Remote Matching Service

API

Version: 23.1

Written by: Documentation Team, R&D
Date: Monday, May 11, 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 Remote Matching Service (RMS) **5**
RMS API *5*
Connecting to the Server **6**
Authentication *6*
Checking for a Session **7**
Creating a Datasource **8**
Creating a Search Index **9**
Performing a Search **10**
Creating a Duplicate Report **11**
Creating an Internal User **13**
Error codes **14**
Changelog **15**
Resources **16**
Data Types **19**
JSON *19*

About Remote Matching Service (RMS)

The Remote Matching Service (RMS) provides centralized search services to applications such as web-based administration clients.

RMS is capable of supporting large search data pools.

RMS API

The RMS server runs as web application and exposes its services via a REST API which can be consumed 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 RMS Server.

However this documentation doesn't always link to the 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.

Connecting to the Server

In order to connect to RMS Server, append RMS/service to server base URL. For example the base address to communicate with the RMS server will look similar to:

```
http://{serverName}:{PortNumber}/RMS/service
```

Authentication

Most calls require authentication. Authentication information is passed in the HTTP header. Two types of authentication are supported:

- Management clients log in using `POST /session/`*`username`*. This creates a session with a unique id that is returned and needs to be passed to subsequent calls using `X-CPTMS-Session`. If the supplied user credentials are incorrect authentication will fail and a string not_authenticated is returned in the response.

```
RequestConnect.Address = "http://{serverName}:
{PortNumber}/RMS/service/session/{UserName}"; RequestConnect.Content =
Password; Response sessionId = client.SendPost(RequestConnect);
```

- Search clients can use an internal account for authentication. This does not create a user session. Instead the account name and a hash code based on the password is submitted as part of the header. To authenticate without creating a session, add a X-RMS-User header to the request that contains the name of the internal user account. Add a X-RMS-Time header to the request that contains the current time in the ISO 8601 format. Add a X-RMS-Timehash header to the request that contains a Base64 encoding of the account password and the time. To generate the timehash first a SHA256 hash must be created from a string that contains the account password plus a space character plus the current time in the ISO 8601 format. Once the SHA256 hash has been generated the value can then be encoded in the Base64 format. The time must not differ by more than 5 minutes from the server time. It is recommended that you use the server time which can be retrieved by calling GET /server/time.

-
```
httpClient.DefaultRequestHeaders.Add ("X-RMS-User", "AccountName");
string ServerTime = GetServerTime();
httpClient.DefaultRequestHeaders.Add ("X-RMS-Time", serverTime);
byte[] hash = Compute256Hash("AccountPassword" + " " + serverTime);
string base64 = Convert.ToBase64String(hash);
httpClient.DefaultRequestHeaders.Add ("X-RMS-Timehash", base64);
```

-

Checking for a Session

To check for a session a call can be made to `GET session/current`. The session id is passed in the header `X-CPTMS-Session`.

```
client.DefaultRequestHeaders.Add ("X-CPTMS-Session", sessionId);
RequestCurrentSession.Address = "http://{serverName}:
{PortNumber}/RMS/service/session/current"; Response currentSessionInfo =
client.SendGet(RequestCurrentSession);
```

The result structure of the `session/current` API method is as described below:

Session Info

name	data type	description
userName	string	Name of the user that is logged in
sessionId	string	Id of the active session
status	string	not_authenticated, not_authorized, expired, logged_in
roles	array of string	
type	number	Type of session: 0 = not logged in, 1 = technical account, 2 = user account, 3 = user account/SSO

Example

```
{
 "userName" : "...",
 "sessionId" : "...",
 "status" : "...",
 "roles" : [ "...", "..." ],
 "type" : 12345
}
```

Creating a Datasource

In order to pull record data from a database a Datasource configuration is required. To create or update a Datasource use the method POST /datasource. You will need to provide the following pieces of required information, a name for the configuration, the type of connection, the driver class, and a JDBC URL. There are two included driver classes you can use by default:

- SQL : `com.microsoft.sqlserver.jdbc.SQLServerDriver`
- ORACLE : `oracle.jdbc.driver.OracleDriver`

A JDBC URL for the driver is also required. The default JDBC URLs for SQL and Oracle are:

- SQL : `jdbc:sqlserver://`*`host`*`:`*`port`*`;databaseName=`*`databaseName`*`;responseBuffering=adaptive;selectMethod=cursor;`
- ORACLE : `jdbc:oracle:thin:@`*`host`*`:`*`port`*`:`*`sid`*

Example

```
{ name: "Datasource Name" type: "MSSQL" driverClass:
"com.microsoft.sqlserver.jdbc.SQLServerDriver" jdbcUrl:
"sqlserver://hostUI:1433;databaseName=databaseUI;responseBuffering=adaptiv
e;selectMethod=cursor;" }
```

There is also an option to use a generic driver, please see the User Guide documentation to configure a generic driver class and generic JDBC URL.

Creating a Search Index

To create a Search Index a unique name and a `SeachIndexCfg` object are required, see the documentation about the datatype for more information. With the object created use the method `POST /index/{name}/` where the name is the unique name of the index.

```
client.DefaultRequestHeaders.Add ("X-CPTMS-Session", sessionId);
 RequestCreateIndex.Address = "http://{serverName}:
{PortNumber}/RMS/service/index/{name}";
 RequestCreateIndex.Content = SearchIndexCfg.toJson()
 Response createIndexResponse = client.SendPost(RequestCreateIndex);
```

Performing a Search

Search operations can be done by using created search indexes. There are two types of search options available, i.e - candidate search and best search.

Candidate Search :

It searches for the candidate records that are contained in the supplied text. To invoke the search operation, a unique name and a `Query` object are required. See the documentation about the datatype for more information. With the object created use the method `POST /index/{name}/record/candidates` where the name is the unique name of the index.

Example

```
client.DefaultRequestHeaders.Add ("X-CPTMS-Session", sessionId);
RequestCandidateSearch.Address = "http://{serverName}:
{PortNumber}/RMS/service/index/{name}/record/candidates";
RequestCandidateSearch.Content = Query.toJson() Response
candidateSearchResponse = client.SendPost(RequestCandidateSearch);
```

Best Search :

It searches for the best candidate records that are contained in the supplied document texts. To invoke the search operation, a unique name and a `DocumentQuery` object are required. See the documentation about the datatype for more information. With the object created use the method `POST /index/{name}/record/best` where the name is the unique name of the index.

Example

```
client.DefaultRequestHeaders.Add ("X-CPTMS-Session", sessionId);
RequestBestSearch.Address = "http://{serverName}:
{PortNumber}/RMS/service/index/{name}/record/best";
RequestBestSearch.Content = Query.toJson() Response bestSearchResponse =
client.SendPost(RequestBestSearch);
```

Creating a Duplicate Report

To create a Duplicate Report for an index the method `POST /report/duplicates/{IndexName}/detection` is used. This will start the process that creates the duplicate report. If an index does not exist with the provided name a `404` error is returned. Once a duplicate report has been started the progress can be checked by using `GET /report/duplicates/{IndexName}`. The complete percentage is returned as a decimal between 0 and 1. This can be called multiple times in order to retrieve an updated percentage value. When the duplicate report has finished generating, the `GET /report/duplicates/{IndexName}/hits` method can be used to retrieve the results. The `page` and the `pageSize` parameters of this call can help tailor the amount of results returned. The full report can also be downloaded as a CSV or an Excel document using the `report/duplicates/{IndexName}/csv` or `report/duplicates/{IndexName}/xls` respectively.

Example

```
client.DefaultRequestHeaders.Add("X-CPTMS-Session", sessionId);

RequestDuplicateReport.Address = "http://{serverName}
{PortNumber}/RMS/service/report/duplicates/{indexName}/detection";

client.SendPost(RequestDuplicateReport);


ReportStatus response;

do

{

//Create a pause to limit the amount of requests to the server

RequestReportStatus.Address = "http://{serverName}:
{PortNumber}/RMS/service/report/duplicates/{indexName}";

response = client.SendGet(RequestReportStatus);

} while (response.currentStatus.Equals("in_progress"));


if (response.currentStatus.Equals("completed"))

{

if (response.numHits > 0)

{

//get the duplicates
```

```
    RequestDuplicates.Address = "http://{serverName}:
    {PortNumber}/RMS/service/report/duplicates/
    {indexName}/hits?page=1&pageSize=25";

    List hits = client.SendGet(RequestDuplicates);

}

}
```

Creating an Internal User

To create an Internal user the method `POST /user/internal/{UserName}` is used where {UserName} is the name of the user which would be passed as input. The password also needs to be passed in the Request payload section. The password needs to meet few requirement criteria in order to successfully create an internal user. Depending on cases those criteria not met an `401` (Unauthorized) error is returned along with an error code in the `X-CPTMS-ErrorCode` section of the Response Header. There can be several error codes as follows depending on which requirements the password does not meet.

PWD001 - Empty Username or Password

PWD002 - Username appearing in Password

PWD003 - The application (i.e. RMS/Remote Matching Service) or the company name appearing in Password

PWD004 - The length of the password less than 12 or more than 30 characters.

PWD005 - Not having a Upper case character in Password

PWD006 - Not having a Lower case character in Password

PWD007 - Not having a number in Password

PWD008 - Not having a special character (!@#$%^&*()=+_- or a space) in Password

PWD009 - Having 3 or more consecutive characters in Password

PWD0010 - Having the frequently used common words in Password

**Following is the list of forbidden frequently common words in the password considered as of now:

```
{
   { 123456 sunshine qwerty123 princess
   !@#$%^&* charlie 654321 donald
   Password Football welcome iloveyou
   iloveu admin abc123 aa123456 }
```

Error codes

Unauthorized requests without authentication headers or with an incorrect password receive 403 as the response code.
All other errors usually produce a 5xx response code.

Changelog

Current released version is RMS 23.1

For more details on the version history, please refer to the product Release Notes - https://docs.hyland.com/Portal_Public/Products/en/Remote_Matching_Service.htm

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
DuplicateDetectionService	• /report/duplicates • /report/duplicates/{name} • /report/duplicates/{name}/csv • /report/duplicates/{name}/detection • /report/duplicates/{name}/hits • /report/duplicates/{name}/xls	• GET • DELETE GET • GET • DELETE POST • GET • GET	Manages the creation of reports about potential duplicate records and provides access to results.
JDBCDataSourceService	• /datasource • /datasource/test • /datasource/{name} • /datasource/{name}/delete • /datasource/{name}/used	• GET POST • POST • DELETE GET • DELETE • GET	Management of JDBC data sources. JDBC data sources can be referenced when configuring a record source of type "JDBC" for a field.

name	path	methods	description
SearchIndexService	• /index • /enginetype/index • /enginetype/scoring • /import/index • /index/cfgsummary • /index/{name} • /index/{name}/dump • /index/{name}/export • /index/{name}/import • /index/{name}/rebuild • /index/{name}/recreate • /index/{name}/statistics • /index/{name}/status • /index/{name}/status-combined • /index/{name}/columns/names • /index/{name}/dump/status • /index/{name}/engine/files • /index/{name}/export/status • /index/{name}/indexer/abort • /index/{name}/indexer/status • /index/{name}/record/best • /index/{name}/record/candidates • /index/{name}/record/{id} • /index/{name}/source/columnnames • /index/{name}/statistics/{category} • /index/{name}/engine/files/{filename}	• GET • GET • GET • GET • GET • DELETE GET POST • HEAD • HEAD • POST • HEAD • HEAD • GET • GET • GET • GET • GET • GET • GET • HEAD • GET • POST • POST • GET • POST • GET • GET	Manages projects, fields and associated search indexes. Provides the search functionality. To perform a search request, use either /index/{name}/record/best
ServerInstanceService	• /server • /server/connect • /server/list • /server/starttime	• POST • HEAD • GET • GET	Manages the available server instances

name	path	methods	description
	• /server/time • /server/time-utc • /server/{id} • /server/settings/checkSSLCertificates	• GET • GET • DELETE • GET PUT	
SystemService	• /system/messages • /system/smtp • /system/messages/count • /system/notification/settings • /system/notification/types • /system/report/settings • /system/smtp/test	• GET • DELETE GET POST • GET • DELETE GET POST • GET • GET POST • POST	Manages general system settings.

Data Types

JSON

type	description
AssaScoringSettings	Scoring Configuration for the ASSA Verifier
Column	Configuration of a column. This provides information about whether or not a column is searchable or whether it is used as filter.
ColumnScoringSettings	Scoring settings for a single column. This is only used for record verifier scoring. This is not relevant for candidate search.
DocumentQuery	Query for finding records in an index that appear on a given document. This type of query triggers two searches : • One on the big search index for identifying candidates • One on a pool with just the document text for finding out which candidates are really the best
DumpStatus	Status information about the dump of the content of a search index to a file.
DuplicateRecordCandidate	Result information about a matching record.
DuplicateRecordDetectionTask	Wrapper object for containing the criteria to detect the duplicate data in the index's record source.
EnhancedJDBCDataSourceCfg	Enhancement of JDBCDataSourceCfg that include user name and password - required for testing a connection.
FieldScore	Field scoring information
Filter	Filter object consists a key-value pair , used for filtering in queries at the time of candidate or best search.
IndexerStatus	Status information about the last rebuild or recreate action on an index.
JDBCDataSourceCfg	Configuration of a JDBC data source that can be used for a

type	description
	JDBCRecordIndexer
LocalizedSystemMessage	A system message represents an important information or error message related to the status of the system itself or a component of the system.
NotificationSettings	Configuration settings for email notification
NotificationType	Refers the type of notification for rebuild status or server availability.
PerformanceStatistics	Provides statistics for a given period of time.
Query	Full query object controls most of the settings for an individual search request.
QueryResultSet	Collection of results for a single record.
RecordScoringSettings	Scoring configuration for RecordVerifier.
ReportSettings	Configures the creation of reports for third party reporting tools.
ReportStatus	Status of a previous or ongoing report creation.
SMTPAccount	Configuration for using an SMTP server to send mails.
ScoringSettings	Parent and abstract scoring configuration which can be used either for `AssaScoringSettings` or `RecordScoringSettings` implementation.
SearchIndexCfg	Configuration settings for a Search Index.
SearchIndexCfgSummary	Configuration summary instance for a search index.
SearchIndexStatus	Status information about a single search index
SearchResult	Result information about a matching record.
ServerInstanceCfg	Configuration information about a registered server instance.
TextReference	It references a section within a document text.