Common Web Services for Laserfiche

Contents

System Description

The Common Web Services API (CWSAPI) provides API endpoints for CRUD functionality with a Laserfiche document store. In Figure 1, it sits in a public/DMZ zone, providing accessibility and functionality to the firewall protected Laserfiche server for SAAS services. It uses a login credential, tokenized and passed with every query, to ensure security.

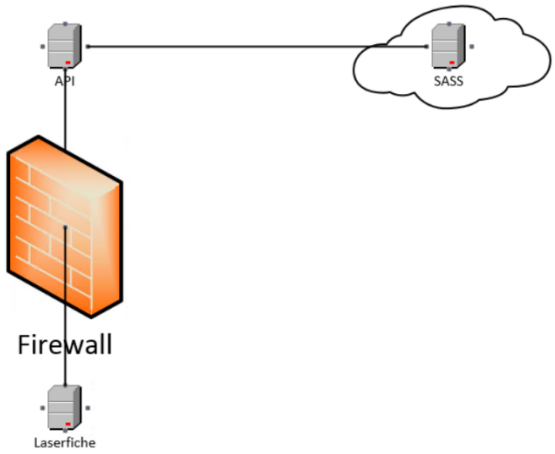

Figure 1 – visual representation of the API and its connections

Service Pattern

The service loosely follows RESTful web service standards[i]. POST, GET, PUT, and DELETE commands are employed with named routes and parameters where necessary. It also follows the OAuth style of using bearer tokens to secure interactions.

File Uploading

Overall, uploading files through the CWSAPI is a two-step process. First, the API accepts uploads to a temporary location, and then a background process completes the upload interaction with the Laserfiche server. There are three procedures available to upload files through the CWSAPI, one for files of smaller than 10MB and the other for files larger.

Two procedures are available for uploading a file smaller than 10MB. The simplest is to post to CreateDocument with the file content present in the POST header. Alternatively, create a document with no file content (CreateGenericDocument), then upload the file contents with UploadContent.

The procedure for uploading documents larger than 10MB is to create a document with no file content (CreateGenericDocument), upload individual file contents via chunks (UploadChunk), and then set the file upload as completed (CompleteUpload) to finalize. It is possible to monitor a file's uploading/finalized status with the endpoint function GetStatus.

User Limitation

The Laserfiche connection operates on named users (a lot like form users). Each user has four concurrent connections. After the user exceeds this limit, they will get an error message ("maximum sessions exceeded"). In rare instances, Laserfiche may grant an organization an exception to the four concurrent connection limit through a paid-for integrator license. This exception must be managed through the organization's Laserfiche Solution Provider (also known as a VAR) through a Special Order Desk process.

Thresholds

The application has some thresholds and constant values. One example, above, is the user concurrent connection limit of four. These are provided primarily for reference.

- Upload buffer length: for uploading images or documents to the CWSAPI, the limit on content length is 10485760 bytes (10 MB)
- Background buffer length: for the background uploading process, moving files from temporary storage to permanent Laserfiche storage, the buffer limitation is 4096 bytes (4 KB).
- User connection limitation: a user interacting with the service is limited to 4 connections to maintain availability across many users, unless a paid-for special exception is provided by Laserfiche for the application and organization in question.

Testing the CWSAPI

Here we'll present an example of creating a document in the Laserfiche server via the CWSAPI. Essentially this is a walkthrough of how we have applied OAuth API security practices to the CWSAPI.

Requirements:

- A method for creating a Base64 string of the JSON serialized LaserficheCredentials class
- A method for testing API endpoints, typically an HTTP client. In this document we will be showing examples using Postman.
- A running instance of CWSAPI, connected to a Laserfiche server

Steps

1- Encode the credentials
2- Log into the service
3- Collect the bearer token
4- Call the CreateDocument function

Encode the Credentials

Create a Base64 string of the JSON serialized LaserficheCredentials class. First, a look at the LaserficheCredentials class. Here we present it in the native C# (Figure 2) and the deserialized JSON.

```
public class LaserficheCredentials
{
    1 reference | simplecode, 279 days ago | 1 author, 1 change
    public string Username { get; set; }

    1 reference | simplecode, 279 days ago | 1 author, 1 change
    public string Password { get; set; }

    1 reference | simplecode, 279 days ago | 1 author, 1 change
    public string ServerName { get; set; }

    1 reference | simplecode, 279 days ago | 1 author, 1 change
    public string RepositoryName { get; set; }
}
```

Figure 2- LaserficheCredentials C#

And the serialized representation of that in JSON:

{"username": null,

"password": null,

"serverName": null,

"repositoryName": null }

The Username and Password are required for this connection, regardless of if you have enabled an Active Directory for your login to access Laserfiche. ServerName is specifically the name of your Laserfiche server; its IP address will not work.

Take those credentials (with parameters specific to your setup), encode them to UTF8, then convert them to a Base64 string. Figure 3 shows the native C# appearance.

```
var encodedCredentials = Convert.ToBase64String(Encoding.UTF8.GetBytes(jsonCredentials));
```

Figure 3- encoding the JSON serialized credentials

Log into the Service

Using those encoded credentials, open your Http client software and point it to ConnectionToLaserfiche for a POST. In my example, that looks like: http://localhost/CWSAPI/api/ConnectionToLaserfiche. Add the following header keys and values to the request:

- Authorization :: “Basic [encodedCredentials value]”
- Content-Type :: “application/x-www-form-urlencoded”

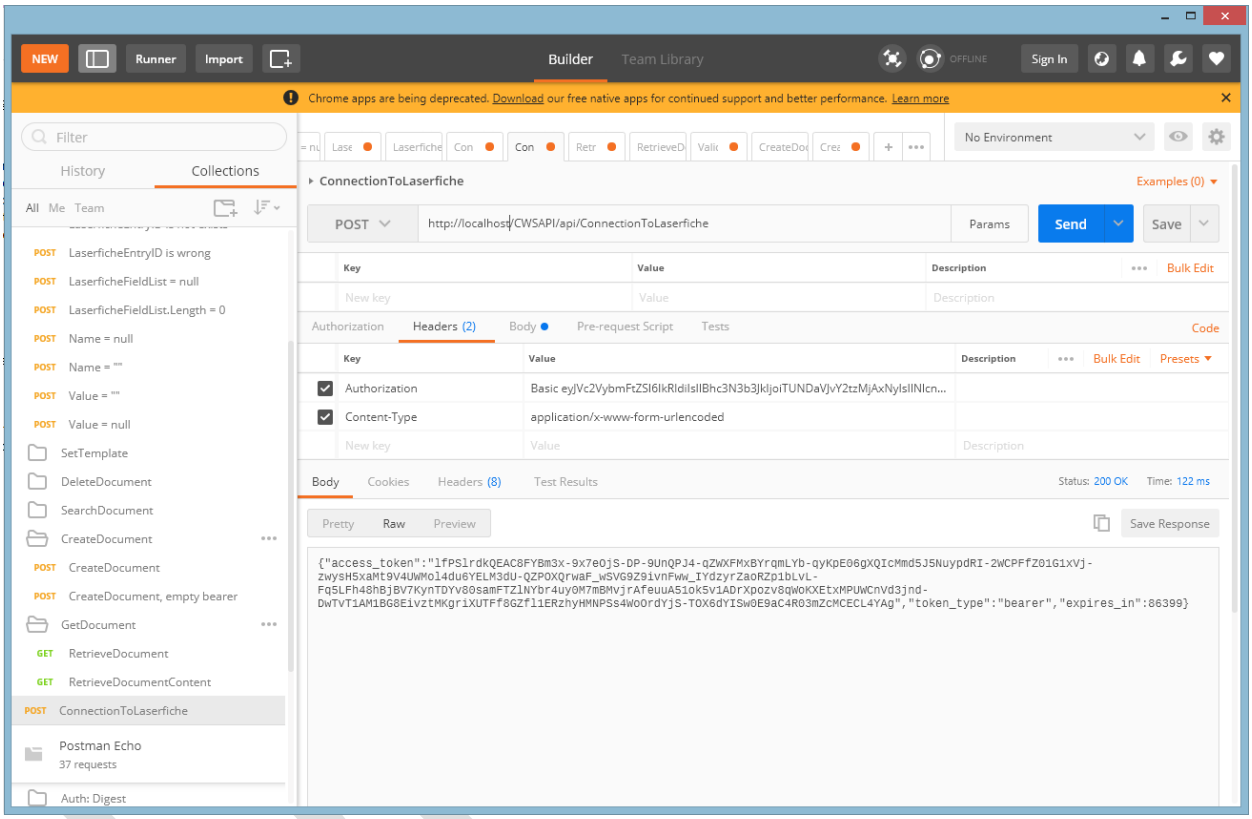

Figure 4- calling ConnectionToLaserfiche

In that image, you can see the response from the server contains an access token of “bearer” type and its expiration in seconds. That is the bearer token to use in subsequent calls to the API.

If the connection works but the Authorization header is incorrect or empty, you will receive an appropriate error response from the server. Possible responses are detailed in this endpoint’s details section later in the document.

Call the CreateDocument Function

In the Http client software, create a new POST connection to the CreateDocument endpoint. Add a header for this connection with the key “Authorization”, and the value “Bearer [bearer token value]”. In the Body, add the keys/values:

- file (of file type) :: a sample file smaller than 10MB
- parameters (of text type) :: [JSON serialized parameter]

As seen in Figure 5, a successful POST call to the endpoint yields a Laserfiche ID for the new document created.

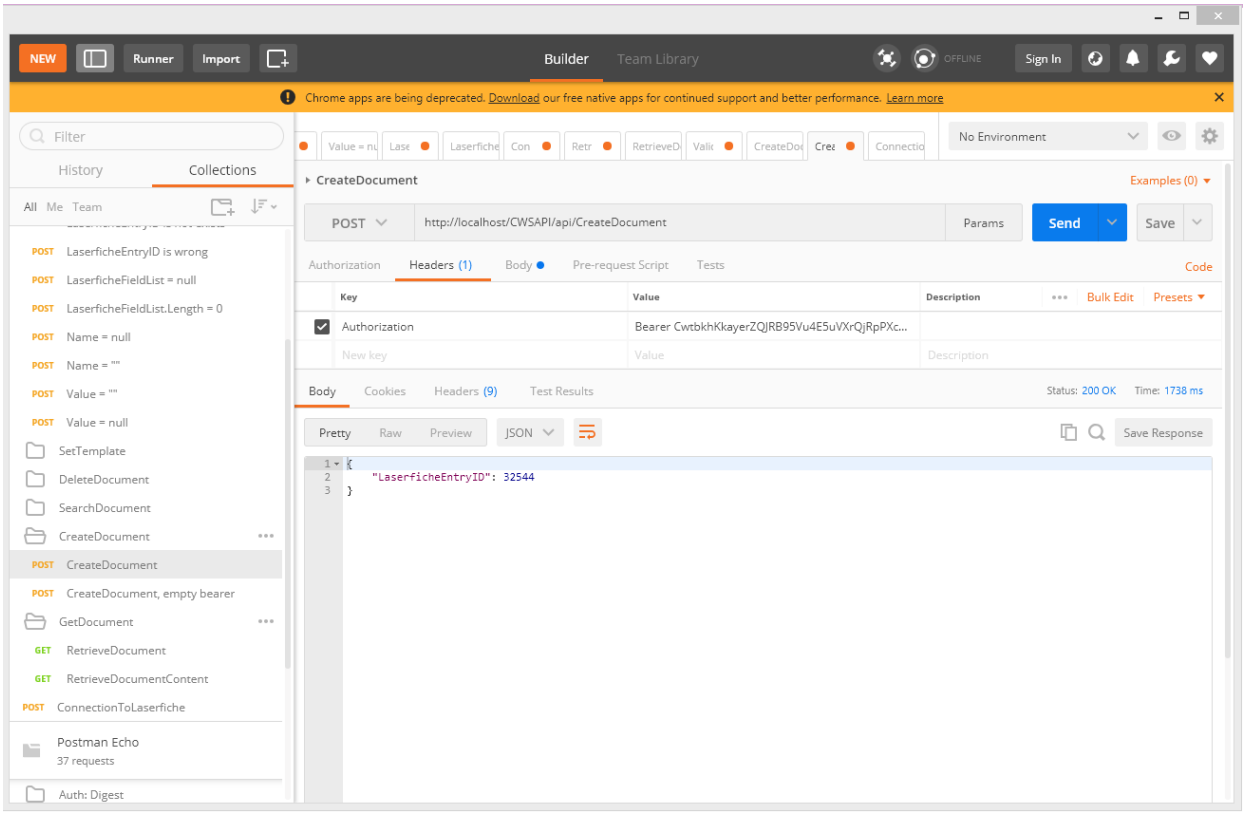

Figure 5- a successful POST to CreateDocument

CWSAPI Endpoints

All API calls except for ConnectionToLaserfiche require SecurityToken to be passed with the query, as all other calls require the connection to be open. ConnectionToLaserfiche initializes this token based on the parameters given to it. With the understanding that not including this token will always throw an "Authorization has been denied for this request" error for all calls other than ConnectionToLaserfiche, that authorization error will not be specified for the functions detailed here.

Note: in this version of the document we are focusing on the core functionality and processes. A future version will further flesh out specific file-chunking functionality.

Meta and Folder Functions

Function: ConnectionToLaserfiche

This function is the foundation for security. It returns a bearer token to be used with all other API calls, which expires after 86399 seconds (1 day). A step-by-step instruction on accessing this function is detailed in the Testing the CWSAPI section.

Route: /api/ConnectionToLaserfiche [POST]

Returns: JSON object

Name	Type
Access_token	String
Token_type	String
Expires_in	int

Headers:

Name	Type
Authorization	Basic [Base64 string (encrypted JSON object containing Username, Password, Repository, and Server)]
Content-Type	application/x-www-form-urlencoded

Possible errors: cannot connect to Laserfiche, invalid credentials, missing credentials

Function: EncryptString

Returns a UTF8 encrypted string as plain text.

Route: /api/EncryptString

Headers:

Name	Description
Authorization	Bearer [bearer token]

Parameters:

Name	Type
text	string

Function: GetInfo

Returns an object containing two lists available to this login: volumes, and templates with fields listed as children. This is used in your application if you need to allow users to select from a list of templates, or to know which fields are available for a template. These results can then be used in other calls such as CreateDocument and SetTemplate.

Route: /api/GetInfo [GET]

Headers:

Name	Description
Authorization	Bearer [bearer token]

Returns: a JSON object containing collections of the complex object Templates and simple object Volumes. See Fig. 6

Figure 6 - GetInfo response

Parameters: none

Possible errors: none

Function: Browse

Get a list of folders and documents directly underneath the specified folder. This reads in a URL parameter that is the Laserfiche folder path. The path URL parameter is mandatory.

Returns: a JSON object containing collections of the complex object Templates and simple object Volumes, identical to GetFolderContents. See Fig. 7

Name	Type
EntryId	Int
Name	String
EEntryType	Document = -2, Shortcut = -1, Folder = 0, RecordSeries = 1

Route: /api/browse [GET]

Sample route:
/api/browse?path=rootFolder\subfolder


```
[
    {
        "EntryId": 31215,
        "Name": "filename",
        "Type": "Document"
    },
    {
        "EntryId": 32338,
        "Name": "filename (2)",
        "Type": "Document"
    },
    {
        "EntryId": 32368,
        "Name": "filename (3)",
        "Type": "Document"
    },
    {
        "EntryId": 32400,
        "Name": "filename (4)",
        "Type": "Document"
    },
    {
        "EntryId": 32545,
        "Name": "test 2",
        "Type": "Folder"
    }
]
```


Figure 7- an example of Browse and GetFolderContents response

Headers:

Name	Description
Authorization	Bearer [bearer token]

URL parameters:

Name	Type
Path	string

Possible errors: ObjectNotFound, ArgumentOutOfRange, ArgumentException, InvalidOperation

Function: GetFolderContents

Get a list of folders and documents directly underneath the specified folder. This reads in a URL parameter that is the Laserfiche folder ID. See Fig. 7 and the Browse function.

Returns a collection:

Name	Type
EntryId	Int
Name	string
EEntryType	Document = -2, Shortcut = -1, Folder = 0, RecordSeries = 1

Route: /api/folders/{folderId:int:min(1)}/contents [GET]

Sample route: /api/folders/1/contents

Headers:

Name	Description
Authorization	Bearer [bearer token]

Parameters:

Name	Type
folderId	int

Possible errors: ObjectNotFound, ArgumentOutOfRange, ArgumentException, InvalidOperation

Document Functions

Function: CreateDocument

CreateDocument can be used to set template and field data at time of creation with one transaction, or the setting of template and field data can be done separately through the below setMetadata and setTemplate calls. Also, while this function can upload smaller files, we recommend larger files be uploaded using the chunking methods.

Returns JSON with key LaserficheEntryId and int value.

Route: /api/CreateDocument [POST}

Headers:

Name	**Description**
Authorization	Bearer [bearer token]

Body, in form-data:

Key	**Value**
File	File of less than 10MB
Parameters	A JSON object of the contents seen below

Parameters value in form-data key:

Key	**Value**
LaserficheFolderPath	Path in Laserfiche to store document
LaserficheDocumentName	Name of document to create in Laserfiche
LaserficheVolumeName	Volume in Laserfiche to store document
LaserficheTemplateName	**Optional** name of template to set on document
LaserficheFieldList	**Optional** List of fields and values (name, value)

Example Parameters value:

```
{
"LaserficheFolderPath": "test",
"LaserficheDocumentName": "test10.tif",
"LaserficheVolumeName": "Default",
"LaserficheTemplateName": "templ14",
"LaserficheFieldList": [
{"Name": "Description", "Value": "88"}
]
}
```

Possible errors: ArgumentNullException, JsonReaderException, LaserficheRepositoryException, InvalidMetadataException, ArgumentException

Function: CreateGenericDocument (Incomplete)

CreateGenericDocument can be used to create a Laserfiche document, setting template and field data at time of creation, and leaving open the possibility of adding document file contents for later. The setting of template and field data can be done separately through the below SetMetadata and SetTemplate calls. Add file contents with the UploadChunk and CompleteUpload functions.

Returns JSON with key LaserficheEntryId and int value.

Route: /api/CreateGenericDocument [POST]

Headers:

Name	**Description**
Authorization	Bearer [bearer token]

Parameters value in form-data key:

Name	**Description**
DocumentName	Name of document to create in Laserfiche
EntityType	An entity type listed in the file EntityTypes.xml
Metadata	**Optional** List of fields and values (name, value)

Possible errors: ArgumentNullException, JsonReaderException, LaserficheRepositoryException, InvalidMetadataException, ArgumentException

Function: SetMetadata

Using the a Laserfiche entry id (supplied by CreateDocument) this sets new template field data or update it.

Returns a 200 status if successful.

Route: /api/SetMetadata [POST}

Headers:

Name	Description
Authorization	Bearer [bearer token]

Paramenters:

Name	Description
LaserficheEntryID	Laserfiche Document ID
LaserficheFieldList	List of JSON objects: Name(string) IsMulti(bool) Value(string) Values(string[])

Example Parameters value:

```
{
  "LaserficheEntryID": "30699",
  "LaserficheFieldList": [
    {"Name": "Description", "Value": "2266"}
  ],
}
```

Possible errors: LaserficheRepositoryException

Function: SetTemplate

This is used to set the template of a document by LaserficheEntryId. If a template has a required field that has not previously been set the template will fail to apply.

Returns a 200 status if successful.

Route: /api/SetTemplate [POST}

Headers:

Name	Description
Authorization	Bearer [bearer token]

Parameters:

Name	Description
LaserficheEntryID	Laserfiche Document ID
LaserficheTemplateName	Template Name

Example Parameters value:

```
{
 "LaserficheEntryID": 124,
 "LaserficheTemplateName": "General"
}
```

Possible errors: LaserficheRepositoryException

Function: DeleteDocument

Deletes document from Laserfiche based on the entry ID. **DeleteDocument doesn't delete from Laserfiche; rather it will tag the document in Laserfiche with a "Flagged for Deletion" tag so that an administrator can review. Prior to deleting, SearchDocument call may be run to ensure that the application knows the LaserficheEntryID to request for deletion.**

Returns a 200 status if successful.

Route: /api/DeleteDocument [DELETE}

Headers:

Name	Description
Authorization	Bearer [bearer token]

Parameters:

Name	Description
LaserficheEntryID	Laserfiche Document ID

JSON Example:

```
{
  "LaserficheEntryID": 124
}
```

Possible errors: LaserficheRepositoryException

Function: SearchDocument

Search Laserfiche based on a Laserfiche search phrase, including an optional set of requested metadata, and returns a JSON object of the results. This is the same search functionality as found in Laserfiche Client or Web Access. Go to the Basic Search Queries subsection below for more examples and options.

Route: /api/SearchDocument [POST}

Headers:

Name	Description
Authorization	Bearer [bearer token]

Parameters:

Name	Description
LaserficheSearchPhrase	A string containing a Laserfiche formatted search phrase
MetaDataObjectList	A JSON array of requested metadata, with the format {“Name”: [name]}

JSON Example:

```
{
        “LaserficheSearchPhrase":"{[General]:[Document]=\"search text\", [Date]=\"*\"} &
{LF:Name=\"*\", Type=\"F\"}",
        "MetaDataObjectList":[
                {"Name","Description"},
                {"Name","Date"}
                ]
}
```

Returns JSON objects:

Name	Type
DocumentId	Laserfiche Document Id
DocumentName	Laserfiche Document Name
FileType	Mime Type
String array of field data (FieldDataList)	List of Field name and Field data for selected field types

Possible errors: LaserficheRepositoryException

Using Laserfiche Search Syntax in LaserficheSearchPhrase

Instead of building Search Syntax by referencing documentation, MCCi recommends building your Search Syntax through the front-end end-user Laserfiche UI, then copying out the resulting syntax to utilize in your API call. Search queries can be generated both in the Laserfiche “thick” Client as well as in the thin Web Client. To generate the query in Laserfiche Client, open the client to a repository, and click

the Search command (magnifying glass icon). Within that search, build the search that you plan to have your API call run, and then run the search. Then, select the "Customize Search" dropdown and add the Search Syntax option, which is where you observe the resulting search phrases, as seen in Figure 8.

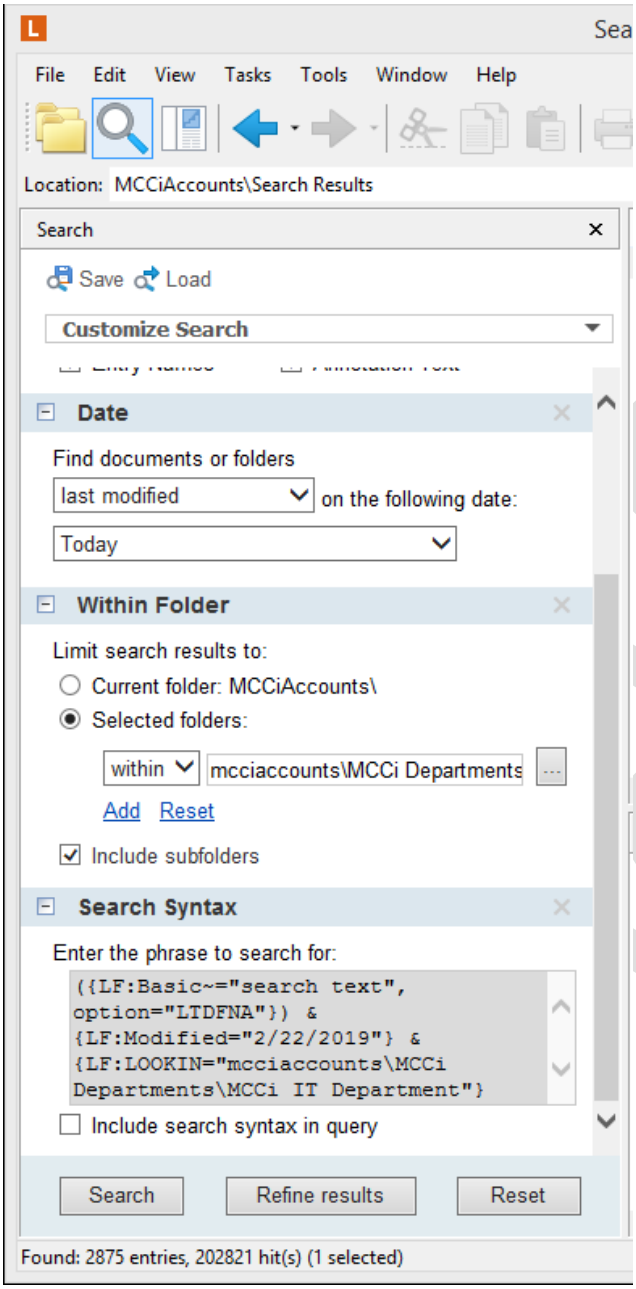

Figure 8 - Laserfiche Search Syntax as displayed in thick Client

All the options within the "Customize Search" dropdown are available to pass through the CWSAPI SearchDocument function. This is an example of the Laserfiche search syntax, which would search for documents named Laserfiche or documents with the word "Laserfiche" in their OCRed text, but restricted to only documents located in the Laserfiche folder path "dev-01\OCR\OCR Test\Job1" or one of its sub folders:

({LF:Basic~="Laserfiche", option="LTDN"}) & {LF:LOOKIN="dev-01\OCR\OCR Test\Job1"}

After generating the syntax, for use in posts against the CWSAPI as the value of LaserficheSearchPhrase, you must escape any double quotes with a backslash ("\"). The above query would then look like this:

({LF:Basic~=\"Laserfiche\", option=\"LTDN\"}) & {LF:LOOKIN=\"dev-01\OCR\OCR Test\Job1\"}

For ease of use, the rest of the examples presented in this document have had their double quotes escaped with a \. For ease of using the CWSAPI, we have included here some examples of search phrases. We still recommend building your search in the front-end end-user UI, then looking at the Advanced Search Syntax to ensure that the syntax meets your needs.

Searching for "search text" within a document, document names, annotation text, and all metadata fields:

"LaserficheSearchPhrase": "({LF:Basic~=\"search text\", option=\"LTDFNA\"})"

Searching for a document or folder with the name of "client", last modified on or after 2/21/2019:

“LaserficheSearchPhrase": “{LF:Name=\"client\", Type=\"DF\"} & {LF:Modified>=\"2/21/2019\"}”

Figure 9- How to generate a search query in Laserfiche Client

Searching for all entries with a single tag, “Agendas”, created on or after 12/11/2018:

“LaserficheSearchPhrase": “{LF:Created>=\”12/22/2018\”} & ({LF:Tags=\"Agendas\"})”

For more examples of valid search syntaxes and searching options, we refer you to Laserfiche’s Search documentation[ii].

Using MetaDataObjectList

By default, the SearchDocument call only returns a list of relevant EntryIDs, Entry Names, and mime-types for documents that are relevant to the LaserficheSearchPhrase presented to the API. If you want to return metadata assigned to the document (known as Fields in Laserfiche), you need to include a list of the Fields you are interested in when making the call. These fields are listed in the MetaDataObjectList. In the example provided at the beginning of this Function’s documentation, the values of the fields “Description” and “Date” will be returned alongside each Entry ID, Entry Name, and the document’s mime-type. Returning this information is especially useful if you plan to build a list of search results in your application. Without this metadata, you would be limited to presenting the name of the document in your list of results. With this metadata, you could potentially build out a list of metadata to the side of document names, to provide the end user additional context prior to selecting which document they want to open.

A Note on Opening Documents

SearchDocument is not intended to provide the actual content of the documents returned by the Search. It provides a list of documents that are responsive to the search. With that list, you may allow the user to access the content through two different methods after building a list of search results for them to utilize.

Method 1: You may use the RetrieveDocumentContent method described below to download the content of the record. This completely circumvents the need to use the Laserfiche User Interface, or the need to have a Named User license for every user accessing the records. However, the user misses out on additional functionality.

Method 2: Directly link to the content in Web Client (formerly known as Web Access) or Weblink (Laserfiche’s read-only portal, also known as Public Portal). Advantages of this method are described in the next paragraph.

In Laserfiche’s Web Client, every document has a static and unique URL. The user must have a named-user license with appropriate access rights in Laserfiche to open content this way. The advantage of presenting content in this manner is that the user gains all of the functions and features of the Laserfiche Document View, including annotation capabilities and the ability to search for text within the document while viewing the document. After running SearchDocument, you will be presented with a set of EntryIDs, Entry Names, mime-types and relevant Metadata. You could then build a link that the user

can click on to access the record in a new tab, or in an embedded iFrame. As of Laserfiche 10.4 the default link syntax is as follows (change segments in *italics)*:

To access most Document mime-types in Web Client including TIF, JPB, BMP, GIF, DOC, DOCX, XLS, XLSX, PDF, MP4, MP3, OGG, OGV, MOV, WEBM, WAV, OPUS, OGA:

http://*servername*/laserfiche/Docview.aspx?repo=*RepositoryName*&docid=*UniqueEntryID*

Please note that for the above list of supported file types, the system is dependent on the file types that can be streamed by a given browser. Video and Audio file types listed above may or may not stream depending on the browser the user is utilizing. Reference the supported mime-types for your organization's official browsers. Additional file types may be added to the list above if you know for a fact that your organization's browser supports streaming that file type. These can be added to the web.config in your Laserfiche Web Client installation directory with help from your Laserfiche Service Provider.

Other file types may not have a native viewer built in to the Laserfiche Web Client but may still be stored in the repository. For these other mime-types, the URL syntax is slightly different and will initiate a download of the file so that the user may open the file on their local system using a viewer installed on their local system.

To access Document mime-types in Web Client that do not have a native viewer in Laserfiche:

http://*servername*/laserfiche/ElectronicFile.aspx?repo=*RepositoryName*&docid=*UniqueEntryID*

Your search may return documents as well as Folders, since Folders in Laserfiche can have Fields and other metadata assigned to them. If a Folder is returned to you, the following syntax is used to navigate to that folder.

To access a Folder in Web Client:

http://*servername*/laserfiche/browse.aspx?repo=*RepositoryName*#?id=*UniqueEntryID*

In Laserfiche Weblink (aka Public Portal) there is also a unique syntax for each document ID and for accessing Folders.

To access Documents in Weblink:

http://servername/WebLink/DocView.aspx?id=*UniqueEntryID*&dbid=*SingleDigitRepositoryIDConfiguredInWeblink*&repo=*RepositoryName*

To access a Folder in Weblink:

https://*servername*/WebLink/Browse.aspx?id=*UniqueEntryID*&dbid=*SingleDigitRepositoryIDConfiguredInWeblink*&repo=*RepositoryName*

Method 3: Rather than provide a list of search results, you could instead embed an iFrame in to your interface that launches Laserfiche Web Client or Weblink and presents the list of Search Results in the the native Laserfiche interface. Instead of running a SearchDocument API call, you could instead have an

iFrame run the search on page-load. This does not use the API's authentication, and requires you to authenticate the user that is running the browser directly to the Laserfiche repository. This iFrame can log in through pass-through Windows Authentication IF the user is logged in on a workstation that is on the same domain as the Laserfiche Server, and the browser supports passing their credentials to the Laserfiche Web Client or Weblink. This is a very different licensing model than what is presented in this document and may require Unlimited Weblink licensing or Laserfiche Named Users for all of the users that may utilize the Web Client. We suggest working with the Laserfiche Solution Provider to better understand this method, if licensing permits the use of this method.

Function: RetrieveDocument

Fetches a document's primary information (and not contents) from Laserfiche.

Route: /api/RetrieveDocument [GET]

Sample route: /api/RetrieveDocument?LaserficheEntryID=30699

Headers:

Name	Description
Authorization	Bearer [bearer token]

URL parameters:

Name	Type
LaserficheEntryID	int

Returns JSON objects:

Name	Type
LaserficheEntryId	Int
CreationTime	String
LastModifiedTime	String
Extension	String
MimeType	String
Name	String, Laserfiche Document Name
Path	String, path inside of Laserfiche repository
Volume	String
RetrieveDocumentContentURL	String, URL for downloading the content through this API instance

Possible errors: ArgumentException

Function: RetrieveDocumentContent

Fetches a document's contents as attachments/downloads.

Route: /api/RetrieveDocumentContent [GET]

Sample route: /api/RetrieveDocumentContent?LaserficheEntryID=30699

Headers:

Name	Description
Authorization	Bearer [bearer token]

URL parameters:

Name	Type
LaserficheEntryID	int

Returns the contents of the Laserfiche document as a download.

Function: CreateRelationship

Creates a parent/child relationship between two documents.

Returns: 200 if it worked, otherwise Bad request

Route: /api/documents/{parentDocumentId:int:min(1) [PUT]

Sample route: /api/documents/1

Headers:

Name	Description
Authorization	Bearer [bearer token]

URL parameters:

Name	Type
parentDocumentId	int

Parameters:

Name	Type
RelationshipId	Int
ChildDocumentId	Int
Description	String

Function: GetRelationships

Gets all added relationships.

Returns a JSON collection of relationships (Id, Parent, and Child)

Route: /api/relationships [GET]

Headers:

Name	Description
Authorization	Bearer [bearer token]

Returns JSON objects:

Name	Type
Id	Int
Parent	String
Child	String

JSON Return Example:

```
[
  {
    "Id": 2,
    "Parent": "Attachment",
    "Child": "Message"
  },
  {
    "Id": 4,
    "Parent": "DirectTestRel",
    "Child": "ReverseTestRel"
  },
  {
    "Id": 5,
    "Parent": "grandParent",
    "Child": "Parent"
  },
```

```
  {
    "Id": 3,
    "Parent": "Parent",
    "Child": "Child"
  },
  {
    "Id": 8,
    "Parent": "Parent",
    "Child": "jjjj"
  },
  {
    "Id": 6,
    "Parent": "Parent",
    "Child": "PseudoChild"
  },
  {
    "Id": 1,
    "Parent": "Supersedes",
    "Child": "Superseded by"
  }
]
```

Document Sections in Progress

Some document functions need more testing and documentation effort, and will be documented in the next instructions doc version. They are as follows:

- Function: UploadContent
- Function: InitUpload
- Function: UploadChunk
- Function: CompleteUpload
- Function: ResetUpload
- Function: GetStatus

References

[i] REST API Guidelines and Best Practices, https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9

[ii] Laserfiche Search help documentation, https://www.laserfiche.com/support/webhelp/Laserfiche/10/en-US/userguide/#../Subsystems/client_wa/Content/Search/Search-Syntax.htm