ALE Learnset Manager

API

Version: 23.1

Written by: Documentation Team, R&D
Date: Thursday, May 7, 2026

HYLAND™

Documentation Notice

Information in this document is subject to change without notice. The software described in this document is furnished only under a separate license agreement and may only be used or copied according to the terms of such agreement. It is against the law to copy the software except as specifically allowed in the license agreement. This document or accompanying materials may contain certain information which is confidential information of Hyland Software, Inc. and its affiliates, and which may be subject to the confidentiality provisions agreed to by you.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright law, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Hyland Software, Inc. or one of its affiliates.

Hyland, HXP, OnBase, Alfresco, Nuxeo, and product names are registered and/or unregistered trademarks of Hyland Software, Inc. and its affiliates in the United States and other countries. All other trademarks, service marks, trade names and products of other companies are the property of their respective owners.

© 2026 Hyland Software, Inc. and its affiliates.

The information in this document may contain technology as defined by the Export Administration Regulations (EAR) and could be subject to the Export Control Laws of the U.S. Government including for the EAR and trade and economic sanctions maintained by the Office of Foreign Assets Control as well as the export controls laws of your entity's local jurisdiction. Transfer of such technology by any means to a foreign person, whether in the United States or abroad, could require export licensing or other approval from the U.S. Government and the export authority of your entity's jurisdiction. You are responsible for ensuring that you have any required approvals prior to export.

.

Table of Contents

Documentation Notice **2**
About the ALE Learnset Manager(ALM) 5
ALM API *5*
Standards *5*
How-to **5**
5
Connect to Server 5
6
Create a Project 6
6
Get all Projects 6
7
Create a Document Class 7
7
Get Project level Fields 7
7
Configure Project level Fields 7
8
Get all Classes for a Project 8
8
Get Class level Fields 8
8
Check if project is Learnable 8
9
Add Training Documents 9
9
Get Training Document Fields 9
10
Create a Test Set 10
On basis of Uploaded Contents 10
On basis of files located at given path 10

10

Update StreamSet 10

11

Update Learnset 11

11

Training a Field Extractor 11

11

Extraction 11

12

Learn or Relearn a Project 12

File Types 13

Instance management 13

Error Codes 13

Document Formats 13

14

Changelog 14

Resources **14**

Data Types **18**

JSON *18*

About the ALE Learnset Manager(ALM)

The ALE Learnset Manager (ALM) is an application utilizing a trainable engine (ALE - Automated Learning Engine) to bring automation to document processing. ALM is a web-based administration client enabling the capture, preparation, and management of training documents to be learned through the Automatic Learning Engine (ALE).

You can also edit classes by adding and removing training documents to improve the performance of each learnset.

ALM API

ALM Server provides REST interface endpoints of the ALE field extraction engine so that it can be used by applications that are coded in standard development languages, such as Java, C++, or C#, and that are also compatible with HTTP Web services, to send and receive data from ALM Server.

Standards

The ALM Server's approach to working with client applications is based on widely accepted standards. ALM Server uses a RESTful approach for Web services and HTTP/HTTPS transport for structured data exchange.

REST is an architectural style for message exchange that addresses the web as remote resources. In a RESTful application such as ALM Server, each URL points to a resource. This approach differs from that of SOAP in which SOAP exposes the functionality as URL endpoints containing functions that can be called. Unlike SOAP applications that are restricted to using GET or POST operations, REST-based applications include a greater range of operations like GET, POST, PUT, and DELETE.

RESTful applications are stateless, meaning no session state is stored on the server. The information required for a request is included in the request message itself. The client application can cache a resource representation, potentially improving application performance.

However, the documentation does not always link to return types that are used for requests. The types are mentioned in the description - please check the `Data Model` section for details on the referenced type.

How-to

Connect to Server

In order to connect to ALM Server, append `ALM/service` to server base URL. The URL should also contain the `/session` and `User Name` to connect to ALM Server and create a Session.

So the complete URL looks something similar like http://{serverName}:{PortNumber}/ALM/Service/session/{UserName}

To connect to the ALM server, complete the following steps.

- Connect to the server using `POST /ALM/Service/session/{User ID}`
- The Password is passed as parameter in Request Payload.
- Request Header should contain "application/json" as the Content Type.

On a successful login a session is created and session ID is returned in the response. This session ID is used for subsequent calls to ALM Server.

If the supplied user credentials are incorrect then authentication fails and a string `not_authenticated` is returned in the response.

To check whether the session has expired or not, the following request is sent:

- Connect to the server using `GET /session/current`
- In the request header, session ID is passed in "X-CPTMS-Session"

The X-CPTMS-Session header is included in the following subsequent calls. That means if the session is active then the client request header contains the key-value pair of "X-CPTMS-Session" and "Session ID".

Create a Project

A project in ALM is created using `PUT /learnset/project`.

The Request object has the following 3 parameters

- Name: Name of the Project
- usePositionalInformationForClassification: Indicates whether positional information is used for classification or not(Boolean, default false). This option allows the ALE to use the positional information of the words during the classification step. This is typically good to use when working with structured documents, a W2 tax form for example.
- useUTF8: Passing texts in UTF-8 format or not to the engine (Boolean, default false)

A unique project ID is returned as the response

Once the project is created, a FieldDeclaration object having field name "document_class_id" needs to be set to the created project ID.

Get all Projects

This action fetches a list of all projects in ALM using `GET /learnset/project/`. Request object contains "application/json" as accept header. An array of projects in json format is returned as a successful response from ALM server.

You can append a random element to the request URL in order to prevent caching. Example: /learnset/project?_156578654332

The response JSON array containing the list of Project types has the following structures:

```
{ "id" : "...", "lastLearnedAt" : 12345, "lastModifiedAt" : 12345, "name"
: "...", "usePositionalInformationForClassification" : true, "useUTF8" :
true }
```

Create a Document Class

This action creates a new document class within a project using `PUT /learnset/project/{projectId}/class`

Request object contains the following parameters.

- projectId: ID of the project
- name: Name of the class to be created
- Request object contains "application/json" as accept header.

This returns the numeric ID of the new class as the response.

Get Project level Fields

This action fetches the field declarations for a project. The project level fields are fetched from the ALM server using `GET /learnset/project/{projectId}/fields`

Request object contains the following parameters.

- projectId: ID of the project
- Request object contains "application/json" as accept header.

It returns the array of fields(`FieldDeclaration`) objects as the response from the server. The response JSON array containing the list of FieldDeclaration types has the following structures:

```
{ "fieldId" : 12345, "name" : "...", "type" : "...", "required" : true,
"format" : "...", "constant" : "..." }
```

Configure Project level Fields

This action explicitly sets the field declarations for a project.

All the fields must be submitted. If the end user wants to add one more new field all previous fields plus the new one must be re-submitted. The array of fields always include the following default entry:

```
{ "fieldId": 1, "name": "document_class_id", "type": "Integer",
"required": true, "format": "...", "constant": "COMPANY" }
```

Depending on the number of fields the FieldId is incremented subsequently.

The request body contain an array of `FieldDeclaration` objects. The project level fields are set using `POST /learnset/project/{projectId}/fields`

The request body containing a collection of `FieldDeclaration`objects has the following structures:

```
{ "fieldId" : 12345, "name" : "...", "type" : "...", "required" : true,
"format" : "...", "constant" : "..." }
```

Get all Classes for a Project

This action fetches all classes for a specific project.

Request object contains the following parameters.

- projectId: ID of the project
- Request object contains "application/json" as accept header.

It returns an array of `DocumentClass` objects in response. The list of classes are retrieved using `GET /learnset/project/{projectId}/class`

On a successful request the response containing the JSON array of `DocumentClass`objects has the following structures:

```
{ "id" : 12345, [The id of the document class - this id is unique within a project] "name" : "...",[Name of the document class. This is just a label that is supposed to help identifying classes in a UI. It is not used by ALE itself.] "numTrainingDocuments" : 12345 [ the number of training documents that are currently available for this class. This is provided when retrieving a class or a list of classes from the server. It is not supposed to be set when creating a new class. }
```

Get Class level Fields

This action fetches all class field declarations of the project

Request object contains the following parameters.

- projectId: ID of the project
- Request object contains "application/json" as accept header.

It returns an array of `ClassFieldDeclaration` objects as response. The list of class field declarations are retrieved using `GET /learnset/project/{projectId}/classfields`

On successful request the response containing the JSON array of `ClassFieldDeclaration`objects has the following structures:

```
{ "docClassId" : 12345, Id of the document class "fieldId" : 12345, Id of the field that is assigned to the document class "name" : "...", Name that is used for this class. If none is assigned the name from the field declaration will be used. "projectId" : "..." The id of the project }
```

Check if project is Learnable

This action determines if the ALM project is learnable or not using `GET /learnset/project/{projectId}/isLearnable`

The request object contains the following parameters.

- projectId: ID of the project
- The request object contains "application/json" as accept header.

This returns a Boolean value indicating whether the project can be learned without error.

Add Training Documents

This action adds training documents from a zip file. The zip file must contain an image, a .pos file and an .ival file for each document. If one of those files is missing the document is skipped. A field declaration is generated based on the values available values in the .ival files. Missing classes are also created.

Request body contains a multi-part form data object. Request object contains the following parameters.

- projectId: ID of the project
- append: true, if append the documents from the zip files to the existing training set, false if create a new training set.
- Request object contains "application/json" as accept header.
- Content-Type: multi-part/form-data

This returns the status information for each document in response. It returns a collection of `DocumentImportStatus` objects eventually.

The request call to server happens using `POST /learnset/project/{projectId}/docs`

On successful request the response containing the JSON array of `DocumentImportStatus` objects has the following structures:

```
{ "name" : "...", [Gets the base name of the file] "errorCode" : 12345,[
Gets the error code for the file. The code is 0 if the import succeeded or
a bitwise combination of the available error code values.] "docId" : "..."
[Gets the id under which the document was stored] }
```

Get Training Document Fields

This action fetches the fields of a training document using `GET /learnset/project/{projectId}/class/{classId}/doc/{docId}/fields`

The request object contains the following parameters.

- classID: ID of the class
- docID: ID of the training document
- projectID: ID of the Project
- Request object contains "application/json" as accept header.

This returns a collection of field `FieldInfo` objects in the response.

On successful request the response containing the JSON array of `FieldInfo` objects has the following structures:

```
{ "fieldId": 12345, Id of the field as declared in FieldDeclaration object
 "location" : { Location of the field value on the document - only needs
to be set for limited learning "left" : 12345, "top" : 12345, "right" :
12345, "bottom" : 12345 }, "pageNumber" : 12345, The fields page number
"value" : "..." Value of the field. Either set a value or set word
indexes. }
```

Create a Test Set

On basis of Uploaded Contents

This action creates a set of test documents based on the uploaded content. This can either be a single document (image and .pos files) or a zip file with multiple documents.

The request object contains the following parameters.

- projectId: ID of the project
- The request object contains "application/json" as accept header.

The request body has the following details:

- Request contains the list of files(as attachments) having the media type multipart/form-data.
- Use the Content-Type:multipart/form-data HTTP header to specify this media type to the server.
- The form data contains the image and positional file information. While uploading the files, the file extension must be present. Example:file_0.png, file_1.pos . The image and .pos files are uploaded as binary to the ALM Server.

The test sets are uploaded using `POST /learnset/project/{projectId}/testset`

This returns the test document set id in response. Media-type being application/json

On basis of files located at given path

This action configures the test sets based on the documents located in a given path/file share.

Request object contains the following parameters.

- projectId: ID of the project
- path: path within the servers local file system or a file share on the network
- Request object contains "application/json" as accept header.

The test sets are uploaded using `PUT /learnset/project/{projectId}/testset` The path parameters value is appended as querystring to the request URL `path="`

It returns the test document set ID in response.

Update StreamSet

This action creates or updates a batch stream set that contains the training documents. The stream set has the same id as the project and can be referenced in the stream set service or the field extractor service.

The Request body contains the documents to learn (collection of `DocumentAdapter` objects). Request object contains the following parameters.

- projectId: ID of the project
- Request object contains "application/json" as accept header.

Streamset is updated using `HEAD /learnset/project/{projectId}/updateStreamSet`

Update Learnset

This action involves the following chain of events:

- It uploads the document along with positional data(.pos file) and value data(.ival file) to the server. While uploading the documents the path where image file is located needs to be mentioned.
- The .ival data is generated by iterating through a collection of project fields and field value combinations.
- The request body contains a multipart form data object consisting the binary data of image,.pos and .ival files.
- The files are having the naming convention like file_{0}.png, file_{1}.pos, file_{2}.ival respectively.

The documents are updated in the learnset using `POST /learnset/project/{ProjectID}/class/{ClassID}/doc`

Training a Field Extractor

A field extractor can either be trained by passing a list of documents that is learned or by learning all documents from a batch stream set.

To train an extractor by passing a list of documents

- Submit your field declaration using `POST /extractor/{id}/fields`
- Learn the documents using `POST /extractor/{id}/learn`
- Download the extractor for further use using `GET /extractor/{id}/file/extractor`

To train an extractor using a batch stream set

- Create a batch stream set using `PUT /streamset`
- Add documents to the stream set using `POST /streamset/{id}/document`
- Submit your field declaration to the extractor using `POST /extractor/{id}/fields`
- Learn the documents using `GET /extractor/{id}/streamset/{streamSetId}/learn`
- Download the extractor for further use using `GET /extractor/{id}/file/extractor`

You may also want to download the PTB and CBM files after adding documents to the stream set and uploading them at a later time rather than creating the stream set from the scratch.

Extraction

Field extraction can be done for a given document or for documents that are stored in a batch stream set.

To extract fields from a given single document

- Upload the extractor file to the server using `POST /extractor/{id}/file/extractor`
- Pass the document in a call of `/{id}/extract`

To extract fields from documents that are stored in a batch stream set

- Create or upload your batch stream set as described in the training section
- Upload the extractor file to the server using `POST /extractor/{id}/file/extractor`
- Extract fields for a document by calling `GET /{id}/streamset/{streamSetId}/extract/{docNum}`

Learn or Relearn a Project

This action involves the following chain of events:

- Check if the project is learnable using `GET /learnset/project/{ProjectId}/isLearnable`
- Check if the extractor exists for the given project ID using `HEAD /extractor/{Projectid}`
- Create a new field extractor on the server. The returned ID is required for subsequent calls that work with the extractor instance. The call is made using `PUT /extractor/id={ProjectID}&persistent=true`
- Update the stream set using `HEAD /learnset/project/{ProjectId}/updateStreamset` This creates or update the batch stream set that contains the training documents. The stream set has the same ID as the project and can be referenced in the stream set service or the field extractor service.
- Addition of fields to the extractor using `POST /extractor/{ID}/fields` where {ID} denotes Project/Extractor ID.
- Declare a set of fields for a field extractor. If no fields are declared, then it throws an error. There must be a field of type Integer with the constant value "COMPANY" present that is used as identifier of document Class.
- Learn the project using `GET /extractor/{ProjectID}/streamset/{StreamsetID}/learn` It trains a field extractor based on the documents that are stored in a stream set.

The engine learns which fields to extract for each class as defined by the passed documents. The state of the extractor is written to an extractor file that can be downloaded for future use. With relearnable flag, the user indicates that the extractor enables the relearning of classes, which stores additional information into the connected extractor stream.

The request object contains the following parameters:

- Id: ID of the extractor
- relearnable: true to enable relearning (defaults to false)
- Request object contains "application/json" as accept header.

Projects can be learned using `POST /extractor/{id}/relearn` The request body contains the following structure:

```
{ "id" : 12345, Document id "fileName" : "...", File name where the
document is located, if any. "words" : [ { List of words with positioning
information, collection of  WordInfo  objects. "pageNumber" : 12345,
"word" : "...", "boundingBox" : { "left" : 12345, "top" : 12345, "right" :
12345, "bottom" : 12345 } }, { "pageNumber" : 12345, "word" : "...",
"boundingBox" : { "left" : 12345, "top" : 12345, "right" : 12345, "bottom"
: 12345 } } ], "pages" : [ { List of pages, array of  PageInfo  objects
"rotationAngle" : 12345.0, "rotationOrigin" : 12345, "boundingBox" : {
"left" : 12345, "top" : 12345, "right" : 12345, "bottom" : 12345 } }, {
"rotationAngle" : 12345.0, "rotationOrigin" : 12345, "boundingBox" : {
"left" : 12345, "top" : 12345, "right" : 12345, "bottom" : 12345 } } ],
"fields" : [ {List of fields (only required for learning, not for
extraction), array of FieldInfo  objects. "fieldId" : 12345, "location" :
{ "left" : 12345, "top" : 12345, "right" : 12345, "bottom" : 12345 },
"pageNumber" : 12345, "value" : "..." }, { "fieldId" : 12345, "location" :
{ "left" : 12345, "top" : 12345, "right" : 12345, "bottom" : 12345 },
"pageNumber" : 12345, "value" : "..." } ], "companyFieldValue" : "..." }
```

File Types

ALM Server does not accept all image file formats. When you upload the files to the ALM Server, you need to ensure that the following supported types(.tiff, .jpg, and .png) are used.

Instance management

Any instances of batch stream sets or field extractors that are created on the server is destroyed automatically after they have not been accessed for a default duration of 30 minutes. This is a server configuration and is denoted in **alm.config.xml** under `settings` bean through the following:

entry key="expirationTime" value="30"

Error Codes

Unauthorized requests without authentication header or with an incorrect password get 403 as the response code.
Attempts to access non-existing stream sets or extractors lead to a 404 response code.
All other errors usually produce a 5xx response code.

Document Formats

Documents that are supposed to be used for training or extraction can always be passed in JSON format using the `DocumentAdapter` type with words, pages and - for training - fields.

As an alternative, .pos files and .ival files can be uploaded. A .pos file contains one line per word, using the following format:

```
wordIdx,page,left,top,width,height,word
```

.ival files are only required when creating a batch stream set for learning. They are used to provide field values for a given document. An .ival file contains one entry per line, using one of the following formats:

```
fieldname,type,value fieldname<TAB>type<TAB>value
```

The user can send fieldname, type, value, `boundingBox` and it is in the following format

```
fieldname<TAB>type<TAB>value<TAB>pageId,left,top,right,bottom For example
f1 int 4500022612 0,751,1246,980,1275
```

Changelog

Current released version is ALM 23.1.

For more details on the version history, please refer to the product Release Notes.

Resources

The resources use a data model that is supported by a set of client-side libraries that are made available on the files and libraries page.

There is a WADL document available that describes the resources API.

name	path	methods	description
BatchStreamSetService	• /streamset • /streamset/{id} • /streamset/{id}/document • /streamset/{id}/fields • /streamset/{id}/utf8 • /streamset/{id}/file/cbm • /streamset/{id}/file/ptb • /streamset/{id}/file/zip	• PUT • DELETE HEAD • POST • GET POST • GET PUT • GET HEAD POST • GET HEAD POST • GET HEAD	Creation and management of batch stream sets
FieldExtractorService	• /extractor • /extractor/{id}	• PUT	Training of field

name	path	methods	description
	• /extractor/{id}/extract • /extractor/{id}/fields • /extractor/{id}/fieldtargets • /extractor/{id}/learn • /extractor/{id}/relearn • /extractor/{id}/usePositionalInformation • /extractor/{id}/utf8 • /extractor/{id}/file/extractor • /extractor/{id}/streamset/{streamSetId}/learn • /extractor/{id}/streamset/{streamSetId}/extract/{docNum}	• DELETE HEAD • POST • GET POST • POST • POST • POST • GET PUT • GET PUT • GET HEAD POST • GET • GET	extractors and extraction of fields.
LearnSetManagerService	• /learnset/project • /learnset/ocr/available • /learnset/project/{projectId} • /learnset/project/{projectId}/check • /learnset/project/{projectId}/class • /learnset/project/{projectId}/classfields • /learnset/project/{projectId}/docs • /learnset/project/{projectId}/fields • /learnset/project/{projectId}/isLearnable • /learnset/project/{projectId}/learn • /learnset/project/{projectId}/numDocs	• GET PUT • GET • DELETE GET PUT • GET • GET PUT • GET • GET POST • GET POST • GET • GET • GET • POST PUT	

name	path	methods	description
	• /learnset/project/{projectId}/testset • /learnset/project/{projectId}/updateStreamSet • /learnset/project/{projectId}/upload • /learnset/project/{projectId}/check/extraction • /learnset/project/{projectId}/check/fieldtargets • /learnset/project/{projectId}/class/{classId} • /learnset/project/{projectId}/fields/statistics • /learnset/project/{projectId}/testset/{testSetId} • /learnset/project/{projectId}/upload/{uploadId} • /learnset/project/{projectId}/class/{classId}/doc • /learnset/project/{projectId}/class/{classId}/fields • /learnset/project/{projectId}/class/{classId}/doc/{docId} • /learnset/project/{projectId}/class/{classId}/doc/{docId}/check • /learnset/project/{projectId}/class/{classId}/doc/{docId}/class • /learnset/project/	• HEAD • POST • GET • GET • DELETE GET PUT • GET • DELETE GET HEAD POST • DELETE GET • GET POST • GET POST • DELETE GET • GET • PUT • GET POST • GET POST • GET • GET • GET POST • POST • GET • GET • GET • GET • GET • GET	

name	path	methods	description
	{projectId}/class/{classId}/doc/{docId}/fields • /learnset/project/{projectId}/class/{classId}/doc/{docId}/image • /learnset/project/{projectId}/class/{classId}/doc/{docId}/meta • /learnset/project/{projectId}/class/{classId}/doc/{docId}/pageCnt • /learnset/project/{projectId}/class/{classId}/doc/{docId}/pos • /learnset/project/{projectId}/testset/{testSetId}/doc/{docId}/class • /learnset/project/{projectId}/testset/{testSetId}/doc/{docId}/extract • /learnset/project/{projectId}/testset/{testSetId}/doc/{docId}/image • /learnset/project/{projectId}/testset/{testSetId}/doc/{docId}/meta • /learnset/project/{projectId}/testset/{testSetId}/doc/{docId}/pos • /learnset/project/{projectId}/class/{classId}/doc/{docId}/check/extraction • /learnset/project/	• GET • GET • GET • GET	

name	path	methods	description
	{projectId}/class/{classId}/doc/{docId}/check/fieldtargets • /learnset/project/{projectId}/class/{classId}/doc/{docId}/locations/fields • /learnset/project/{projectId}/class/{classId}/doc/{docId}/locations/value • /learnset/project/{projectId}/testset/{testSetId}/doc/{docId}/locations/value • /learnset/project/{projectId}/testset/{testSetId}/doc/{docId}/extract/forceLearning/{learn}		
LearnsetSchedulerService	• /learnset/projects/learn-scheduler • /learnset/projects/learn-scheduler/start • /learnset/projects/learn-scheduler/{schedulerId} • /learnset/projects/learn-scheduler/{schedulerName}	• GET • POST • DELETE • GET	

Data Types

JSON

type	description
BoundingBox	Container to carry the positional information of word.

type	description
ClassFieldDeclaration	Container to carry the information of fields of the Document Class.
DataCell	Simple container to carry extracted data string for a single cell. Used by ExtractedData.
DocumentAdapter	Carrier for document information. When training an extractor, make sure to fill the word list, the page list and the field list. When extracting fields from a document you only need to fill the word list and page list.
DocumentClass	Information about a document class
DocumentImportStatus	Container to carry the document import status information.
DocumentUploadStatus	Container to carry the uploading status information of the documents. It takes account of number of total and imported documents and their status
ExtendedClassFieldStatistics	Field statistics for a single class, covering how many values have been found at all for a field within that class and for how many of those values a target can be located.
ExtractedData	Contains the extraction result for a single field. There are usually multiple candidates which are provided as a list of DataCell.
FieldDeclaration	Declaration of a field that can be extracted.
FieldInfo	Container to carry field information.
FieldLocations	Container to carry the information of word locations respective to the field.
FieldStatistics	Basic field statistics, covering how many values exist for a given field in a project or class
LearnsetSchedulerProperties	
PageInfo	Container to carry the page orientation and positional information.
Project	Container to carry the information of Projects created in the ALM Server.
TrainingDocumentIncident	Description of a failed plausibility check on a traing document

type	description
TrainingDocumentMetaData	Meta data about a stored training document
TrainingSetCheckResult	Result of a training set plausibility check, including found incidents and field statistics.
WordInfo	Container to carry word information