# Introduction

Since version 16.2 OCC provides a Rest API to allow browser based clients to communicate with the OCC server. Right now the API is not released for public use except for those functions needed to ingest documents. These functions are described here together with a code sample[[1]](#footnote-1) for .Net that demonstrates its use.

In order to ingest documents into OCC you follow these steps:

1. Log in to OCC
2. Create a batch
3. Add files to the batch
4. Close the batch

# Login

The login procedure consists of three steps.

1. Get the base path of the ODTS server.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter | Return (JSON) |
| Get | api/v1/account/otdsPath |  | otdsPath |

1. Log in to OTDS, typically using username and password. That login returns an OTDS ticket.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter (body, JSON) | Return (JSON) |
| Get | {otdsBasePath}/v1/authentication/credentials | {  user\_name,  password  } | Ticket |

1. With the OTDS ticket you log in to OCC. This call puts an OCC token into the cookie that controls your session. The login call to OCC also returns a XSFR token that must be placed into the header of each rest call to OCC.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter (form) | Return (Header) |
| Post | api/v1/account/otdsLogin | OTDSTicket | XSRF-TOKEN |

The OCC session is available only for a certain period of time, which can be set on the OCC server. A typical time would be 20 minutes. Whenever a rest call is made against the OCC server the timer is reset and the full 20 minutes are available. Therefore, it makes sense to check the session before a transaction is made against the OCC server whether the session is still valid. The function assertLogin in the sample solution does exactly that.

# Create a batch

There are two types of batch. One takes several files and attaches them to the batch. OCC then is responsible to structure the batch into documents (document separation). In the other type you create the documents using Rest calls and add one or more files to each document.

The return value of the rest call contains the batch-Id and an operation-Id. The latter is only to close the batch.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter | Return (JSON) |
| Post | api/v1/batches?profileName={profile}&batchName={batchname}&batchFormat=looseFilesInput  api/v1/batches?profileName={profile}&batchName={batchname}&batchFormat=documentFilesInput | - | {  BatchID  OperationID  } |

# Create a document

The call to create a document has the batch-ID as parameter. It returns a document-ID.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter | Return (JSON) |
| Post | api/v1/batches/{batchID}/documentFilesInput/inputDocuments?documentClassName= | - | DocumentID |

# Add a file to a batch

In order to add a file to a batch you need to make a Multipart-Form rest call. Parameter is the batch-ID.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter | Return |
| Post | api/v1/batches/{batchID}/looseFilesInput/inputFiles | - | - |

# Add a file to a document

In order to add a file to a batch you need to make a Multipart-Form rest call. Parameter is the batch-ID and the document-ID.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter | Return |
| Post | api/v1/batches/{batchID}/documentFilesInput/inputDocuments/{documentID}/inputFiles | - | - |

# Finalize a batch

You end the batch creation process by closing the current operation. You need to specify the batch-ID and the operations-ID.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter | Return |
| Post | api/v1/batches/{batchID}/operations/{operationID}/closeAndSubmitAction | - | - |

# Error handling

If an error occurs during batch creation you should try to explicitly abort the current operation. If you don’t the batch will remain in state “Input” till OCC’s cleanup services puts the batch into the broken state. The breakAction Rest call will put the batch into the broken state right away.

There are of course chances that the breakAction call itself fails, e.g. if the network connection has an issue.

|  |  |  |  |
| --- | --- | --- | --- |
| Verb | Path | Parameter (Body, JSON) | Return |
| Post | api/v1/batches/{batchID}/operations/{operationID}/breakAction?errorMessage | {  errorMessage,  errorDetails  } | - |

There is one special error situation and that is when the batch closing call (closeAndSubmitAction) fails. In this (highly unlikely) case it is not clear whether the ingestion was successful. A reason for failure can be for example that the communication was interrupted during this call or that OCC when down for some reason. A re-injection of the batch may lead to duplicates. For OCC 16.4 we plan an inquiry call that lets you query the ingestion status based on the batchID.

1. The code sample assumes you want to constantly ingest documents into OCC, like picking them up from some web service. [↑](#footnote-ref-1)