# Unica Collaborate V12.1.1 REST API

# Contents

# Chapter 1. Collaborate REST API

Use the Collaborate REST API to manipulate templates, folders, forms, Lists, and On-demand Campaigns.

## Overview

The API is based on REST (Representational State Transfer) principles, so you can easily write and test applications. You can use your browser to access URLs, and you can use any standard HTTP client in any programming language to interact with the API. All API calls must be entirely lowercase.

## Base URL

The base URL is `http://<hotstaname>:<port>/collaborate/api/collaborate/v1`.

Add a prefix to the base URL to each API that is listed. For example: `http://host:port/collaborate/api/collaborate/v1/lists/{project_id}/projectforms/{form_Id}`

## List of APIs

**Table 1. REST APIs for Collaborate**

| | |
|---|---|
| **Templates** | |
| get: /templates | Lists all templates for the specified template type. |
| get: /templates/{templateId} | Finds a template by the specified ID. |
| **Template folders** | |
| get: /templatefolders | Lists all folders used for templates. |
| get: /templatefolders/{folderId} | Finds a folder by ID. |
| post: /templatefolders | Creates a new folder. |
| put: /templatefolders/{folderId} | Updates details of a folder. |

| | |
|---|---|
| delete: /templatefolders/{folderId} | Deletes an existing folder. |
| get: /templatefolders/{folderId}/templates | Gets a list of all templates from the specified folder. |
| get: /templatefolders/{folderId}/templates/{templateId} | Finds a template by ID. |
| get: /templatefolders/{parentFolderId}/subfolders | Gets subfolders from the parent folder. |
| post: /templatefolders/{parentFolderId}/subfolders | Creates a subfolder under the parent folder. |
| **Forms** | |
| get: /forms | Lists all forms. |
| get: /forms/{formId} | Finds a form by using a form ID. |
| get: /forms/{formId}/attributes | Lists all form attributes for the specified form ID. |
| get: /forms/{formId}/attributes/{attrID} | Fetches a form attribute based on the attribute ID and form ID. |
| get: /forms/{formId}/offers | Lists all offers for a form ID. |
| get: /forms/{formId}/offers/{offerID} | Fetches an offer for an offer ID. |
| **Form folders** | |
| get: /formfolders | Lists all folders used for forms. |
| get: /formfolders/{folderId} | Finds a folder by ID. |
| post: /formfolders | Creates a new folder. |
| put: /formfolders/{folderId} | Updates the details of a folder. |
| delete: /formfolders/{folderId} | Deletes an existing folder. |

| | |
|---|---|
| get: /formfolders/{folderId}/forms | Gets a list of all forms from the specified folder. |
| get: /formfolders/{folderId}/forms/{formId} | Finds a form by using the form ID. |
| get: /formfolders/{folderId}/forms/{formId}/attributes | Lists all form attributes for the specified form ID. |
| get: /formfolders/{folderId}/forms/{formId}/attributes/{attrID} | Fetches a form attribute based on the attribute ID. |
| get: /formfolders/{folderId}/forms/{formId}/offers | Lists all offers IDs for the specified form ID. |
| get: /formfolders/{folderId}/forms/{formId}/offers/{offerID} | Fetches an offer for the specified offer ID. |
| get: /formfolders/{parentFolderId}/subfolders | Gets subfolders from the parent folder. |
| post: /formfolders/{parentFolderId}/subfolders | Creates a subfolder under the parent folder. |
| **Lists** | |
| get: /lists | Lists all simple Lists. |
| post: /lists | Creates a new List campaign. |
| get: /lists/{projectId} | Finds a simple List. |
| put: /lists/{projectId} | Updates the standard attributes of a simple List campaign. |
| put: /lists/{projectId}/changestate | Updates the status of a simple campaign. |
| get: /lists/{projectId}/projectforms | Lists all forms associated with a project. |
| get: /lists/{projectId}/projectforms/{formId} | Gives details about a form and current values of custom attributes. |

| | |
|---|---|
| put: /lists/{projectId}/ projectforms/{formId} | Updates the values of custom attributes and offers. |
| get: /lists/{projectId}/ workflowrows | Lists all workflow tasks and stages from a project. |
| get: /lists/{projectId}/ workflowrows/{rowId} | Finds a workflow task or a stage. |
| post: /lists/{projectId}/ workflowrows/{rowId}/run | Runs a flowchart. |
| get: /lists/{projectId}/ workflowrows/{rowId}/run/{jobId} | Gives run results of a flowchart run by using the job ID. |
| get: /lists/{projectId}/recurrence | Gets a recurrence configuration for a List. |
| put: /lists/{projectId}/recurrence | Updates a recurrence configuration for a List. |
| get: /lists/{projectId}/attachments | Lists all attachments from a project. |
| post: /lists/{projectId}/ attachments/ | Saves an attachment based on details that are passed, and returns a JSON object containing details of the attachment. |
| delete: /lists/{projectId}/ attachments/{attachmentId} | Deletes an attachment based on the attachment ID passed. |
| get: /lists/{projectId}/ attachmentfolders | Lists all the attachment folders from a project. |
| get: /locals/{projectId}/ attachmentfolders/{folderId} | Returns details of an attachment based on the folder ID passed. |
| **Advanced Lists** | |
| get: /advlists | Lists all advanced Lists. |
| post: /advlists | Creates a new advanced List. |
| get: /advlists/{projectId} | Finds an advanced List. |

| | |
|---|---|
| put: /advlists/{projectId} | Updates the standard attribute values of an advanced List. |
| put: /advlists/{projectId}/changestate | Updates the status of an advanced List. |
| get: /advlists/{projectId}/projectforms | Lists all forms associated with a project. |
| get: /advlists/{projectId}/projectforms/{formId} | Gives details about a form and the current values of custom attributes. |
| put: /advlists/{projectId}/projectforms/{formId} | Updates values of custom attributes and offers. |
| get: /advlists/{projectId}/workflowrows | Lists all workflow tasks and stages from a project. |
| get: /advlists/{projectId}/workflowrows/{rowId} | Finds a workflow task or a stage. |
| post: /advlists/{projectId}/workflowrows/{rowId}/run | Runs a flowchart. |
| get: /advlists/{projectId}/workflowrows/{rowId}/run/{jobId} | Gives run results of a flowchart run by using the job ID. |
| get: /advlists/{projectId}/recurrence | Gets a recurrence of an advanced List. |
| put: /advlists/{projectId}/recurrence | Updates a recurrence of an advanced List. |
| get: /advlists/{projectId}/attachments | Lists all attachments from a project. |
| get: /advlists/{projectId}/attachments/{attachmentId} | Returns an attachment valid for the ID as a downloadable file |

| | |
|---|---|
| post: /advlists/{projectId}/ attachments/ | Saves an attachment based on details that are passed, and returns a JSON object containing details of the attachment. |
| delete: /advlists/{projectId}/ attachments/{attachmentId} | Deletes an attachment based on the attachment ID passed. |
| get: /advlists/{projectId}/ attachmentfolders | Lists all the attachment folders from a project. |
| get: /advlists/{projectId}/ attachmentfolders/{folderId} | Returns the details of an attachment based on the folder ID passed. |
| **On-demand Campaigns** | |
| get: /locals | Lists all On-demand Campaigns. |
| post: /locals | Creates a new On-demand Campaign. |
| get: /locals/{projectId} | Finds an On-demand Campaign. |
| put: /locals/{projectId} | Updates standard attributes of an On-demand Campaign. |
| put: /locals/{projectId}/ changestate | Updates the status of an On-demand Campaign. |
| get: /locals/{projectId}/ projectforms | Lists all forms associated with a project. |
| get: /locals/{projectId}/ projectforms/{formId} | Gives details about a form and the current values of custom attributes. |
| put: /locals/{projectId}/ projectforms/{formId} | Updates values of custom attributes and offers. |
| get: /locals/{projectId}/ workflowrows | Lists all workflow tasks and stages from a project. |

| | |
|---|---|
| get: /locals/{projectId}/ workflowrows/{rowId} | Finds a workflow task or a stage. |
| post: /locals/{projectId}/ workflowrows/{rowId}/run | Runs a flowchart. |
| get: /locals/{projectId}/ workflowrows/{rowId}/run/{jobId} | Gives run results of the flowchart run by using the job ID. |
| get: /locals/{projectId}/recurrence | Gets a recurrence of an On-demand Campaign. |
| put: /locals/{projectId}/recurrence | Updates a recurrence of an On-demand Campaign. |
| get: /locals/{projectId}/ attachments | Lists all attachments from a project. |
| get: /locals/{projectId}/ attachments/{attachmentId} | Returns an attachment that is valid for the ID as a downloadable file. |
| post: /locals/{projectId}/ attachments/ | Saves an attachment based on details that are passed, and returns a JSON object containing details of the attachment. |
| delete: /locals/{projectId}/ attachments/{attachmentId} | Deletes an attachment based on the attachment ID passed. |
| get: /locals/{projectId}/ attachmentfolders | Lists all the attachment folders from a project. |
| get: /locals/{projectId}/ attachmentfolders/{folderId} | Returns the details of an attachment based on the folder ID passed. |

## HTTP Status Codes

The Collaborate REST APIs attempt to return appropriate HTTP status codes for every request. For example:

200 - OK

201 - New object has been created at the server. Check the Location response header to get the URI of that object.
204 - Update/Delete request has been completed successfully.
400 - Bad Request (Fix the error before resubmitting.)
401 - User does not have permission to do this action. Unauthorized request.
404 - Requested object not found.
422 - Invalid data is provided in the request.
500 - Internal Server Error (Check `<Collaborate_home>/logs/system.log` on the Collaborate web application server for details.)
404 - Not Found

Many online resources provide information about HTTP status codes.

## Security framework for HCL Unica APIs

Platform provides the security framework for the APIs implemented by HCL Unica products.

A set of configuration properties on the **Settings > Configuration** page enables developers to set the following security for the APIs provided by HCL Unica products.

- For a specific product API, you can block access to the product.
- For a specific product API, you can require HTTPS for communication between the specified API and the product.
- For a specific product API, you can require authentication for communication between the specified API and the product.

The configuration properties that control API security are located under the **Unica Platform | Security | API management** category. Each product has a configuration property template that you can use to create new security settings for the APIs provided by that product.

You can set and change the security settings for an API as appropriate for unit testing or deployment or during the overall lifecycle of APIs.

The security framework currently supports APIs for Campaign only.

The Platform security framework supports the following two authentication options for accessing protected APIs. You can use either one, depending on your environment.

- Internal users who are registered with Platform can be authenticated using their Platform login credentials to obtain a secure token.
- External users who are part of a federation that Platform is set up to use can be authenticated though the Identity Provider server.

## Internal user authentication with the Platform login API

To authenticate internal users in client applications, use the Platform `login` API to generate secure tokens. You can then invoke any protected APIs by passing the required parameters in the request header, in addition to the parameters expected by the API itself.

The security filter intercepts these protected requests, validates them, and then passes them through for processing.

After the Platform user is authenticated, the Platform security filter adds the user's login name to the request as an attribute of the `USER_NAME_STRING` key before passing it to the product for processing.

The secure tokens have a default life span of 15 seconds. After the life span of the token expires, it cannot be used to invoke a protected API. Each time the Platform `login` API is invoked for a user, all previous security tokens for that user are invalidated.

You can change the life span of secure tokens by setting the value of the **Token lifetime** property located on the **Settings > Configuration** page under the **General | Miscellaneous** category.

**Example URL**

`http[s]://host:port/unica/api/manager/authentication/login/`

**Header parameters**

**Table 2. Header parameters for the login API with internal users**

| Parameter | Description |
|---|---|
| `m_user_name` | The internal user's Platform login name. |

| Parameter | Description |
|---|---|
| `m_user_password` | The internal user's Platform password in plain text. |

**Response**

When login succeeds, the response is HTTP 200 with the following JSON data.

- `m_tokenId` - randomly generated token
- `m_user_name` - user name of the logged in user
- `createDate` - timestamp in the format that is shown in the following example, where the time zone is IST:

  ```
  Mon Jul 06 18:23:35 IST 2015
  ```

When login fails with bad credentials, the response is HTTP 401 (unauthorized). When the `login` API is configured to be blocked, the response is 403 (forbidden). When the `login` API is configured to use HTTPS and if it is invoked on HTTP, the response is 403 (forbidden).

To log out internal users, use the Platform `logout` API.

## Internal user logout with the Platform logout API

Use the Platform `logout` API to log out internal users and delete the secure token.

The `logout` API is protected by default. The authentication parameters are expected in the request header against pre-defined keys.

**Example URL**

`http[s]://`*`host`*`:`*`port`*`/unica/api/manager/authentication/logout/`

**Header parameters**

**Table 3. Header parameters for the logout API**

| Parameter | Description |
|---|---|
| `m_user_name` | The internal user's Platform login name. |
| `m_tokenId` | The secure token obtained through authentication. |

| Parameter | Description |
|---|---|
| `api_auth_mode` | Use the value `manager` for internal users. |

**Response**

When authentication succeeds, the response is `HTTP 200`, and the secure token is deleted. If the response is HTTP 200, the client application should confirm the logout.

When authentication fails, the response is `HTTP 401`.

## External user authentication and logout through a federation

When Platform is integrated with a supported federation, users can log in to their own system, and the client application gets a token through the Identity Provider (IdP) server provided by Platform.

After a federated user is authenticated, their corresponding Platform login name is added to the request as an attribute of the `USER_NAME_STRING` key.

Log out should be done at the IdP server.

**Header parameters**

The following table describes the header parameters to use when authenticating through the IdP server provided by Platform.

**Table 4. Header parameters with a federation**

| Parameter | Description |
|---|---|
| **f_userId** | User ID in the federation. |
| **f_clientId** | Client ID in the federation. |
| **f_spId** | Service provider ID in the federation. |
| **f_tokenId** | Single sign-on token from the IdP server. |
| **api_auth_mode** | Use the value `fsso` for federated authentication. |

**Response**

The response is `HTTP 200`, with additional items depending on the API.