocks are generated and defined.

```
public with sharing class MyConsumerClass {
    private final dvelop_docs_dev.DocumentManager docManager;

    // The document manager is injected into the constructor as a dependency
    public MyConsumerClass(dvelop_docs_dev.DocumentManager docManager) {
        this.docManager = docManager;
    }

    public dvelop_docs_dev.Document doSomeMagic(String documentId) {
        if (String.isBlank(documentId)) {
            throw new System.IllegalArgumentException('Document ID must not
be blank');
        }

        return docManager.getDocument(documentId);
    }
}
```

You can use the stub API to replace this dependency with a stub. To do so, you create a helper class that uses the **System.StubProvider** interface. You can find more detailed information and instructions for using the interface in the Apex guide.

```
@IsTest
public class MyStubProvider implements System.StubProvider {
    // Implement the handleMethodCall method
    public Object handleMethodCall(
        Object stubbedObject,
        String stubbedMethodName,
        Type returnType,
        List<Type> listOfParamTypes,
        List<String> listOfParamNames,
        List<Object> listOfArgs
    ) {
        // Check the name of the called method on the mocked instance
(dvelop_docs_dev.DocumentManager) and return a mocked value
```

```
        switch on stubbedMethodName {
            when 'getDocument' {
                dvelop_docs_dev.Document mockedDocument = new
dvelop_docs_dev.Document();
                mockedDocument.id = '123';
                return mockedDocument;
            }
            when else {
                return null;
            }
        }
    }
}
```

At the end, the tests for the **MyConsumerClass** class can access the (**MyStubProvider**) and the mocked return values with **OpenAPI.createStub(parentType, stubProvider)**.

> **Note**
> Because the stub API only allows you to create stubs for classes in the same namespace, you must perform the call within the **OpenAPI** class in the **dvelop_docs_dev** namespace, instead of calling **Test.createStub(parentType, stubProvider)**.

```
@IsTest
private class MyTestClass {
    static dvelop_docs_dev.DocumentManager stubbedManager =
(dvelop_docs_dev.DocumentManager) dvelop_docs_dev.OpenAPI.createStub(
        dvelop_docs_dev.DocumentManager.class,
        new MyStubProvider()
    );

    static MyConsumerClass underTest = new MyConsumerClass(stubbedManager);

    @IsTest
    static void myUnitTest() {
        Test.startTest();
        dvelop_docs_dev.Document result = underTest.doSomeMagic('123');
        Test.stopTest();

        Assert.areEqual('123', result.id, 'Document ID should be 123');
    }
}
```

1.2. Apex reference

This chapter provides detailed information and instructions on Apex classes and Apex interfaces in the **dvelop_docs_dev** namespace.

1.2.1. Using the class “AsyncProcessFinisher”

You can use the class **dvelop_docs_dev.AsyncProcessFinisher** to finish an asynchronous **queueable** process and then respond to the output of the process.

The **AsyncProcessFinisher** job implements the queueable interface, which you can start using **System.enqueueJob()**.

Signature

```
global with sharing virtual class AsyncProcessFinisher implements
IServiceProcessFinisher
```

Constructors

You can create an instance of **AsyncProcessFinisher** jobs with the following constructors:

- AsyncProcessFinisher(defaultParamName)

AsyncProcessFinisher(defaultParamName)

This creates an **AsyncProcessFinisher** job with the name of the default parameter that is used to output most asynchronous processes.

Signature

```
global AsyncProcessFinisher(String defaultParamName)
```

Parameters

defaultParamName: The name of the default output parameter. This parameter is used by most asynchronous processes to transfer the output of the processes.

Data type: **String**

Methods

The class **AsyncProcessFinisher** provides the following methods:

- execute(processOutput)

execute(processOutput)

This method executes custom logic after finishing a **queueable** process. To control the executed logic, overwrite the method in your own implementation.

Signature

```
protected virtual void execute(Map<String, Object> processOutput)
```

Parameters

processOutput: A dictionary collection of output values from the previous process.

- **Data type**: **Map<string, object>**
- **Default value**: **new Map<string, object>()**

Use

The next chapter shows you some application scenarios depicting the correct use of the class.

Example

```
public class CustomFinishJob extends dvelop_docs_dev.AsyncProcessFinisher {
    private static final String DEFAULT_PARAM_NAME = 'default';

    public CustomFinishJob() {
        super(DEFAULT_PARAM_NAME);
    }

    public override void execute(Map<String, Object> processOutput) {
        System.debug(processOutput);
```

```
        Object defaultOutput = processOutput.get(DEFAULT_PARAM_NAME);
        System.debug(defaultOutput);
    }
}
```

1.2.2. Using the "AttachmentMigrationRequest" class

The **dvelop_docs_dev.AttachmentMigrationRequest** class defines the parameters with which **EmailMessage** items and files are to be migrated to a record in the DMS.

Signature

Using the "IDocumentUploadRequest" interface

```
global with sharing class AttachmentMigrationRequest implements
IDocumentUploadRequest
```

Constructors

You can create an **AttachmentMigrationRequest** using the following constructors:

- AttachmentMigrationRequest()

AttachmentMigrationRequest()

This constructor creates a **AttachmentMigrationRequest** without parameters.

Signature

```
global AttachmentMigrationRequest()
```

Properties

The **AttachmentMigrationRequest** class has the following properties:

- **relatedEntityId**: The ID of a record which items are to be migrated from.
  Data type: **String**
- **doctypeKey**: The category key to be used to save the documents in the DMS.
  Data type: **String**
- **fileTypesToProcess**: A comma-separated list of file types (file extensions) to be migrated from the record. To migrate all file types, specify *.
  - **Data type**: **String**
  - **Default value**: *
- **preserveFilesAfterTransfer**: Defines whether to save the files to Salesforce once successfully uploaded.
  - **Data type**: **Boolean**
  - **Default value**: **false**
- **transferAndRemoveEmails**: Defines whether to also migrate e-mails.
  - **Data type**: **Boolean**
  - **Default value**: **true**
- **preserveEmailsAfterTransfer**: Defines whether to save e-mails to Salesforce once successfully uploaded.
  - **Data type**: **Boolean**
  - **Default value**: **false**
- **reallocateRemainingFiles**: Defines whether to add the e-mail attachments for a record (second layer) to the original record once the e-mail has been successfully uploaded.
  - **Data type**: **Boolean**
  - **Default value**: **false**

- **documentAttributes**: A list of properties to override calculated properties (from the record and category).
  Data type: **List<**dvelop_docs_dev.DocumentAttribute**>**
- **useConfigUserForUpload**: Defines whether the process is run with the service user's credentials instead of the process for the current logged-in user.

> **Note**
> The **useConfigUserForUpload** property is obsolete and is only supported in flows. You can run the upload process with different credentials using the DocumentUploadOptions class instead.

  - **Data type**: **Boolean**
  - **Default value**: **false**

1.2.3. Using the "ContentDocumentUploadRequest" class

The **dvelop_docs_dev.ContentDocumentUploadRequest** class defines the parameters that are used to upload a **ContentDocument** or **Attachment** file from Salesforce to the DMS.

Signature

Using the "IDocumentUploadRequest" interface

```
global class ContentDocumentUploadRequest implements IDocumentUploadRequest
```

Constructors

You can create a **ContentDocumentUploadRequest** using the following constructors:

- ContentDocumentUploadRequest()

ContentDocumentUploadRequest()

This constructor creates a **ContentDocumentUploadRequest** without parameters.

Signature

```
global ContentDocumentUploadRequest()
```

Properties

The **ContentDocumentUploadRequest** class has the following properties:

- **relatedEntityId**: The ID of a record in the context of which the file is to be uploaded.
  Data type: **String**
- **contentDocumentId**: The ID of the **ContentDocument** or **Attachment** file to be uploaded.
  Data type: **String**
- **documentTypeKey**: The category key to be used to save the document in the DMS.
  Data type: **String**
- **preserveFileAfterUpload**: Defines whether to keep the file record once it has been successfully uploaded to Salesforce.
  - **Data type**: **Boolean**
  - **Default value**: **false**
- **documentAttributes**: A list of properties to override calculated properties (from the record and category).
  Data type: **List<**dvelop_docs_dev.DocumentAttribute**>**
- **useConfigUserForUpload**: Defines whether the process is run with the service user's credentials instead of the process for the current logged-in user.

> **Note**
> The **useConfigUserForUpload** property is obsolete and is only supported in flows. You can run the upload process with different credentials using the DocumentUploadOptions class instead.

- **Data type**: **Boolean**
- **Default value**: **false**

1.2.4. Using the class "Document"

The **dvelop_docs_dev.Document** class includes all the metadata of a DMS document and lets you work with documents in Apex in a structured and simple way.

Signature

```
global with sharing class Document
```

Constructors

You can create a **Document** instance with the following constructors:

- Document()

Document()

This method creates a simple **Document** instance.

Signature

```
global Document()
```

Properties

The class **Document** has the following properties:

- **id**: The unique ID of the DMS document.
  Data type: **String**
- **viewingUrl**: An absolute URL that can be used to view the document in d.velop documents.
  Data type: **String**
- **title**: The title (**filecaption**) of the document.
  Data type: **String**
- **downloadUrl**: An absolute URL that can be used to download the document.
  Data type: **String**
- **categories**: A list of the keys for the categories assigned to the DMS category of the document.
  Data type: **List<String>**
- **defaultCategory**: The first category from the **categories** list.
  Data type: **String**
- **attributes**: The list of document attributes.
  Data type: **List<**dvelop_docs_dev.DocumentAttribute**>**

Methods

The **Document** class provides the following methods:

- getDocumentAttributes()

getDocumentAttributes()

This method outputs the document attributes in generic format.

Signature

```
global Map<String, Object> getDocumentAttributes()
```

Returned entry

The document attributes in a generic dictionary. The values may be an individual text value (**String**) or a list of text values (**List<string>**).

1.2.5. Using the class "DocumentAttribute"

The class **dvelop_docs_dev.DocumentAttribute** encapsulates all the metadata and information for a DMS document attribute or property.

Signature

```
global class DocumentAttribute
```

Constructors

You can create a **DocumentAttribute** instance with the following constructors:

- DocumentAttribute()
- DocumentAttribute(key, value)

DocumentAttribute()

This constructor creates a simple **DocumentAttribute** instance.

Signature

```
global DocumentAttribute()
```

DocumentAttribute(key, value)

This constructor creates a **DocumentAttribute** instance with a key and one or more attribute values.

Signature

```
global DocumentAttribute(String key, Object value)
```

Parameters

- **key**: The key for the attribute or property.
  Data type: **String**
- **value**: The value of the attribute or property. You can transfer an individual text value or a list of text values.
  Data type: **Object**

Properties

The class **DocumentAttribute** has the following properties:

- **key**: The key for the attribute or property.
  Data type: **String**
- **values**: A list with the property values. If there are multiple values, the list contains more than one value.
  Data type: **List<String>**
- **value**: Reads out the first value from the **values** list. Defining this property automatically fills the **values** property with a list with the specified value.
  Data type: **String**

1.2.6. Using the class “DocumentDownloadResult”

The class **dvelop_docs_dev.DocumentDownloadResult** combines the output parameters of a successful download of a DMS document.

Signature

```
global with sharing class DocumentDownloadResult
```

Methods

The class **DocumentDownloadResult** provides the following methods:

- getFilename()
- getBody()

getFilename()

This method returns the name of the original file of the DMS document that was downloaded.

Signature

```
global String getFilename()
```

Returned entry

The name of the original file.

getBody()

This method returns the contents of the original file of the DMS document that was downloaded.

Signature

```
global Blob getBody()
```

Returned entry

The binary contents of the original file.

Use

The following section shows you some application scenarios depicting the use of the class.

Handling the output of a successful download

```
dvelop_docs_dev.DocumentDownloadResult downloadResult =
documentManager.downloadDocument('XH00014562');
String filename = downloadResult.getFilename();
Blob filebody = downloadResult.getBody();
```

1.2.7. Using the class “DocumentEmailOptions”

The class **dvelop_docs_dev.DocumentEmailOptions** defines the parameters that can be used to send an e-mail with Salesforce and DMS attachments.

Signature

```
global with sharing class DocumentEmailOptions implements
IDocumentEmailOptions
```

Constructors

You can create an instance of **DocumentEmailOptions** with the following constructors:

DocumentEmailOptions()

This constructor creates a **DocumentEmailOptions** instance without custom values.

Signature

```
global DocumentEmailOptions()
```

DocumentEmailOptions(subject, body, recipients)

This constructor creates an instance of **DocumentEmailOptions** with a subject, content and recipients.

Signature

```
global DocumentEmailOptions(String subject, String body, List<String>
recipients)
```

Parameters

- **subject**: The subject of the e-mail.
  Data type: **String**
- **body**: The text content of the e-mail.
  Data type: **String**
- **recipients**: A list of e-mail addresses for the recipients.
  Data type: **List<String>**

Methods

The class **DocumentEmailOptions** provides the following methods:

setSubject(subject)

This method adds the relevant value as the subject of the e-mail.

Signature

```
global void setSubject(String subject)
```

Parameters

subject: The subject of the e-mail.

Data type: **String**

setBody(body)
This method adds the relevant value as the content of the e-mail.

Signature

```
global void setBody(String body)
```

Parameters
body: The content of the e-mail.

Data type: **String**

addRecipient(recipient)
This method adds another e-mail address to the existing list of recipients.

Signature

```
global void addRecipient(String recipient)
```

Parameters
recipient: The e-mail address of the new recipient.

Data type: **String**

addRecipients(recipients)
This method adds multiple additional e-mail addresses to the existing list of recipients.

Signature

```
global void addRecipients(List<String> recipients)
```

Parameters
recipients: A list with the e-mail addresses of the new recipients.

Data type: **List<String>**

addCcRecipient(ccRecipient)
This method adds another e-mail address to the existing list of CC recipients.

Signature

```
global void addCcRecipient(String ccRecipient)
```

Parameters
ccRecipient: The e-mail address of the new CC recipient.

Data type: **String**

addCcRecipients(ccRecipients)
This method adds multiple additional e-mail addresses to the existing list of CC recipients.

Signature

```
global void addCcRecipients(List<String> ccRecipients)
```

Parameters
ccRecipients: A list with the e-mail addresses of the new CC recipients.

Data type: **List<String>**

addBccRecipient(bccRecipient)
This method adds another e-mail address to the existing list of BCC recipients.

Signature

```
global void addBccRecipient(String bccRecipient)
```

Parameters
bccRecipient: The e-mail address of the new BCC recipient.

Data type: **String**

addBccRecipients(bccRecipients)
This method adds multiple additional e-mail addresses to the existing list of BCC recipients.

Signature

```
global void addBccRecipients(List<String> bccRecipients)
```

Parameters
bccRecipients: A list with the e-mail addresses of the new BCC recipients.

Data type: **List<String>**

addContentDocumentId(contentDocumentId)
This method adds a **ContentDocument** item as an e-mail attachment based on its ID.

Signature

```
global void addContentDocumentId(Id contentDocumentId)
```

Parameters
contentDocumentId: The ID of the **ContentDocument** item to be sent as the e-mail attachment.

Data type: **Id**

addContentDocumentIds(contentDocumentIds)
This method adds multiple **ContentDocument** items as e-mail attachments based on their IDs.

Signature

```
global void addContentDocumentIds(List<Id> contentDocumentIds)
```

Parameters
contentDocumentIds: A list of IDs of the **ContentDocument** items to be sent as e-mail attachments.

Data type: **List<ID>**

addDmsDocumentId(dmsDocumentId)
This method adds a DMS document as an e-mail attachment based on its ID.

Signature

```
global void addDmsDocumentId(String dmsDocumentId)
```

Parameters

dmsDocumentId: The ID of the DMS document to be sent as the e-mail attachment.

Data type: **String**

addDmsDocumentIds(dmsDocumentIds)

This method adds multiple DMS documents as e-mail attachments based on their IDs.

Signature

```
global void addDmsDocumentIds(List<String> dmsDocumentIds)
```

Parameters

dmsDocumentIds: A list of IDs of the DMS documents to be sent as e-mail attachments.

Data type: **List<String>**

setRelatedRecordId(recordId)

This method defines the ID of the record you want to link to the e-mail to be sent.

Signature

```
global void setRelatedRecordId(String recordId)
```

Parameters

recordId: The ID of the Salesforce record linked to the e-mail.

Data type: **String**

setEmailTemplateId(templateId)

This method defines the e-mail template for the e-mail to be sent.

Signature

```
global void setEmailTemplateId(String templateId)
```

Parameters

templateId: The ID of the e-mail template used.

Data type: **String**

setUseSignature(useSignature)

This method determines whether the signature of the current Salesforce user is to be added to the content of the e-mail.

Signature

```
global void setUseSignature(Boolean useSignature)
```

Parameters

useSignature: Determines whether the signature is added.

Data type: **Boolean**

Use

The following section shows you some application scenarios depicting the use of the class.

Sending an e-mail without attachments

```
dvelop_docs_dev.DocumentEmailOptions emailOptions = new
dvelop_docs_dev.DocumentEmailOptions();
emailOptions.setSubject('Urgent d.velop documents for Salesforce Update');
emailOptions.setBody('Hey, please review the release notes of the latest
version of d.velop documents for Salesforce. Regards.');
emailOptions.addRecipient('astro@salesforce.com');
emailOptions.addCcRecipient('codey@salesforce.com');
emailOptions.addBccRecipient('einstein@salesforce.com');
```

Sending an e-mail with DMS and Salesforce attachments

```
dvelop_docs_dev.DocumentEmailOptions emailOptions = new
dvelop_docs_dev.DocumentEmailOptions();
emailOptions.setSubject('Urgent d.velop documents for Salesforce Update');
emailOptions.setBody('Hey, please review the release notes of the latest
version of d.velop documents for Salesforce. Regards.');
emailOptions.addRecipient('astro@salesforce.com');
emailOptions.addDmsDocumentId('XH00014562');
emailOptions.addContentDocumentId('069AP0000006hn3YAA');
```

1.2.8. Using the class “DocumentManager”

The **dvelop_docs_dev.DocumentManager** class provides a number of different functions for working with and managing documents from the connected document management system (DMS) using your custom Apex code.

Signature

Using the “SubscriberInterface” class

```
global with sharing class DocumentManager extends SubscriberInterface
```

Constructors

You can create an instance of **DocumentManager** with the following constructors:

- ???TITLE???

DocumentManager()

This constructor creates a **DocumentManager** instance with default values.

Signature

```
global DocumentManager()
```

Methods

The class **DocumentManager** provides the following methods:

- downloadDocument(documentId)
- downloadDocuments(documentIds, finisher, proceedOnFailure)
- downloadDocumentToRecord(recordId, documentId)
- searchDocuments(options)
- search(options)

- searchDocumentsWithIds(searchAttributeKey, documentIds)
- getDocument(documentId)
- updateDocument(documentId, updatedCategory, updatedAttributes)
- deleteDocument(documentId, ?reason)
- sendEmail(options)
- createFolder(category, attributes)
- shareDocuments(downloadUrls)
- shareDocuments(options)
- changeDocumentStatus(documentId, newStatus, ?editorId, ?alterationText)

> **Note**
> For the superclass methods, take a look at Using the "SubscriberInterface" class.

downloadDocument(documentId)

This method downloads a DMS document with the transferred ID and returns the class **DocumentDownloadResult** with the name and content of the document file as the result.

> **Note**
> You can find a detailed guide to downloading documents here.

Signature

```
global DocumentDownloadResult downloadDocument(String documentId)
```

Parameters

documentId: The ID of the DMS document to be downloaded to Salesforce.

Data type: **String**

Returned entry

An instance of the class dvelop_docs_dev.DocumentDownloadResult containing the file name and the content of the document file.

downloadDocuments(documentIds, finisher, proceedOnFailure)

This method asynchronously downloads multiple DMS documents and passes the results to the transferred **AsyncProcessFinisher** job.

> **Note**
> You can find a detailed guide to downloading documents here.

> **Warning**
> This method starts an asynchronous **queueable** process with **System.enqueueJob()**. Note the limits for queueable jobs in the context in which you call the method.

Signature

```
global void downloadDocuments(List<String> documentIds,
AsyncProcessFinisher finisher, Boolean proceedOnFailure)
```

Parameters

- **documentIds**: A list of the IDs of the DMS documents to be downloaded to Salesforce.
  Data type: **List<String>**
- **finisher**: An instance of an **AsyncProcessFinisher** job. Once completed, this job is added to the Download queue and receives a list from the class **DocumentDownloadResult** as an output for its default parameter.
  Data type: dvelop_docs_dev.AsyncProcessFinisher
- **proceedOnFailure**: Defines whether the process is continued if an error occurs while downloading a document.
  - **Data type**: **Boolean**
  - **Default value**: **false**

Returned entry (to the "finisher" job)

A list with instances of the class dvelop_docs_dev.DocumentDownloadResult that contains the results of the individual downloads.

downloadDocumentToRecord(recordId, documentId)

This method downloads a DMS document with the transferred ID and saves the document as a file in a Salesforce record.

> **Note**
> You can find a detailed guide to downloading documents here.

> **Warning**
> This method executes DML. Note the limits for DML statements in the context in which you call the method. HTTP callouts are no longer possible directly after a DML operation.

Signature

```
global Id downloadDocumentToRecord(Id recordId, String documentId)
```

Parameters

- **recordId**: The ID of the data record to which the downloaded DMS document is to be attached.
  Data type: **Id**
- **documentId**: The ID of the DMS document to be downloaded to Salesforce.
  Data type: **String**

Returned entry

The ID of the **ContentDocument** record that was created for the downloaded DMS document on the related data record.

searchDocuments(options)

This method performs a search for documents in the DMS. The parameters of the search are defined by the transferred instance of **DocumentSearchOptions**.

> **Warning**
> The method **searchDocuments(options)** is obsolete but is still supported. However, we recommend switching to the new search(options) method to ensure that you can utilize all the benefits.

Signature

```
global List<Map<String, String>> searchDocuments(DocumentSearchOptions
options)
```

Parameters

options: An instance of a choice of configured search options that are used as the basis for determining the parameters for the search.

Data type: dvelop_docs_dev.DocumentSearchOptions

Returned entry

A list of the documents found in generic dictionary format (**Map**).

search(options)

This method performs a search for documents in the DMS. The parameters of the search are defined by the transferred instance of **DocumentSearchOptions**.

> **Note**
> You can find a detailed guide to searching for DMS documents with complex search parameters here.

Signature

```
global List<Document> search(DocumentSearchOptions options)
```

Parameters

options: An instance of a choice of configured search options that are used as the basis for determining the parameters for the search.

Data type: dvelop_docs_dev.DocumentSearchOptions

Returned entry

A dvelop_docs_dev.Document list for the DMS documents found.

searchDocumentsWithIds(searchAttributeKey, documentIds)

This method performs a search for documents with specific IDs in the DMS.

> **Note**
> You can find a detailed guide to searching for DMS documents based on their IDs here.

Signature

```
global List<Document> searchDocumentsWithIds(String searchAttributeKey,
List<String> documentIds)
```

Parameters

- **searchAttributeKey**: The source property key that is assigned to the document ID in the connected DMS.
  Data type: **String**
- **documentIds:** The list of IDs for the searched documents.
  Data type: **List<String>**

Returned entry
A dvelop_docs_dev.Document list for the DMS documents found.

getDocument(documentId)
This method retrieves a document with the specified ID from the DMS and provides the result as an instance of the **Document** class.

> **Note**
> You can find a detailed guide to retrieving a specific DMS document here.

Signature

```
global Document getDocument(String documentId)
```

Parameters
documentId: The ID of the DMS document whose properties, links and categories are identified.

Data type: **String**

Returned entry
An instance of the class dvelop_docs_dev.Document for the DMS document that was found using the ID on the connected DMS.

updateDocument(documentId, updatedCategory, updatedAttributes)
This method updates the properties and optionally the category of the DMS document with the given ID.

> **Note**
> You can find a detailed guide to updating a specific DMS document here.

Signature

```
global void updateDocument(String documentId, String updatedCategory,
List<DocumentAttribute> updatedAttributes)
```

Parameters

- **documentId**: The ID of the DMS document to be updated.
  Data type: **String**
- **updatedCategory**: The key of the new category. If the category is not to be updated, **zero** can be transferred instead.
  Data type: **String**
- **updatedAttributes**: A list of properties to be updated.
  Data type: **List<**dvelop_docs_dev.DocumentAttribute**>**

deleteDocument(documentId, ?reason)
This method deletes a document or a folder with the given ID from the DMS.

> **Note**
> You can find a detailed guide to deleting a specific DMS item here.

Signature

```
global void deleteDocument(String documentId)
```

```
global void deleteDocument(String documentId, String reason)
```

Parameters

- **documentId**: The ID of the DMS item to be deleted.
  Data type: **String**
- **reason**: The reason for deleting the item. The character string must be between three and 80 characters long.
  Data type: **String**
  Default value: "Deleted from d.velop documents for Salesforce."

sendEmail(options)

This method sends an e-mail with attachments from Salesforce (**ContentDocument**) and the connected DMS (DMS documents). The details of the e-mail are defined by the transferred instance of **DocumentEmailOptions**.

> **Note**
> You can find a detailed guide to sending e-mails with Salesforce and DMS attachments here.

Signature

```
global void sendEmail(DocumentEmailOptions options)
```

Parameters

options: An instance of a choice of configured e-mail options that are used as the basis for determining the required e-mail details and attachments.

Data type: dvelop_docs_dev.DocumentEmailOptions

createFolder(category, attributes)

This method creates a folder with the specified category and the list of properties in the DMS.

> **Note**
> You can find a detailed guide to creating folders in the DMS here.

Signature

```
global String createFolder(String category, List<DocumentAttribute>
attributes)
```

Parameters

- **category**: The key of the category that the folder is to have.
  Data type: **String**
- **attributes:** A list of properties to be defined for the folder.
  Data type: **List<**dvelop_docs_dev.DocumentAttribute**>**

Returned entry

The ID of the created folder in the DMS.

shareDocuments(downloadUrls)

This method creates temporary URLs for sharing DMS documents with third parties.

> **Note**
> You can find a detailed guide to sharing DMS documents with third parties here.

> **Warning**
> The method is unavailable in SharePoint environments.

Signature

```
global Map<String, String> shareDocuments(Set<String> downloadUrls)
```

Parameters

downloadUrls: A set of URLs for downloading the documents to be shared.

Data type: **Set<string>**

Returned entry

A generic dictionary (map) with the transferred download URLs and the associated sharing URLs.

shareDocuments(options)

Creates temporary URLs with user-defined settings for sharing DMS documents with third parties.

Signature

```
global Map<String, String> shareDocuments(DocumentShareOptions options)
```

Parameters

options: A collection of settings that apply to the created sharing URLs.

Data type: dvelop_docs_dev.DocumentShareOptions

Returned entry

A generic dictionary (map) with the transferred download URLs and the associated sharing URLs, based on the transferred settings.

changeDocumentStatus(documentId, newStatus, ?editorId, ?alterationText)

Updates the status and optionally the editor of a DMS document.

Signature

```
global void changeDocumentStatus(String documentId, DocumentStatus
newStatus)
```

```
global void changeDocumentStatus(String documentId, DocumentStatus
newStatus, String editorId, String alterationText)
```

Parameters

- **documentId**: The ID of the DMS document to be updated.
  Data type: **String**
- **newStatus**: The new status of the document.
  Data type: dvelop_docs_dev.DocumentStatus

- **editorId**: The ID of the identity provider user to be entered for document editing.
  Data type: **String**
- **alterationText**: A short text that explains the reason for or describes the change.
  Data type: **String**

1.2.9. Using the class “DocumentSearchOptions”

The class **dvelop_docs_dev.DocumentSearchOptions** defines the parameters used to retrieve documents from the connected document management system (DMS).

Signature

```
global with sharing class DocumentSearchOptions implements
IDocumentSearchOptions
```

Constructors

You can create an instance of **DocumentSearchOptions** with the following constructors:

- DocumentSearchOptions()

DocumentSearchOptions()

This constructor creates a **DocumentSearchOptions** instance with default values.

Signature

```
global DocumentSearchOptions()
```

Methods

The class **DocumentSearchOptions** provides the following methods:

- buildOptions()
- useRecordContext(recordId)
- useObjectContext(objectApiName)
- useObjectContext(objectApiName, recordTypeId)
- addSearchAttribute(attributeKey, value)
- addSearchAttribute(attributeKey, values)
- addSearchAttributes(searchAttributes)
- ignoreSearchAttributes()
- addSearchCategory(categoryKey)
- addSearchCategories(categoryKeys)
- ignoreSearchCategories()
- hideFolders()
- showFolders()
- sortByAttribute(attributeKey)
- sortAscending()
- sortDescending()
- setSearchText(searchText)
- setMaxAgeInDays(maxAgeInDays)
- setPageIndex(pageIndex)
- setPageSize(pageSize)

buildOptions()

This method creates an instance of the class **DocumentSearchOptions.Builder**.

> **Note**
> You can find a detailed guide to using the **DocumentSearchOptions.Builder** class here

Signature

```
global static DocumentSearchOptions.Builder buildOptions()
```

Returned entry
A new instance of dvelop_docs_dev.DocumentSearchOptions.Builder.

useRecordContext(recordId)
This method changes the context of the search for the record with the transferred ID.

All the settings, such as mappings, document type assignments, property assignments and sorting settings, are then determined based on the object types and the record type of the transferred record.

> **Note**
> Using this method overrides the prior use of the following methods:
>
> - **useObjectContext(objectApiName)**
> - **useObjectContext(objectApiName, recordTypeId)**
>
> You should only ever use one of the available methods.

Signature

```
global void useRecordContext(Id recordId)
```

Parameters
recordId: The ID of the record to be used as the context for the search.

Data type: **Id**

useObjectContext(objectApiName)
This method changes the context of the search to the Salesforce object with the transferred API name.

All the settings, such as mappings, document type assignments, property assignments and sorting settings, are then determined based on the transferred item and the standard record type **Master**.

> **Note**
> Using this method overrides the prior use of the following methods:
>
> - **useObjectContext(objectApiName)**
> - **useObjectContext(objectApiName, recordTypeId)**
>
> You should only ever use one of the available methods.

Signature

```
global void useObjectContext(String objectApiName)
```

Parameters
objectApiName: The API name of the Salesforce object to be used as the context for the search.

Data type: **String**

useObjectContext(objectApiName, recordTypeId)

This method changes the context of the search to the Salesforce object with the transferred API name and the record types with the transferred ID.

All the settings, such as mappings, document type assignments, property assignments and sorting settings, are then determined based on the transferred object and the transferred record type.

> **Note**
> Using this method overrides the prior use of the following methods:
>
> - **useObjectContext(objectApiName)**
> - **useObjectContext(objectApiName, recordTypeId)**
>
> You should only ever use one of the available methods.

Signature

```
global void useObjectContext(String objectApiName, String recordTypeId)
```

Parameters

- **objectApiName**: The API name of the Salesforce object to be used as the context for the search.
  Data type: **String**
- **recordTypeId**: The ID of the record type to be used as the context for the search.
  Data type: **String**

addSearchAttribute(attributeKey, value)

This method adds a user-defined search attribute with a corresponding value to the existing search attributes.

You can define any number of properties and values for the search, which limit the result of the search to documents with matching properties.

> **Note**
> Using this method overrides the following global settings:
>
> - **Mapping**
> - **Properties > Mappings**

Signature

```
global void addSearchAttribute(String attributeKey, String value)
```

Parameters

- **attributeKey**: The key of the property to be used for the search. You can find a full overview of the available keys under **Properties > Mappings** in your d.velop documents configuration.
  Data type: **String**
- **value**: The value of the property that is synchronized with the properties of the documents.
  Data type: **String**

addSearchAttribute(attributeKey, values)

This method adds a user-defined search attribute with a list of corresponding values to the existing search attributes.

You can define any number of properties and values for the search, which limit the result of the search to documents with matching properties.

> **Note**
> Using this method overrides the following global settings:
>
> - **Mapping**
> - **Properties > Mappings**

Signature

```
global void addSearchAttribute(String attributeKey, List<String> values)
```

Parameters

- **attributeKey**: The key of the property to be used for the search. You can find a full overview of the available keys under **Properties > Mappings** in your d.velop documents configuration.
  Data type: **String**
- **values**: A list of possible values for the property that is synchronized with the properties of the documents. The values have an **OR** link. If one value matches, a document is found.
  Data type: **List<String>**

addSearchAttributes(searchAttributes)

This method adds multiple search attributes and values for the search to the existing search attributes.

You can define any number of properties and values for the search, which limit the result of the search to documents with matching properties.

> **Note**
> Using this method overrides the following global settings:
>
> - **Mapping**
> - **Properties > Mappings**

Signature

```
global void addSearchAttributes(Map<String, String> searchAttributes)
```

Parameters

searchAttributes: A collection of key-value pairs for search properties.

Data type: **Map<string, string>**

ignoreSearchAttributes()

This method ignores all the global search attributes and all the user-defined search attributes added previously using the **addSearchAttribute** or **addSearchAttributes** methods.

> **Note**
> Using this method overrides the following global settings:
>
> - **Mapping**
> - **Properties > Mappings**
>
> Using this method nullifies the effect of the following methods, regardless of the order of use:
>
> - **addSearchAttribute**
> - **addSearchAttributes**

Signature

```
global void ignoreSearchAttributes()
```

addSearchCategory(categoryKey)

This method adds a user-defined search category to the existing search categories.

You can define any number of search categories, which limit the result of the search to documents with matching categories.

> **Note**
> Using this method overrides the following global settings:
>
> - **Document Types > Assignment**
> - **Document Types > General Settings > Search only selected document types**

Signature

```
global void addSearchCategory(String categoryKey)
```

Parameters

categoryKey: The key of the category to be used for the search. You can find a full overview of the available keys under **Document Types > Assignment** in your d.velop documents configuration.

Data type: **String**

addSearchCategories(categoryKeys)

This method adds multiple search categories to the existing search categories.

You can define any number of search categories, which limit the result of the search to documents with matching categories.

> **Note**
> Using this method overrides the following global settings:
>
> - **Document Types > Assignment**
> - **Document Types > General Settings > Search only selected document types**

Signature

```
global void addSearchCategories(List<String> categoryKeys)
```

Parameters

categoryKeys: A list of keys for search categories.

Data type: **List<String>**

ignoreSearchCategories()

This method ignores all the global search categories and all the user-defined search categories added previously using the **addSearchCategory** or **addSearchCategories** methods.

> **Note**
>
> Using this method overrides the following global settings:
>
> - **Document Types > Assignment**
> - **Document Types > General Settings > Search only selected document types**
>
> Using this method nullifies the effect of the following methods, regardless of the order of use:
>
> - **addSearchCategory**
> - **addSearchCategories**

Signature

```
global void ignoreSearchCategories()
```

hideFolders()

This method hides folders in the search results.

> **Note**
>
> Using this method overrides the following global settings:
>
> - **Document List > General Settings > Show Folders**
>
> This method has an effect only if folders can actually be identified with a search. The following prerequisites apply:
>
> - **Document Types > General Settings > Search only selected document types** is deactivated
> - **Document Types > General Settings > Search only selected document types** is activated and search categories added with **Document Types > Assignment** contain folders
> - **Document Types > General Settings > Search only selected document types** is activated and search categories added with the method **addSearchCategory** or **addSearchCategories** contain folders

Signature

```
global void hideFolders()
```

showFolders()

This method shows folders in the search results.

> **Note**
> Calling this method overwrites the following global settings:
>
> - **Document List > General Settings > Show Folders**
>
> This method has an effect only if folders can actually be identified with a search. The following prerequisites apply:
>
> - **Document Types > General Settings > Search only selected document types** is deactivated
> - **Document Types > General Settings > Search only selected document types** is activated and search categories added with **Document Types > Assignment** contain folders
> - **Document Types > General Settings > Search only selected document types** is activated and search categories added with the method **addSearchCategory** or **addSearchCategories** contain folders

Signature

```
global void showFolders()
```

sortByAttribute(attributeKey)

This method adds the property with the transferred key as a sorting attribute. The result of the search is then sorted based on the values of the specified property. The sorting is in ascending order by default.

Signature

```
global void sortByAttribute(String attributeKey)
```

Parameters

attributeKey: The key of the property to be used to sort the search results.

Data type: **String**

sortAscending()

This method sorts the search results in ascending order based on the sorting attribute defined using the **sortByAttribute** method.

> **Note**
> This method only has an effect if you have also defined a sorting attribute using the **sortByAttribute** method.

Signature

```
global void sortAscending()
```

sortDescending()

This method sorts the search results in descending order based on the sorting attribute defined using the **sortByAttribute** method.

> **Note**
> This method has an effect only if you have also defined a sorting property using the method **sortByAttribute**.

Signature

```
global void sortDescending()
```

setSearchText(searchText)

This method searches for the transferred search term in all the documents and via a full-text search and restricts the results to matching documents.

Signature

```
global void setSearchText(String searchText)
```

Parameters

searchText: The search term for the full-text search.

Data type: **String**

setMaxAgeInDays(maxAgeInDays)

This method sets a maximum age (in days) for the search. Any documents that are older than the specified number of days are not included in the search.

Signature

```
global void setMaxAgeInDays(Integer maxAgeInDays)
```

Parameters

maxAgeInDays: The maximum age of the documents in days.

Data type: **Integer**

setPageIndex(pageIndex)

This method sets the page number of the documents in the search result. The number of pages available is defined by the number of all the available documents and the page size defined using the **setPageSize** method.

Signature

```
global void setPageIndex(Integer pageIndex)
```

Parameters

pageIndex: The number/index of the page. The starting index is one.

- **Data type**: **Integer**
- **Default value**: 1

setPageSize(pageSize)

This method sets the page size and thus determines the maximum number of documents in the search result.

Signature

```
global void setPageSize(Integer pageSize)
```

Parameters

pageSize: The maximum number of documents per page.

- **Data type**: **Integer**

- **Default value**: 100

Use

The following section shows you some application scenarios depicting the correct use of the class.

Identifying all the documents for a data record (with global settings)

```
dvelop_docs_dev.DocumentSearchOptions searchOptions = new
dvelop_docs_dev.DocumentSearchOptions();
searchOptions.useRecordContext('001AP000003aVJhYAM');
```

Identifying all the documents for accounts (without global settings)

```
dvelop_docs_dev.DocumentSearchOptions searchOptions = new
dvelop_docs_dev.DocumentSearchOptions();
searchOptions.useObjectContext('Account');
searchOptions.ignoreSearchAttributes();
searchOptions.ignoreSearchCategories();
```

Identifying all the documents for accounts with a matching "accountnumber" property, "Schriftverkehr Kunde" (customer correspondence) or "Unbestimmte Dokumentart" (unspecified document type) document type, without folders

```
dvelop_docs_dev.DocumentSearchOptions searchOptions = new
dvelop_docs_dev.DocumentSearchOptions();
searchOptions.useObjectContext('Account');
searchOptions.addSearchAttribute('accountnumber', '001AP000003aVJhYAM');
searchOptions.addSearchCategory('Schriftverkehr_Kunde');
searchOptions.addSearchCategory('Unbestimmte_Dokumentart');
searchOptions.hideFolders();
```

Complex use case

```
dvelop_docs_dev.DocumentSearchOptions searchOptions = new
dvelop_docs_dev.DocumentSearchOptions();
searchOptions.useRecordContext('001AP000003aVJhYAM');

searchOptions.addSearchCategories(new List<String>{
'Schriftverkehr_Kunde','Eingangsrechnung', 'Unbestimmte_Dokumentart' });
searchOptions.addSearchAttributes(
    new Map<String, String>{
        'accountnumber', '001AP000003aVJhYAM',
        'objecttitle' => 'Testaccount',
        'published' => 'Ja'
    }
);

searchOptions.sortByAttribute('createddate');
searchOptions.sortAscending();

searchOptions.setSearchText('Produkt');
searchOptions.setMaxAgeInDays(1);
searchOptions.setPageIndex(1);
searchOptions.setPageSize(500);
```

1.2.10. Using the "DocumentSearchOptions.Builder" class

The **dvelop_docs_dev.DocumentSearchOptions.Builder** class makes it easier to create search parameters by implementing chainable methods.

Basics

The builder design pattern is a software engineering model that is frequently used to create complex data types using simple, compact commands and methods. The builder lets you concatenate method calls to create your product (that is, an instance of the complex data type) in a single statement, among other things.

The **DocumentSearchOptions.Builder** class implements the builder design pattern. You can create the class using the superclass **buildOptions()** method. This builder can configure a **DocumentSearchOptions** instance using any number of chainable methods.

You then obtain the result ("product") of the builder through a call of **getResult()**.

Example

To create a **DocumentSearchOptions** instance using the builder, follow the example below:

```
dvelop_docs_dev.DocumentSearchOptions searchOptions =
    dvelop_docs_dev.DocumentSearchOptions.buildOptions()    // Create an
instance of the builder
    .withRecordContext('001AP000003aVJhYAM')                // Configure
options - here: record context
    ...                                                     // Configure
further options if necessary
    .getResult();                                           // Get
the result - A completely configured instance of
dvelop_docs_dev.DocumentSearchOptions
```

Methods

The **DocumentSearchOptions.Builder** class provides the following methods as corresponding counterparts to base class methods:

Method (builder)	Method(s) (base class)
withRecordContext(recordId)	useRecordContext(recordId)
withObjectContext(objectApiName)	useObjectContext(objectApiName)
withObjectContext(objectApiName, recordTypeId)	useObjectContext(objectApiName, recordTypeId)
withSearchAttribute(attributeKey, value)	addSearchAttribute(attributeKey, value)
withSearchAttribute(attributeKey, values)	addSearchAttribute(attributeKey, values)
withoutSearchAttributes()	ignoreSearchAttributes()
withSearchCategory(categoryKey)	addSearchCategory(categoryKey)
withSearchCategories(categoryKeys)	addSearchCategories(categoryKeys)
withoutSearchCategories()	ignoreSearchCategories()
withFolders()	showFolders()
withoutFolders()	hideFolders()
sortedAscendingBy(sortAttribute)	sortByAttribute(attributeKey)
	sortAscending()
sortedDescendingBy(sortAttribute)	sortByAttribute(attributeKey)
	sortDescending()
withSearchText(searchText)	setSearchText(searchText)
withMaxAgeInDays(maxAgeInDays)	setMaxAgeInDays(maxAgeInDays)
withPageIndex(pageIndex)	setPageIndex(pageIndex)
withPageSize(pageSize)	setPageSize(pageSize)

Use

You can find out how to use the builder to implement the application scenarios from DocumentSearchOptions > Use here.

Identifying all the documents for a data record (with global settings)

```
dvelop_docs_dev.DocumentSearchOptions searchOptions =
    dvelop_docs_dev.DocumentSearchOptions.buildOptions()
    .withRecordContext('001AP000003aVJhYAM')
    .getResult();
```

Identifying all the documents for accounts (without global settings)

```
dvelop_docs_dev.DocumentSearchOptions searchOptions =
    dvelop_docs_dev.DocumentSearchOptions.buildOptions()
    .withObjectContext('Account')
    .withoutSearchAttributes()
    .withoutSearchCategories()
    .getResult();
```

Identifying all the documents for accounts with a matching "accountnumber" property, "Schriftverkehr Kunde" (customer correspondence) or "Unbestimmte Dokumentart" (unspecified document type) document type, without folders

```
dvelop_docs_dev.DocumentSearchOptions searchOptions =
    dvelop_docs_dev.DocumentSearchOptions.buildOptions()
    .withObjectContext('Account')
    .withSearchAttribute('accountnumber', '001AP000003aVJhYAM')
    .withSearchCategory('Schriftverkehr_Kunde')
    .withSearchCategory('Unbestimmte_Dokumentart')
    .withoutFolders()
    .getResult();
```

Complex use case

```
dvelop_docs_dev.DocumentSearchOptions searchOptions =
dvelop_docs_dev.DocumentSearchOptions
    .buildOptions()
    .withRecordContext('001AP000003aVJhYAM')
    .withSearchCategories(new List{ 'Schriftverkehr_Kunde',
'Eingangsrechnung', 'Unbestimmte_Dokumentart' })
    .withSearchAttribute('accountnumber', '001AP000003aVJhYAM')
    .withSearchAttribute('objecttitle', 'Testaccount')
    .withSearchAttribute('published', 'Ja')
    .sortedAscendingBy('createddate')
    .withSearchText('Produkt')
    .withMaxAgeInDays(1)
    .withPageIndex(1)
    .withPageSize(500)
    .getResult();
```

1.2.11. Using the "DocumentShareOptions.Builder" class

The class **dvelop_docs_dev.DocumentShareOptions** bundles together the parameters for sharing DMS documents and settings for the validity of the created URLs.

Signature

```
global with sharing class DocumentShareOptions
```

Constructors

The class **DocumentShareOptions** has the following constructors:

- DocumentShareOptions()

DocumentShareOptions()
Creates a new **DocumentShareOptions** object.

Signature

```
global DocumentShareOptions()
```

Methods
The class **DocumentShareOptions** offers the following methods:

- addEntry(downloadUrl, validUntil, allowMultiUse)

addEntry(downloadUrl, validUntil, allowMultiUse)
Adds a new entry for sharing. The entry contains a URL from which the document can be downloaded, a timestamp for the validity and an option for multiple use.

Signature

```
global void addEntry(String downloadUrl, Datetime validUntil, Boolean
allowMultiUse)
```

Parameters

- **downloadUrl**: A URL from which the document to be shared can be downloaded.
  Data type: **String**
- **validUntil**: A timestamp that defines the maximum validity of the created URL. The stamp must be in the future and must not be more than 180 days.
  Data type: **Datetime**
- **allowMultiUse**: Specifies whether the created sharing URL is allowed to be used multiple times.
  Data type: **Boolean**

1.2.12. Using the “DocumentStatus” enum
The enum **dvelop_docs_dev.DocumentStatus** represents the various states that a DMS document can have.

Values

- **PROCESSING**: The document is currently being processed.
- **VERIFICATION**: The document is currently being verified.
- **RELEASE**: The document has been published and can be viewed.

1.2.13. Using the class “DocumentUploader”
Use different functions in the **dvelop_docs_dev.DocumentUploader** class to upload Salesforce files (**ContentDocument**, **Attachment** and **EmailMessage**) items to the connected document management system (DMS) using your custom Apex code.

Signature
Using the “SubscriberInterface” class

```
global with sharing class DocumentUploader extends SubscriberInterface
```

Constructors
You can create an instance of **DocumentUploader** with the following constructors:

- DocumentUploader()

DocumentUploader()
This constructor creates a **DocumentUploader** instance with default values.

Signature

```
global DocumentUploader()
```

Methods
The class **DocumentUploader** provides the following methods:

- getAttributesFromConfig(recordId, doctypeKey, filename)
- startUpload(requests, ?options)

> **Note**
> You can find more information on the superclass methods at Using the "SubscriberInterface" class.

getAttributesFromConfig(recordId, doctypeKey, filename)
Use this method to calculate the properties for a document that you want to upload based on the configured global settings. The settings are defined by the object type for the transferred record ID and the document type.

Signature

```
global List<DocumentAttribute> getAttributesFromConfig(Id recordId, String
doctypeKey, String filename)
```

Parameters

- **recordId**: The record ID. The ID determines the calculated properties and their source (record) using the global settings in combination with the document type.
  Data type: **Id**
- **doctypeKey**: The document type key, which predefines the calculated properties based on the global settings.
  Data type: **String**
- **filename**: An optional file name that is used as the source value for object mapping of the **File name at run time** type. If the file name is irrelevant, **zero** is transferred.
  Data type: **String**

startUpload(requests, ?options)
Use this method to upload different records (for example, **ContentDocument**, **Attachment** or **EmailMessage**) items to d.velop documents. Define the upload parameters using different types of upload requests.

> **Warning**
> This method starts an asynchronous **queueable** process with **System.enqueueJob()**. Note the limits for queueable jobs in the context in which you use the method.

Signature

```
global Id startUpload(List<IDocumentUploadRequest> requests)
```

```
global Id startUpload(List<IDocumentUploadRequest> requests,
DocumentUploadOptions options)
```

Parameters

- **requests**: A list of upload requests that define the upload frame parameters. Depending on the type of request, either single or multiple files are uploaded.
  Data type: **List<dvelop_docs_dev.IDocumentUploadRequest>**
- **options**: An optional collection of settings that can control the behavior of the upload process.
  Data type: dvelop_docs_dev.DocumentUploadOptions

Returned entry

The ID of the process's **queueable** job.

1.2.14. Using the "DocumentUploadOptions" class

The **dvelop_docs_dev.DocumentUploadOptions** class defines the settings and options that control the behavior of the asynchronous process for uploading files.

Signature

```
global with sharing class DocumentUploadOptions
```

Constructors

You can create **DocumentUploadOptions** using the following constructors:

- DocumentUploadOptions(useConfigUser, skipValidate)

DocumentUploadOptions(useConfigUser, skipValidate)

The constructor creates **DocumentUploadOptions** with the available parameters.

Signature

```
global DocumentUploadOptions(Boolean useConfigUser, Boolean skipValidate)
```

Parameters

- **useConfigUser**: Specifies whether the process is run with the service user's credentials or those for the current logged-in user.
  Data type: **Boolean**
- **skipValidate**: Specifies whether a user session retrieved from the cache is revalidated using **Identity Provider** and possibly requested again.
  Data type: **Boolean**

1.2.15. Using the "EmailMessageUploadRequest" class

The **dvelop_docs_dev.EmailMessageUploadRequest** class defines the parameters that are used to upload an **EmailMessage** item from Salesforce to the DMS.

Signature

Using the "IDocumentUploadRequest" interface

```
global class EmailMessageUploadRequest implements IDocumentUploadRequest
```

Constructors

You can create an**EmailMessageUploadRequest** class using the following constructor:

- EmailMessageUploadRequest()

EmailMessageUploadRequest()

This constructor creates an **EmailMessageUploadRequest** without parameters.

Signature

```
global EmailMessageUploadRequest()
```

Properties

The **EmailMessageUploadRequest** class has the following properties:

- **relatedEntityId**: The ID of a record in the context of which the e-mails are to be uploaded.
  Data type: **String**
- **emailMessageIds**: A list of IDs for the **EmailMessage** items to be uploaded.
  Data type: **List<String>**
- **documentTypeKey**: The category key to be used to save the documents in the DMS.
  Data type: **String**
- **useConfigUserForUpload**: Specifies whether the process is run with the service user's credentials or those for the current logged-in user.

  > **Warning**
  > The **useConfigUserForUpload** property is obsolete and is only supported in flows. You can run the upload process with different credentials using the DocumentUploadOptions class instead.

  Data type: **Boolean**
  Default value: **false**

1.2.16. Using the "IDocumentUploadRequest" interface

Use the **dvelop_docs_dev.IDocumentUploadRequest** interface to implement different upload requests to create different application scenarios for uploading files or e-mails from Salesforce.

In combination with the **DocumentUploader** class, you can define a list under **IDocumentUploadRequest**. The list is then processed in an asynchronous process request by request, with any number of items (**ContentDocuments**, **Attachments**, **EmailMessages**) being uploaded.

Currently, only those process interface implementations already included in the application are supported. Custom implementations are ignored.

Signature

```
global interface IDocumentUploadRequest
```

Is implemented by:

- Using the "AttachmentMigrationRequest" class
- Using the "ContentDocumentUploadRequest" class
- Using the "EmailMessageUploadRequest" class
- Using the "UploadNewVersionRequest" class

1.2.17. Using the "RecordValueProvider" class

The class **dvelop_docs_dev.RecordValueProvider** is an abstract class that provides a mechanism for dynamically supplying record values.

The class implements the internal interface **dvelop_docs_dev.IDynamicValueProvider** and can be enhanced with other classes to create specific implementations for retrieving record values.

Signature

```
global with sharing abstract class RecordValueProvider implements
IDynamicValueProvider
```

Methods

The class **RecordValueProvider** offers the following methods:

- provide(input)

provide(input)

Dynamically provides record values and metadata based on the given input.

Signature

```
global abstract RecordValueProviderOutput provide(RecordValueProviderInput
input)
```

Returned entry

An instance of the output class dvelop_docs_dev.RecordValueProviderOutput that contains the calculated values and metadata.

Use

The following section shows you an example of how you can deploy the class **RecordValueProvider** for your own dynamic values:

Example

```
global class CustomRecordValueProvider extends
dvelop_docs_dev.RecordValueProvider {

    global override dvelop_docs_dev.RecordValueProviderOutput
provide(dvelop_docs_dev.RecordValueProviderInput input) {
        Id recordId = input.getRecordId();

        List<System.SelectOption> selectOptions = new
List<System.SelectOption>{
            new System.SelectOption('valueA', 'Label A'),
            new System.SelectOption('valueB', 'Label B'),
            new System.SelectOption(recordId, 'Record ID')
        };

        return
dvelop_docs_dev.RecordValueProviderOutput.picklist(selectOptions);
    }

}
```

1.2.18. Using the “RecordValueProviderInput” class

The class **dvelop_docs_dev.RecordValueProviderInput** provides input values for the dynamic determination of values and metadata using a custom implementation of the class dvelop_docs_dev.RecordValueProvider.

Signature

```
global with sharing class RecordValueProviderInput implements
IDynamicValueProviderInput
```

Methods
The class **RecordValueProviderInput** offers the following methods:

- getRecordId()

getRecordId()
Specifies the ID of the context record for the process.

A context record is a data record on which a process was started with the dynamic determination of values and metadata (for example, an upload process in which dynamic selection values or default values are to be determined for the document properties).

Signature
```
global Id getRecordId()
```

Returned entry
The ID of the context record. If a process does not have a context record, **zero** is returned.

1.2.19. Using the "RecordValueProviderOutput" class
The class **dvelop_docs_dev.RecordValueProviderOuput** is used to transfer output values for the dynamic value and metadata determination process.

Signature
```
global with sharing class RecordValueProviderOutput implements
IDynamicValueProviderOutput
```

Methods
The class **RecordValueProviderOutput** offers the following methods:

- picklist(selectOptions, defaultValue)
- multipicklist(selectOptions, defaultValues)
- of(displayType, value)

picklist(selectOptions, defaultValue)
Factory method that creates a class instance for the data type **Schema.DisplayType.PICKLIST** from a list (**System.SelectOption**) in order to display a picklist.

Signature
```
global static RecordValueProviderOutput picklist(List<System.SelectOption>
selectOptions, String defaultValue)
```

Parameters
- **selectOptions**: A list of options for the displayed picklist.
  Data type: **List<System.SelectOption>**
- **defaultValue**: An optional value for the default value. If you do not want to set a value as the default, you can submit **zero**.
  Data type: **String**

Returned entry
A class instance for the data type **Schema.DisplayType.PICKLIST**.

multipicklist(selectOptions, defaultValues)

Factory method that creates a class instance for the data type **Schema.DisplayType.MULTIPICKLIST** from a list on **System.SelectOption** in order to display a picklist.

Signature

```
global static RecordValueProviderOutput
multipicklist(List<System.SelectOption> selectOptions, List<String>
defaultValues)
```

Parameters

- **selectOptions**: A list of options for the displayed multi-picklist.
  Data type: **List<System.SelectOption>**
- **defaultValues**: An optional list of default values. If you do not want to set any values as defaults, you can submit **zero**.
  Data type: **List<String>**

Returned entry

A class instance for the data type **Schema.DisplayType.MULTIPICKLIST**.

of(displayType, value)

Factory method that creates a class instance for a user-defined type and value.

For specific data types, the appropriate data type must be transferred in the parameter **value**. In this case, we recommend calling the specialized method directly. This procedure generally applies for data types for which a specialized factory method exists, such as **Schema.DisplayType.PICKLIST**.

Signature

```
global static RecordValueProviderOutput of(Schema.DisplayType displayType,
Object value)
```

Parameters

- **displayType**: The data type with which the determined value is to be displayed in the storage dialog.
  Data type: **Schema.DisplayType**
- **value**: The dynamically determined value for the search or storage.
  Data type: **Object**

Returned entry

A class instance for the transferred data type and the determined value.

1.2.20. Using the class “OpenAPI”

The class **dvelop_docs_dev.OpenAPI** provides a number of different functions that make it easier to work with DMS documents in Apex.

Signature

```
global class OpenAPI
```

Methods

The **OpenAPI** class provides the following methods:

- salesforceDatetimeToDMSDatetimeString(salesforceDatetime)
- getValidCookieForCurrentUser()

salesforceDatetimeToDMSDatetimeString(salesforceDatetime)

Converts an Apex datetime object into a character string. The character string is supported by the connected DMS. Under some circumstances, property fields in d.velop documents require a specific format for date entries. In such cases, this method can convert an existing **salesforceDatetime** item into the required string.

Signature

```
global static String salesforceDatetimeToDMSDatetimeString(Datetime
salesforceDatetime)
```

Parameters

- **salesforceDatetime**: A **salesforceDatetime** item in Salesforce that is to be converted into a character string. The character string is supported in the connected DMS.
  Data type: **Datetime**

Returned entry

A character string that is accepted by the **Date**-type property in the DMS.

getValidCookieForCurrentUser()

Delivers a valid cookie for the user that is logged in.

Signature

```
global static String getValidCookieForCurrentUser()
```

Returned entry

The cookie for the user that is logged in.

getValidCookieForConfigUser()

Delivers a valid cookie for the service user that is saved in the d.velop documents configuration.

Signature

```
global static String getValidCookieForConfigUser()
```

Returned entry

The cookie for the service user from the d.velop documents configuration.

generateCredentials()

Logs the logged-in user into the underlying DMS. This method stores cookies for further use.

We recommend using this method at the start of an operation chain and processing all the subsequent operations asynchronously in a queue, for example. This ensures that valid cookies are always available for authentication.

Signature

```
global static void generateCredentials()
```

generateConfigUserCredentials()

Logs the service user defined in the d.velop documents configuration into the underlying DMS. This method stores cookies for further use.

We recommend using this method at the start of an operation chain and processing all the subsequent operations asynchronously in a queue, for example. This ensures that valid cookies are always available for authentication.

Signature

```
global static void generateConfigUserCredentials()
```

initializeTestEnvironment()

Creates the minimum required data and HTTP mocks for unit tests.

Signature

```
global static void initializeTestEnvironment()
```

createStub(parentType, stubProvider)

Creates a stub of the given **parentType** using the stub API in the **dvelop_docs_dev** namespace.

Signature

```
global static Object createStub(System.Type parentType, System.StubProvider
stubProvider)
```

Parameters

- **parentType**: The type of stub to be created.
  Data type: **System.Type**
- **stubProvider**: The provider class to edit the method calls for the created stub instance.
  Data type: **System.StubProvider**

Returned entry

The created stub instance for use in tests.

1.2.21. Using the “SubscriberInterface” class

The **dvelop_docs_dev.SubscriberInterface** abstract class serves as the central superclass for all service interfaces that are available to you for uploading and managing documents in Apex.

In particular, the class provides simple methods of authentication to add valid user sessions to expanding subclasses and to ensure that all outgoing HTTP requests are given the necessary authorization.

Signature

```
global abstract with sharing class SubscriberInterface
```

Is enhanced by:

- Using the class “DocumentManager”
- Using the class “DocumentUploader”

Methods

The **SubscriberInterface** class offers the following methods:

- authenticateInCurrentUserContext()

authenticateInCurrentUserContext()

Authenticates the service interface in the context of the logged-in user by transferring the user session with **setAuthentication(cookie)**. After successful authentication, the session information is cached in d.velop documents using the **Salesforce Platform Cache** and retrieved from this memory as required.

Signature

```
global void authenticateInCurrentUserContext()
```

authenticateInServiceUserContext()

Authenticates the service interface in the context of the service user from the d.velop documents configuration by transferring the user session with **setAuthentication(cookie)**. After successful authentication, the session information is cached in d.velop documents using the **Salesforce Platform Cache** and retrieved from this memory as required.

Signature

```
global void authenticateInServiceUserContext()
```

setAuthentication(cookie)

Transfers a user session in the form of a cookie to the service interface.

The exact implementation depends on the specific service interface.

Signature

```
global abstract void setAuthentication(String cookie)
```

Parameters

cookie: A valid user session cookie to authorize the service interface.

Data type: **String**

isAuthenticated()

Returns the authentication status of the service interface.

The exact implementation depends on the specific service interface.

Signature

```
global abstract Boolean isAuthenticated()
```

Returned entry

The authentication status (**true** / **false**).

1.2.22. Using the “UploadNewVersionRequest” class

The **dvelop_docs_dev.UploadNewVersionRequest** class defines the parameters that are used to upload a new version of a DMS document.

Signature

Using the “IDocumentUploadRequest” interface

```
global class UploadNewVersionRequest implements IDocumentUploadRequest
```

Constructors

You can create the **UploadNewVersionRequest** class using the following constructors:

- UploadNewVersionRequest()
- UploadNewVersionRequest(contentDocumentId, dmsDocumentId, useConfigUserForUpdate)

UploadNewVersionRequest()

This constructor creates the **UploadNewVersionRequest** without parameters.

Signature

```
global UploadNewVersionRequest()
```

UploadNewVersionRequest(contentDocumentId, dmsDocumentId, useConfigUserForUpdate)

This constructor creates the **UploadNewVersionRequest** with all the necessary parameters.

Signature

```
global UploadNewVersionRequest(Id contentDocumentId, String dmsDocumentId,
Boolean useConfigUserForUpdate)
```

Parameters

- **contentDocumentId**: The ID of the **ContentDocument** file to be uploaded as a new version.
  Data type: **String**
- **dmsDocumentId**: The ID of the DMS document to be given the new version.
  Data type: **String**
- **useConfigUserForUpdate**: Specifies whether the process is run with the service user's credentials or those for the current logged-in user.

  > **Warning**
  > The **useConfigUserForUpdate** parameter is obsolete and is only supported in flows. You can run the upload process with different credentials using the **DocumentUploader.Options** class instead.

  Data type: **Boolean**
  Default value: **false**

Properties

The **UploadNewVersionRequest** class has the following properties:

- **contentDocumentId**: The ID of the **ContentDocument** file to be uploaded as a new version.
  Data type: **String**
- **dmsDocumentId**: The ID of the DMS document to be given the new version.
  Data type: **String**
- **useConfigUserForUpload**: Specifies whether the process is run with the service user's credentials or those for the current logged-in user.

  > **Warning**
  > The **useConfigUserForUpdate** property is obsolete and is only supported in flows. You can run the upload process with different credentials using the DocumentUploadOptions class instead.

  Data type: **Boolean**

Default value: false

1.3. Additional information sources and imprint

If you want to deepen your knowledge of d.velop software, visit the d.velop academy digital learning platform at https://dvelopacademy.keelearning.de/.

Our E-learning modules let you develop a more in-depth knowledge and specialist expertise at your own speed. A huge number of E-learning modules are free for you to access without registering beforehand.

Visit our Knowledge Base on the d.velop service portal. In the Knowledge Base, you can find all our latest solutions, answers to frequently asked questions and how-to topics for specific tasks. You can find the Knowledge Base at the following address: https://kb.d-velop.de/

Find the central imprint at https://www.d-velop.com/imprint.