ANARK COLLABORATE API REFERENCE

CONTENTS

GENERAL NOTES

All API calls require authentication and authorization to succeed. Authentication checks that the credentials passed with the API correspond to an active system user. Authorization checks that the user invoking the API has permission to invoke that specific API.

Authentication

Basic authentication is recommended; however, cookie authentication is also supported. If using basic authentication, base64 encoded user credentials should be passed using the "Authorization" request header, with every API request.

```
Authorization: Basic username:password
Ex: Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

To use cookie authentication, a session needs to be established by using the /api/auth/login API. After the API calls are done, use /api/auth/logout to logout the user. "mbewebsid" cookie needs to be passed during the API request using the Cookie request header.

Authorization

Authorization checks that the user invoking the API has a specific role permission. Each API requires a different role permission and is listed in the API description. It is recommended to create a custom role with the required role permission(s) and give that custom role to the user that will be invoking the API. Anark Collaborate has built-in roles that may have the required permission.

API Requests and Responses

Most APIs must be sent with a request body. Request bodies should be sent as JSON objects, unless otherwise specified.

```
body: {
    …,
}
```

All APIs will return a status code indicating success or non-success. If any API returns a non-success status code, the response body should be an error object with the message property set to the error message.

SYSTEM PERMISSIONS

The following table details each permission required by at least one of the APIs in this document, and what default role has this permission. NOTE: These permissions do not over-rule any Access Control specifications for given content.

Permission	Description	Default Roles with Permission
VIEW-CONTENTITEM	Can view published content.	Collaborator, Content Author, Viewer
VIEW-ACTIVITY	Can view conversations and comments in the feed.	Collaborator, Viewer
VIEW-WORK-ITEM	Can view work items.	Viewer, Work Item Author
VIEW-SEARCH-CONTENTITEM	Can search for content.	Admin, Collaborator, Content Author, Viewer
VIEW-SEARCH-USER	Can search for users.	Admin, Collaborator, Viewer
VIEW-SEARCH-ACTIVITYLIST	Can search for activity lists.	Admin, Collaborator, Viewer
VIEW-SEARCH-ACTIVITY	Can search for activities.	Admin, Collaborator, Viewer
VIEW-SEARCH-GROUP	Can search for Access Control Lists.	Admin
VIEW-SEARCH-WORKINSTRUCTION	Can search for work instructions.	Viewer
VIEW-SEARCH-WORK-ITEM	Can search/query for work items.	Viewer, Work Item Author, Admin
CREATE-UPDATE-DELETE-ACTIVITY	Can manage activity.	Collaborator
CREATE-UPDATE-DELETE-CONTENTITEM	Can publish content.	Content Author
CREATE-UPDATE-DELETE-USER	Can add and manage users.	Admin
CREATE-UPDATE-DELETE-WORK-ITEM	Can add and manage work items.	Work Item Author, Admin
ADMINISTER-PREFERENCES	Can manage system preferences.	Admin

PUBLISHING A CONTENT ITEM

To publish a content item to Anark Collaborate:

1) Create a content item
2) Add one or more component(s) to the content item. Hierarchy is supported, any components can contain children, so components can be added in a tree like structure.
3) Use the upload API to publish content targeting a specific component in the content item.

The user who is invoking these APIs should have the "CREATE-UPDATE-DELETE-CONTENTITEM" role permission.

CREATE A CONTENT ITEM

POST /api/contents

Creates a content item with metadata (properties). The user who is invoking this API should have "CREATE-UPDATE-DELETE-CONTENTITEM" role permission.

Request:

Property	Property Description
template	template id
name	content name
description *Optional*	content description
properties *Optional*	content properties defined by custom schema

Example request:

```
{
  "template": "612d6178637aa6009400e8ec",
  "name": "sample content",
  "description": "sample description",
  "properties": {
    "revision": "A",
    "version": 3
  }
}
```

If the API call is successful, it will return a status code 200 with the content id and nonce id in the response body.

Response:

Property	Property Description
contentId	content unique identifier
nonceId	content nonce id (this is required for editing the content item)

Example response:

```
{
  "contentId": "612d6178637aa6009400e8ec",
  "nonceId": "612d6178637a16009400f9fd"
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

ADD A COMPONENT TO THE CONTENT ITEM

POST /api/cicomponents

Adds a component to a content item. The user who is invoking this API should have the "CREATE-UPDATE-DELETE-CONTENTITEM" role permission.

Request:

Property	Property Description
content	ID of the content to which the component should be added
name	Component name
type	Component type - document, model, image, video, file, or data.
parent *Optional*	ID of the parent component if adding as a child. If not specified, adds the component to the root
index *Optional*	The index at which to insert the component relative to sibling components. Indexes are "zero" based, meaning if there are 4 existing and you create a new component at index 1, then the new component will be created in the *second* position.
visible *Optional*	Whether or not the component is viewable, i.e., can be visualized in a viewer. If not specified, default values will be applied. For File type, default is false. For all others, default is true.

Example request:

```
{
  “content”: "612d6178637aa6009400e8ec”,
  “name”: “sample component”,
  “type”: “document”,
  “visible”: true
}
```

Components can be one of the following types: model (3D), document (PDF), image (JPG, PNG, TIF or BMP), video (MP4), file (used for attachments – any file extension other than JS or EXE), or data (JSON). If the API call is successful, it will return a status code 200 with the component id in the response body.

Response:

Property	Property Description
cicomponentId	component id

Example response:

```
{
  "cicomponentId": "612d6178637aa6009400e8ec",
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

UPLOAD FILE AND PUBLISH

POST /api/upload/file/{contentId}

Uploads a file and starts the publishing process asynchronously. If a 3D model assembly is being uploaded, the user should create a zip file before uploading. “ContentId” in the URL is the unique identifier of the content for which file is being uploaded. Request body should be of type multipart/form-data and contains the fields as shown in table below. This will overwrite the content if it already exists.

The user who is invoking this API should have either the “CREATE-UPDATE-DELETE-USERFILE” or “CREATE-UPDATE-DELETE-ACTIVITYFILE” or “CREATE-UPDATE-DELETE-CONTENTITEM” role permission.

Field	Value
fileData	File blob data
componentId	Component id
componentType	Component type
assemblyName *Optional*	Top level assembly name (required only in the case of 3D assembly upload)

recipeArgs *Optional*	Recipe arguments that need to be overridden – passed as a string (JSON stringify the array).

If specifying recipe arguments (recipeArgs), each recipe argument should be an object like shown below. “actionType”, “optionKey”, “optionValue” are optional fields, but one of them should be specified. “replacementOptionValue” is required.

```
const recipeArgs = JSON.stringify([{
  “optionKey”: “Anark.Core.CadAdapter.Import.Pdf2d.RotateAngle”,
  “replacementOptionValue”: “Zero”
},
...,
]);
```

Example:

```
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="fileData"; filename="/C:/upload/testcontent.zip"
Content-Type: <Content-Type header here>
(data)
----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="componentId"
60465cc59d46470071d4f1d8
----WebKitFormBoundary7MA4YWxkTrZu0gW
----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="componentType"
model
----WebKitFormBoundary7MA4YWxkTrZu0gW
----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="assemblyName"
car.CATProduct
----WebKitFormBoundary7MA4YWxkTrZu0gW
----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="recipeArgs"
'[{"optionKey":"Anark.Core.CadAdapter.Import.Pdf2d.RotateAngle","replacementOptionValue
":"Zero"}]'
----WebKitFormBoundary7MA4YWxkTrZu0gW
```

If the API call is successful, it will return a status code 200 with the job id in the response body. Jobs service status API should be used to poll periodically to get the status of the job.

Response:

Property	Property Description
jobId	job id (used for status updates)

Example response:

```
{
  "jobId": "xcyuioVB",
}
```


Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

GET THE JOB STATUS

GET /api/jobs/{jobId}/status

"JobId" in the URL is the unique identifier of the job for which status is being queried.

If the API call is successful, it will return a status code 200 with the job status in the response body. If there is an error when processing the job, the message property will contain the error message.

Property	Property Description	Options
status	job status	'queued | complete | processing | failed | cancelled'
message *Optional*	error message	

Example response:

```
{
  "status": "complete",
}
```

Response Status Codes:

200 Success

401 Unauthorized (authentication failed)

403 Forbidden (user does not have the required role permission)

404 Not Found (job not found)

500 Internal Server Error

CLEANUP ON PUBLISH ERRORS

DELETE /api/contents/{contentId}/publish

This needs to be invoked only if either, content item creation or addition of component to a content item, succeeded. “ContentId” in the URL is the unique identifier of the content for which publishing resulted in an error.

If there are any errors during the publish API calls, call this API to roll back the changes. It will delete the entries from database and files from content store if needed.

Request:

Property	Property Description
action	action can be ‘post’, ‘append’, or ‘overwrite’
componentIds *Optional*	list of component ids. Ids can be obtained from POST /api/cicomponents response body

Example request:

```
{
  "action": "post",
  "componentIds": ["78666ac968841279e86c91df", "7ac40968841279e86c601965"]
}
```

Response Status Codes:

204 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

VIEW URL

The URL to view the content item will be of the form shown below. Content ID is returned from the create content item API call.

https://{ANARK_ SERVER}/view/{CONTENTID}/

CONTENT MANAGEMENT

DELETE A CONTENT ITEM

DELETE /api/contents/{contentId}

Deletes the content item from the database and content store. “ContentId” in the URL is the unique identifier of the content which is being deleted. The user who is invoking this API should have the “CREATE-UPDATE-DELETE-CONTENTITEM” role permission.

Response Status Codes:

204 Success

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (content item does not exist)

500 Internal Server Error

DELETE A COMPONENT

DELETE /api/cicomponents/{cicomponentId}

Deletes the component from the database and content store. Removes component from the parent component. “CIComponentId” in the URL is the unique identifier of the component which is being deleted. The user who is invoking this API should have the “CREATE-UPDATE-DELETE-CONTENTITEM” role permission.

Response Status Codes:

204 Success

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (component does not exist)

500 Internal Server Error

GET COMPONENTS

GET /api/cicomponents?content={contentId}

Gets all components of a content item. “ContentId” in the URL is the unique identifier of the content. The user who is invoking this API should have the “VIEW-CONTENTITEM” role permission.

Response is an array of objects with the following properties:

Property	Property Description
name	component name
type	component type
content	content id
children	array of cicomponents with the same properties in this table.
thumbnailUrl	a url to a thumbnail representing this component

Example response:


```
{
  "_id": "61e994daa70a386c96478b9c",
  "name": "sample component",
  "type": "document",
  "content": "62141e2f565da10f983a2936",
  "children": [{
    "_id": "62141e2f565da10f983a2942",
    "children": [{
      "_id": "62141e2f565da10f983a2943",
      "children": [],
      "name": "component10",
      "type": "document",
      "content": "62141e2f565da10f983a2936",
      "thumbnailUrl": "/content/62141e2f565da10f983a2936/62141e2f565da10f983a2943/thumbnail"
    }],
    "name": "component9",
    "type": "data",
    "content": "62141e2f565da10f983a2936",
    "thumbnailUrl": "/content/62141e2f565da10f983a2936/62141e2f565da10f983a2942/thumbnail"
  }],
  "thumbnailUrl": "/content/62141e2f565da10f983a2936/61e994daa70a386c96478b9c/thumbnail",
}
```


Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (content access check failed or role permission check failed)

500 Internal Server Error

GET CONTENT ITEM METADATA

GET /api/contents/{contentId}/properties

Retrieves the content item metadata. “ContentId” in the URL is the unique identifier of the content for which properties need to be retrieved. The user who is invoking this API should have the “VIEW-CONTENTITEM” role permission.

If the API call is successful, it will return the following response body with status code 200.

Property	Property Description
name	content name
description	content description
archived	true if content is archived, otherwise false
creationdate	creation timestamp
lastmoddate	last modification timestamp
lastmodifiedby	user id of the user who last modified the content
properties	properties defined by custom schema
deleted	true if the content was deleted, otherwise false

Example response:

```
{
  "name": "sample content",
  "description": "sample description",
  "archived": false,
  "creationdate": "2021-04-08T16:37:31.192Z",
  "lastmoddate": "2021-04-08T16:37:31.192Z",
  "properties": [
    {
      "revision": "A",
      "version": 1.0
    }
  ],
  "deleted": false,
}
```

Response Status Codes:

200 Success

401 Unauthorized (authentication failed)

403 Forbidden (content access check failed or role permission check failed)

404 Not Found (content does not exist)

500 Internal Server Error

GET /api/contents/{contentId}/properties/{propKey}

Use this API to retrieve a specific property of content item (name, description, archived, deleted, creationdate, lastmoddate, lastmodifiedby, and custom properties). “ContentId” in the URL is the unique identifier of the content for which property needs to be retrieved. “PropKey” is the property key name. The user who is invoking this API should have the “VIEW-CONTENTITEM” role permission.

If the API call is successful, it will return the following response body with status code 200.

Property	Property Description
value	property value

Example response:

```
{
  "value": "sample part"
}
```

Response Status Codes:

200 Success

401 Unauthorized (authentication failed)

403 Forbidden (content access check failed or role permission check failed)

404 Not Found (content does not exist)

500 Internal Server Error

EDIT CONTENT ITEM METADATA

PUT /api/contents/{contentId}/properties

Edits the content item metadata. “ContentId” in the URL is the unique identifier of the content for which properties needs to be modified. The user who is invoking this API should have the “CREATE-UPDATE-DELETE-CONTENTITEM” role permission. Request body should contain the content item properties that needs to be modified.

Request:

Property	Property Description
name *Optional*	content name
description *Optional*	content description
propertyKey *Optional*	content property(ies) defined by custom schema, specified in key-value pairs

Example request:

```
{
  "name": "sample name"
  "description": "sample description"
  "isLatestVersion": "true"
```


```
}
```

Response Status Codes:

204 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (content does not exist)

500 Internal Server Error

PUT /api/contents/{contentId}/properties/{propKey}

Use this API to modify a specific property (name, description, and custom properties). “ContentId” in the URL is the unique identifier of the content for which property needs to be modified. “PropKey” is the property key name. The user who is invoking this API should have the “CREATE-UPDATE-DELETE-CONTENTITEM” role permission.

Request body is of type text and will contain the new value for the property.

Example:

```
let xhr = new XMLHttpRequest();
xhr.open("PUT", "/api/content/62141e2f565da10f983a2936/properties/description");
xhr.send("sample description");
```

Response Status Codes:

204 Success

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (content does not exist)

500 Internal Server Error

ARCHIVING A CONTENT ITEM

PUT /api/contents/{contentId}/archived

Archives or unarchives a content item. Archived items remain in the system but cannot be edited. “ContentId” in the URL is the unique identifier of the content. The user who is invoking this API should have the “CREATE-UPDATE-DELETE-CONTENTITEM” role permission.

Request body should be JSON with key “value” equal to true or false. If true, the content will be marked as archived.

Response Status Codes:

204 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (content does not exist)

500 Internal Server Error

DOWNLOADING A CONTENT ITEM

Components of type document, image, and file can be downloaded. Downloading components of a content item is done asynchronously and requires using three API calls as outlined below.

POST THE DOWNLOAD REQUEST

POST /api/contents/{contentId}/download

Starts the download request asynchronously. “ContentId” in the URL is the unique identifier of the content which needs to be downloaded. Request body will be an array of unique identifiers for the components that needs to be downloaded. The user who is invoking this API should have the “VIEW-CONTENTITEM” role permission.

Request:

Body	Description
ids: ['60566ac968841279e86c9130',..]	component ids to be downloaded

Example request:

```
ids: ["78666ac968841279e86c91df", "7ac40968841279e86c601965"]
```

Response:

Property	Property Description
jobid	job id (used for status updates)

Example response:

```
{
  "jobid": "xctyuiHJ"
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (content access check failed, or user does not have the required role permission)

404 Not Found (no file to download)

405 Content download is not enabled

500 Internal Server Error

GET THE JOB STATUS

GET /api/jobs/{jobId}/status

Getting the job status in download is the same API as getting the job status while Publishing a Content Item. Please refer to the above API for reference: Get the Job Status

TRIGGER THE DOWNLOAD OF THE FILE

GET /api/download/file?jobid={jobId}

When the job status returns “complete,” trigger the download of the file using the above API. If there are multiple components, it will be a zip file (download.zip), otherwise it will be a single file. “JobId” in the URL is the unique identifier of the job which was returned when posting the download request.

If the API call is successful, it will return a status code 200 with the response body containing the file blob data and Content-Disposition response header will be set to “attachment”.

Response Status Codes:

200 Success

401 Unauthorized (authentication failed)

403 Forbidden (content access check failed, or invalid role permission)

404 Not Found (file to be downloaded, not found)

WORK MANAGEMENT

CREATE OR UPDATE A WORK ITEM

POST /api/work-items

Creates a work item in the system. The user who invokes this API should have the “CREATE-UPDATE-DELETE-WORK-ITEM” role permission.

Request:

Property	Property Description
title	work item title
description	work item description
type	enum: task, issue, review
state	enum: proposed, in-progress, resolved, completed, removed
priority *Optional*	enum: low, medium, high
dueDate *Optional*	date the item is due – serialized in ISO-8601 format
activity *Optional*	ID of the Activity associated with this work item. Once an activity has been assigned, it cannot be reassigned or removed.
assignee.item *Optional*	ID of the User or Group to which this work item is assigned. Must also set the `assignee.modelName`.
assignee.modelName *Optional*	enum: MBEUser, MBEGroup NOTE: this field is required if `assignee.item` is set
contents *Optional*	Array of content IDs which to associate with this work item.

relatedItems
Optional
	Array of related items with the following schema:
	{
	"itemId": ObjectId(),
	"modelName": enum("MBEWorkItem", "MBEConversation")
	}

Example request:

```
{
  "title": "Sample work-item title",
  "description": "sample description",
  "type": "task",
  "state": "proposed",
  "author": "612d6178637aa6009400e8ec",
  "priority": "low",
  "dueDate": "2024-06-11T22:12:55.983Z",
  "activity": "612d6178637aa600ed979400",
  "assignee": {
    "item": "612d6178637aa6009400ed97",
    "modelName": "MBEUser"
  },
  "contents": ["612d6178637aa6009400def0", "612d6178637aa6009400abcd"],
  "relatedItems": [
    {
```

```
      "itemId": "612d6178637aa6009400feda",
      "modelName": "MBEWorkItem"
    }
  ]
}
```

If the API call is successful, it will return a status code 200 with the work-item id in the response body.

Response:

Property	Property Description
workItemId	work item unique identifier

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

PUT /api/work-items/{workItemId}

The request to update a work item is the same as creating a work item. Response will include the work item object.

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (work item not found)

500 Internal Server Error

GET A WORK ITEM

GET /api/work-items/{workItemId}

Retrieves the work item information. “WorkItemId” in the URL is the unique identifier of the work item for which properties need to be retrieved. The user who invokes this API should have the “VIEW-SEARCH-WORK-ITEM” role permission or have “VIEW-WORK-ITEM” and be a member of the activity associated with the work item.

Response:

Property	Property Description
title	work item title
description	work item description
type	enum: task, issue, review
state	enum: proposed, in-progress, resolved, completed, removed
priority	enum: low, medium, high
dueDate	date the item is due – serialized in ISO-8601 format
activity	ID of the Activity associated with this work item. Once an activity has been assigned, it cannot be reassigned or removed.
assignee	The User or Group to which this work item is assigned.
comments	All comments that have been added to the work item.
contents	Array of content IDs which to associate with this work item.
relatedItems	Array of related items.
createdBy	The User that created the work item.

Example response:

```
{
  "title": "Sample work-item title",
  "description": "sample description",
  "type": "task",
  "state": "proposed",
  "author": "612d6178637aa6009400e8ec",
  "priority": "low",
  "dueDate": "2024-06-11T22:12:55.983Z",
  "activity": "612d6178637aa600ed979400",
  "assignee": {
    "item": { … },
    "modelName": "MBEUser"
  },
  "contents": ["612d6178637aa6009400def0", "612d6178637aa6009400abcd"],
  "relatedItems": [
    {
      "itemId": "612d6178637aa6009400feda",
      "modelName": "MBEWorkItem"
    }
  ],
  "createdBy": { … }
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (work item not found)

500 Internal Server Error

SEARCH SERVICES

By default, contents, activities, work instructions and users can be searched by its name or title. A user with the Admin role ("ADMINISTER-PREFERENCES" role permission) can configure the searchable item properties in System Preferences. Searchable properties can include the description (content, work instruction and activity), email and userid (users), or all the string metadata properties specified in custom schema being used for content item or work instruction metadata. The Admin can also specify search policies requiring a minimum number of characters for keyword search.

All queries should be URL Encoded so they are parsed correctly by the server. This can be programmatically by using the encodeURIComponent() JavaScript function. In the examples below everything is in plain text for readability.

```
`/api/search/<type>?query=${encodeURIComponent('"blue pump"')}`
```

A search query can be constructed to lookup general keyword and specific starts with phrases. All keywords are implicitly search OR. For example, a search query of <blue pump> will search any indexed searched properties that contain "blue" or "pump". To augment this behavior you can include AND, OR, and parentheticals () that support basic logic and order of operations. To search for full phrases, you can wrap terms in quotes. A search for <"blue pump"> would search for index field that starts with the phrase "blue pump".

```
blue pump
"blue pump"
"blue pump" AND red
"blue pump" AND (red OR yellow)
```

Column names can be specified by using brackets. To search for only the name of a document the query would be <[name] = "blue pump"> A term should be surrounded by quotes in order to be included in the column search. Without quotes around "blue pump" it would query for where name equals blue or any keyword that matches pump.

```
[name] = "blue pump"
[description] = "high pressure"
```

Furthermore, a user can define wildcards to search for partial keywords. A * operator means 0 or more characters and a ? operator means any character, but exactly 1 . For example, a search term of <[name] = blue*> will return results of any documents with names that start with the word blue

```
[name] = "blue pump"
[description] = "high pressure"
```

Custom sorting is supported, by default all search results show the most recently added items at the top. To change the order of the results you can specify <SORTASC [fieldName]> or <SORTDSC [fieldName]> to the end of any query.

```
[DocID] = 0013335B-* AND [Description] = "0013335B-12 ABC" SORTASC [name]
```


For optional performance all search api's use cursor-based pagination to every api result will have the following data structure:

```
{
  "prevPage": "/api/search/<type>?query=&pageSize=10&after=<id>",
  "nextPage": "api/search/<type>?query=&pageSize=10&before=<id>",
  "results": […]
]
```

Characters used to define query parameters such as brackets, quotes, or parentheticals will be parsed as query syntax. To search for items with those special characters directly in the name insert a backslash before the character to treat it as plain text. For example, if document title contains the word "blue" (with quotes around it) you would need to search for the following to match the keyword.

```
\"BLUE\"
```

The pageSize query parameter can be changed to include the number of results returned. By default, or if not specified it will be 10. The after and before parameters will automatically be calculated. To go to the next or previous page fetch the URLs provided. If the prevPage or nextPage properties are empty strings, it indicates that there are no results to display in that page.

CONTENT ITEM

GET /api/search/contents?query={query}

The search keyword should be specified using the "query" query parameter as shown above. The user who is invoking this API should have the "VIEW-SEARCH-CONTENTITEM" role permission.

When a user performs a search, only content items that the user has access to are returned. Content item search can support external content access checks as well, where an external web service will be invoked to check users' content access before the content items are returned in the search results.

If the API call is successful, it will return with a status code 200 with an array of content item objects that match the keyword in the "results" property of the response body. Below properties are included for each content item object.

Property	Property Description
name	content name
description	content description
archived	true if content is archived, otherwise false
creationdate	creation timestamp
lastmoddate	last modification timestamp
lastmodifiedby	user id of the user who last modified the content
properties	properties defined by custom schema

_id	content id

Example response:


```
{
  "prevPage": "/api/search/contents?query=&pageSize=10&after=645a82763280e4b67ca0e3ca",
  "nextPage": "api/search/contents?query=&pageSize=10&before=645a82763280e4b67ca0e3ca",
  "results": [{
    "name": "sample name",
    "description": "sample description",
    "archived": true,
    "creationdate": "2021-04-08T16:37:31.192Z",
    "lastmoddate": "2021-04-08T16:37:31.192Z",
    "lastmodifiedby": "jdoe",
    "properties": {
      "revision": "A",
      "version": 1.0
    },
    "_id": "606f314b0e3b949ecb58bde2"
  }]
]
```


Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authorization failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

ACTIVITY

GET /api/search/activities?query={query}

The search keyword should be specified using the "query" query parameter as shown above. The user who is invoking this API should have the "VIEW-SEARCH-ACTIVITY" role permission.

If the API call is successful, it will return with a status code 200 with an array of activity objects that match the keyword in the "results" property of the response body. Below properties are included for each activity object.

Property	Property Description
title	activity title
description	activity description
owners	array of user objects who are activity owners
owners.**name**	owner name
owners.**userid**	owner user login
owners.**_id**	owner user id

lists	array of activity list objects to be included in the activity
lists.**title**	list title
lists.**_id**	list id
users	array of user objects to be included in the activity
users.**name**	user name
users.**userid**	user login
users.**_id**	user id
contents	array of contents (ids) to be included in the activity
startdate	activity start date
enddate	activity end date
_id	activity id

Example response:

```
{
  "prevPage":
"/api/search/activities?query=&pageSize=10&after=645a82763280e4b67ca0e3ca",
  "nextPage":
"api/search/activities?query=&pageSize=10&before=645a82763280e4b67ca0e3ca",
  "results": [{
    "title": "sample title",
    "description": "sample description",
    "owners": [{
      "_id": "78666ac968841279e86c91df",
      "name": "john doe",
      "userid": "jdoe"
    }],
    "lists": [{
      "_id": "56432ac968841279e86261de",
      "title": "engineering",
    }],
    "users": [{
      "_id": "61e994daa70a384f84478b8f",
      "name": "Samwise Gamgee",
      "userid": "gardener"
    },{
      "_id": "6160818266450927daa1504f",
      "name": "Frodo Baggins",
      "userid": "ringbearer"
    }],
    "contents": ["67866ac968841279e86cdf91"],
    "startdate": "2021-04-08T16:37:31.192Z",
    "enddate": "2021-05-08T16:37:31.192Z",
    "_id": "606f314b0e3b949ecb58bde2"
  }, {
    …
  }]
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

WORK INSTRUCTION

GET /api/search/workinstructions?query={query}

The search keyword should be specified using the "query" query parameter as shown above. The user who is invoking this API should have the "VIEW-SEARCH-WORKINSTRUCTION" role permission.

When a user performs a search, all work instructions that matches the search keywords are returned. Work instruction search can support external access checks as well, where an external web service will be invoked to check users' work instruction access before the work instructions are returned in the search results.

If the API call is successful, it will return with a status code 200 with an array of work instruction objects that match the keyword in the "results" property of the response body. Below properties are included for each work instruction object.

Property	Property Description
title	work instruction title
description	work instruction description
owners	array of user ids
properties	properties defined by custom schema
_id	work instruction id

Example response:

```
{
  "prevPage":
"/api/search/workinstructions?query=&pageSize=10&after=645a82763280e4b67ca0e3ca",
  "nextPage":
"api/search/workinstructions?query=&pageSize=10&before=645a82763280e4b67ca0e3ca",
  "results": [{
    "title": "sample title",
    "description": "sample description",
    "owners": [“6193e8454ae163efb2259a02”, ...],
    "properties": {
      "revision": "A",
      "version": 1.0
    },
    "_id": "606f314b0e3b949eaf58bd12"
  }, {
    …
  }]
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authorization failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

USERS

GET /api/search/users?query={query}

The search keyword should be specified using the "query" query parameter as shown above. The user who is invoking this API should have the "VIEW-SEARCH-USER" role permission.

In addition to the general search policies listed above, an Admin user can also confine users to searching for users within their organization and selected additional organizations.

If the API call is successful, it will return with a status code 200 with an array of user objects that matches the keyword in the "results" property of the response body. Below properties are included for each user object.

Property	Property Description
userid	user login
email	user email address, used for notifications
sso	if true, user is an external user and authenticates through external methods
name	user's name
title	user's title
department	user's department
organization	array of user's organization(s)
roles	User roles (ids)
site	user's site
active	if false, the user cannot log into the system

Example response:


```
{
  "prevPage": "/api/search/users?query=&pageSize=10&after=645a82763280e4b67ca0e3ca",
  "nextPage": "api/search/users?query=&pageSize=10&before=645a82763280e4b67ca0e3ca",
  "results": [{
    "userid": "jdoe",
    "email": "john.doe@example.com",
    "sso": false,
    "name": "john doe",
    "title": "Senior Manager",
```

```
    "department": "Marketing",
    "organization": ["Anark"],
    "roles": ["606f314b0e3b949ecb58bdd4"],
    "site": "North America",
    "active": true,
    "_id": "606f314b0e3b949ecb58bde2"
  }, {
    …
  }]
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authorization failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

GROUPS

GET /api/search/groups?query={query}

The search keyword should be specified using the “query” query parameter as shown above. The user who is invoking this API should have the “VIEW-SEARCH-GROUP” role permission.

If the API call is successful, it will return with a status code 200 with an array of group objects that matches the keyword in the “results” property of the response body. Below properties are included for each user object.

Property	Property Description
name	Group name
description	Group description
roles	Group roles (ids)

Example response:

```
{
  "prevPage": "",
  "nextPage": "",
  "results": [
    {
      "name": "sample group",
      "description": "",
      "roles": [
        "65862c5d74516996234a4c68"
      ],
      "_id": "658ef2a1e59fa12e8c50bb0c"
    }, { … }
  ]
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authorization failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

GET TENANT PREFERENCES

Both GET tenant preference(s) APIs have the same authorization checks. The user who is invoking either API should have one of the following role permissions: “VIEW-CONTENTITEM”, “ADMINISTER-PREFERENCES”, “VIEW-SEARCH-CONTENTITEM”, “VIEW-SEARCH-USER”, “VIEW -SEARCH-ACTIVITYLIST”, “VIEW-SEARCH-ACTIVITY”, “VIEW-SEARCH-GROUP”

GET /api/tenantpreferences

Response will be key value pairs for each tenant preference in the database. For example:

```
{
"contentpropsschema": [
    {
      "displayname": "Revision",
      "propertyname": "revision",
      "propertytype": "integer",
      "isrequired": false
    }
  ],
  "contentpropstoshowinui": [
    {
      "metadatatoshow": [
        "description"
      ],
      "showkey": false
    }
  ],
  "searchitemsmaxcount": 0,
  …,
}
```

Response Status Codes:

200 Success

401 Unauthorized (authorization failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

GET /api/tenantpreferences/{prefKey}

Response will be a key value pair, with key set to “value” and the preference value will be the value. For example, if you call /api/tenantpreferences/searchitemsmaxcount, response will be {“value”: 1}.

Response Status Codes:

200 Success

401 Unauthorized (authorization failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

ACTIVITIES

CREATE AN ACTIVITY

POST /api/activities

Creates an activity with users and contents. The user who is invoking this API should have the "CREATE-UPDATE-DELETE-ACTIVITY" role permission.

Request:

Property	Property Description
title	activity title
description *Optional*	activity description
owners	users (ids) who will be owners of the activity
lists *Optional*	Groups (ids) that will participate in the activity
users *Optional*	Users (ids) that will participate in the activity
contents *Optional*	array of content items (ids)
startdate *Optional*	activity start date
enddate *Optional*	activity end date
state *Optional*	activity state one of ['Open', 'In Review', 'In Progress', 'Closed']

Example request:

```
{
  "title": "sample title",
  "description": "sample description",
  "owners": ["78666ac968841279e86c91df"],
  "lists": ["56432ac968841279e86261de"],
  "users": ["61e994daa70a384f84478b8f", "6160818266450927daa1504f"],
  "contents": ["67866ac968841279e86cdf91"],
  "startdate": "2021-04-08T16:37:31.192Z",
  "enddate": "2021-05-08T16:37:31.192Z",
  "state": "Open"
}
```

If the API call is successful, it will return a status code 200 with the response body:

Property	Property Description
activityId	Activity id
rootFolderId	The root folder where Activity uploaded files and subfolders are stored in.

Example response:

```
{
  "activityId": "606f314b0e3b949ecb58bde2",
  "rootFolderId": "606f314b0e3b949ecb69cef3"
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

UPDATE AN ACTIVITY

PUT /api/activities/{activityId}

Update an activity (users, groups, contents, start/end dates). “ActivityId” in the URL is the unique identifier of the activity which is being updated. The user who is invoking this API should have the “CREATE-UPDATE-DELETE-ACTIVITY” role permission or be the activity owner.

Request is the same as creating an activity.

Response Status Codes:

204 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (activity not found)

500 Internal Server Error

DELETE AN ACTIVITY

DELETE /api/activities/{activityId}

Deletes the activity from database. “ActivityId” in the URL is the unique identifier of the activity which is being updated. The user who is invoking the API should have the “CREATE-UPDATE-DELETE-ACTIVITY” role permission or be the activity owner.

If the API call is successful, it will return a status code 204.

Response Status Codes:

204 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (activity not found)

500 Internal Server Error

GET AN ACTIVITY

GET /api/activities/{activityId}

Retrieves the activity metadata properties. “ActivityId” in the URL is the unique identifier of the activity for which properties need to be retrieved. The user who invokes this API should have the “CREATE-UPDATE-DELETE-ACTIVITY” role permission or the “VIEW-ACTIVITY” role permission.

Response will have two properties, “activity” and “rootFolder”. The “activity” property will have the following information:

Property	Property Description
title	activity title
description	activity description
owners	array of user objects who are activity owners
owners.**name**	owner name
owners.**userid**	owner user login
owners.**email**	owner user email
owners.**title**	owner user title
owners.**department**	owner user department
owners.**organization**	owner user organization
owners.**sso**	is owner user an external user
owners.**active**	Is owner user an active user
owners.**preferences**	owner user’s preferences object
owners.**_id**	owner user id
lists	array of activity list objects included in the activity
lists.**title**	list title

lists.**_id**	list id
users	array of user objects included in the activity
users.**name**	user name
users.**userid**	user login
users.**email**	user email
users.**title**	user title
users.**department**	user department
users.**organization**	user organization
users.**sso**	is user an external user
users.**active**	Is user an active user
users.**preferences**	user's preferences object
users.**_id**	user id
contents	array of content item objects included in the activity - metadata is only returned for content items that user can access
contents.**name**	content name
contents.**archived**	if the content has been archived
contents.**description**	the content description
contents.**creationdate**	the creation date of that content
contents.**lastmoddate**	the last date the content was modified
contents.**lastmodifiedby**	the last user who modified the content
contents.**properties**	an object of key value pairs consistent with the custom properties schema defined for contents
contents.**_id**	content id
startdate	activity start date
enddate	activity end date
state	activity state
_id	activity id

Example response:


```
{
  "activity": {
    "_id": "612d6178637aa6009400e8ec",
    "contents": [
      {
        "_id": "612e5444637aa6009400e96f",
        "archived": false,
        "name": "ASM-RRC-001",
        "creationdate": "2021-08-31T16:09:40.769Z",
        "lastmoddate": "2021-08-31T17:11:24.996Z",
        "lastmodifiedby": "sampleuser",
        "properties": {
```


```
          "version": "B",
          "revision": "1.0"
        },
        "description": "A description about a one of a kind Roof Rack"
      },
      {
        "_id": "612e550d637aa6009400ec46",
        "archived": false,
        "name": "Ion Drive",
        "creationdate": "2021-08-31T16:13:01.795Z",
        "lastmoddate": "2021-12-01T17:26:25.754Z",
        "lastmodifiedby": "sampleuser",
        "properties": {
          "version": "A",
          "revision": "1.0"
        },
        "description": "MBE67400311-1"
      }
    ],
    "description": "Review of samples",
    "enddate": "2021-08-31T22:52:00.000Z",
    "lists": [],
    "owners": [
      {
        "_id": "612d613d637aa6009400e559",
        "sso": false,
        "organization": "your company here",
        "active": true,
        "userid": "sampleuser",
        "email": "su@anark.com",
        "name": "sample user",
        "title": "Mr.",
        "department": "samplization",
        "preferences": {
          "notifications": true
        }
      }
    ],
    "startdate": "2021-08-30T22:52:59.000Z",
    "state": "Open",
    "title": "sample activity",
    "users": [
      {
        "_id": "6079e0cb64810f0055ca2b9a",
        "sso": false,
        "active": false,
        "userid": "test user",
        "email": "testuser@anark.com",
        "name": "testuser",
        "title": "Mr.",
        "department": "Engineering",
        "organization": "anark",
        "preferences": {
          "notifications": true
        }
      }
    ]
  },
  "rootFolder": {
    "_id": "645d2b081c0bcb4cb690b8dd",
    ...
  }
}
```


Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

GENERATE AN ACTIVITY REPORT

POST /api/activities/{activityId}/report

Initiates generating an activity report. The user who is invoking this API should have the "VIEW-ACTIVITY" role permission.

Request:

Property	Property Description
conversationIds *Optional*	The ids of conversations to include in the report.

Example request:

```
{
  "conversationIds": ["61e994daa70a384f84478b8f", "6160818266450927daa1504f"]
}
```

If the API call is successful, it will return a status code 200 with the job id in the response body. Jobs service status API should be used to poll periodically to get the status of the job.

Property	Property Description
jobId	Job id (used for status updates)

Example response:

```
{
  "jobId": "xctyuiHJ"
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (activity not found)

500 Internal Server Error

USER MANAGEMENT

ADD A USER

POST /api/users

Creates a user in the system. The user who invokes this API should have the “CREATE-UPDATE-DELETE-USER” role permission.

Request:

Property	Property Description
userid	User login
email	User email address, used for notifications
password *Optional*	User password (required if sso is not specified)
sso *Optional*	If true, user is an external user and authenticates through external methods (required if password is not specified)
name	User’s name
title *Optional*	User’s title
department *Optional*	User’s department
organization *Optional*	Array of user’s organization(s)
roles	Roles (ids) the user needs for application permissions and access classifiers.
site *Optional*	User’s site
customprops *Optional*	An array of custom properties saved as key-value pairs.

Example request:

```
{
  "userid": "jdoe",
  "email": "john.doe@example.com",
  "sso": false,
  "name": "john doe",
  "title": "Senior Manager",
  "department": "Marketing",
  "organization": ["Anark"],
  "roles": ["612d613d637aa6009400e789", “6144d9ba38b7140058b0d7ac”],
  "site": "North America",
  "customprops": [{ key: "anarkid", value: "example.user" }],
}
```

If the API call is successful, it will return a status code 200 with the user id in the response body.

Property	Property Description
userId	user's unique identifier

Example response:

```
{
  "userId": "60566ac968841279e86c9130"
}
```

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

500 Internal Server Error

UPDATE A USER

PUT /api/users/{userId}

Updates a user in the system. "UserId" is the unique identifier of the user being updated. The user who invokes this API should have the "CREATE-UPDATE-DELETE-USER" role permission.

Request is the same as when creating a user.

Response Status Codes:

204 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (user not found)

500 Internal Server Error

DELETE A USER

DELETE /api/users/{userId}

Deletes a user in the system. “UserId” is the unique identifier of the user being updated. The user who invokes this API should have the “CREATE-UPDATE-DELETE-USER” role permission.

Response Status Codes:

204 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (user not found)

500 Internal Server Error

GET A USER

GET /api/users/{userId}

Retrieves the user information. “UserId” in the URL is the unique identifier of the user for which properties need to be retrieved. The user who invokes this API should have the “CREATE-UPDATE-DELETE-USER” role permission or the “VIEW-ACTIVITY” role permission.

Response:

Property	Property Description
userid	User login
email	User email address, used for notifications
sso	If true, user is an external user and authenticates through external methods
name	User’s name
title	User’s title
department	User’s department
organization	Array of user’s organization(s)
roles	User roles (ids)
site	User’s site
active	If false, the user cannot log into the system
lastlogin	The corresponding timestamp when the user last logged in

preferences	User's preferences object
customprops	An array of custom properties
profilePicture	User's profile picture if present or null

Example response:


```
{
  "userid": "jdoe",
  "email": "john.doe@example.com",
  "sso": false,
  "name": "john doe",
  "title": "Senior Manager",
  "department": "Marketing",
  "organization": ["Anark"],
  "roles": ["612d613d637aa6009400e789", "6144d9ba38b7140058b0d7ac"],
  "site": "North America",
  "active": true,
  "lastlogin": "2021-08-31T16:13:01.795Z",
  "preferences": {
    "notifications": true
  },
  "customprops": [{ key: "anarkid", value: "example.user" }],
  "profilePicture": {
    "cropped": {
      "data": "iVBORw0KGgoAAAANSUhEUgAAAgAAAAIACAMAAADDpi…",
    "contentType": "image/png",
    }
  },
  "_id": "606f314b0e3b949ecb58bde2"
}
```


Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

403 Forbidden (role permission check failed)

404 Not Found (user not found)

500 Internal Server Error

AUTHENTICATION

LOGIN

POST /api/auth/login

Logs a user into the system and creates an active session.

Request:

Property	Property Description
login	user's login (base64 encoded)
password	user's password (base64 encoded)

Example request:

```
{
  "login": "amRvZQ==",
  "password": "YmFKcGFzc3dvcmQ=",
}
```

A successful login will return a cookie and status code 200.

Response Status Codes:

200 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)

500 Internal Server Error

LOGOUT

POST /api/auth/logout

Logs a user out of the system and deletes the active session.

Request is an empty body with the cookie obtained at login, specified in the request header. For example:

```
headers: {
  "cookie": "mbewebsid=……"
}
```

Response Status Codes:

200 Success

401 Unauthorized (authentication failed)

500 Internal Server Error

GET AUTHENTICATED USER

GET /api/auth/user

Gets the logged in user information from the active session.

Response:

Property	Property Description
userid	user login
sso	if true, user is an external user and authenticates through external methods
name	user's name
roles	Array of roles assigned to the user
_id	user's unique identifier (returned as userId in POST /api/users)
canCreateActivity	true if the user has permission to create an activity
canCreateConv	true if the user has permission to create a conversation
canCreateComment	true if the user has permission to create a comment
canCreateContent	true if the user has permission to create, edit, and delete a content item
canCreateWI	true if the user has permission to create a work instruction
canCreateGroup	true if the user has permission to create a group
canManageGroup	true if the user has permission to edit or delete a group
canManageActivity	true if the user has permission to edit or delete an activity
canManageWI	true if the user has permission to edit or delete a work instruction
canCreateActivityList	true if the user has permission to create an activity group of users
canManageActivityList	true if the user has permission to edit or delete a work instruction
canManageUser	true if the user has permission to edit or delete users

canPublish	true if the user has permission to publish content items, upload files, and upload templates
canViewContent	true if the user has permission to view content (separate from content access control)
canViewFeed	true if the user has permission to view feed
canViewWI	true if the user has permission to view work instructions
canSearchContent	true if the user has permission to search content items
canSearchActivityList	true if the user has permission to search activity groups
canSearchUser	true if the user has permission to search users
canSearchActivity	true if the user has permission to search activities
canSearchGroup	true if the user has permission to search groups
canSearchWI	true if the user has permission to search work instructions
canExecuteWI	true if the user has permission to execute a work instruction
canQueryWIExecution	true if the user has permission to generate execution reports
canManagePreferences	true if the user has permission to edit tenant preferences
canPublishUserFile	true if the user has permission to upload to My Files
canPublishActivityFile	true if the user has permission to upload to an activity
sessionInfo	Object containing headers stored in guest user's session that are custom (not standard user metadata properties). Only returned if Admin configured "Guest User Configuration" in System Preferences.

Example response:


```
{
  "userid": "jdoe",
  "name": "john doe",
  "sso": false,
  "roles": ["activityauthor","contentauthor","viewer"],
  "canCreateActivity": true,
```

```
  "canCreateConv": true,
  "canCreateComment": true,
  "canManageActivity": true,
  "canCreateActivityList": true,
  "canManageUser": false,
  "canManageGroup": false,
  "canPublish": true,
  "canViewContent": true,
  "canViewFeed": true,
  "canSearchContent": true,
  "canSearchUser": true,
  "canSearchActivityList": true,
  "canSearchActivity": true,
  "canSearchGroup": false,
  "canManagePreferences": false,
  “canCreateContent”: true,
  “canCreateWI”: false,
  “canManageWI”: false,
  “canSearchWI”: false,
  "canExecuteWI”: false,
  “canQueryWIExecution”: false,
  “canPublishUserFile”: true,
  “canPublishActivityFile”: true,
  "_id": "5d83a7f9558e9f0029543514"
}
```


Response Status Codes:

200 Success

401 Unauthorized (authentication failed)

500 Internal Server Error

INTEGRATION API HOOKS

Anark Collaborate can support external authentication and access control checks for content items and search results. Also, custom REST API's can be added, for example, an API can be added to synchronize users from an external system. All these extensions are possible using the integration API hooks, stub implementation of these API's is in a file "integrationapi.js" in the "sdk" sub directory under Anark Collaborate installation folder. Node.js/Express and JavaScript knowledge is required to develop integrations.

INITIALIZE

The initialization hook **init** performs any additional initialization required to run the integration code. It gets two arguments:

- *app:* is the Anark Collaborate Express application instance
- *config:* an object that contains the env settings and tenant preferences

You can use the "app" object to mount any required Express middleware, for example, to use custom REST API. If the integration requires new env settings that needs to be configured, use the config object to add new env settings that were specified in the PM2.json file.

Ex.

```
init(app, config) {
  app.use('/api/supplier', supplierApi);
  config.showcontentlandingpage = process.env.SHOWCONTENTLANDINGPAGE;
}
```

AUTHENTICATE AN EXTERNAL USER

Out of the box, Anark Collaborate supports local authentication. It can be customized and configured to be integrated with the single sign-on solution that is deployed in the enterprise (SAML 2.0, OpenID Connect) or perform authentication against LDAP server or other systems if required. Anark Collaborate platform uses Passport.js framework for authentication. Passport strategies are available to support numerous authentication mechanisms, integration can use one of them to support the authentication provider used in the enterprise.

The **authenticateUser** hook is used to authenticate an external user, it will be invoked if "AUTHEXTERNALSTRATEGY" env setting is specified. Implement the authentication functionality inside this hook. It gets Express request object, response object and the next callback arguments. This method will be invoked as an Express middleware and should invoke *next()* or *next(error)* after authentication, to return the status.

Ex.

```
authenticateUser(req, res, next) {
  return ssoAuth(req, res, next);
}
```

In the above example, "ssoAuth" is implemented in the integration code that will use Passport module to authenticate the external user.

EXTERNAL USER LOG OUT

The **logOutUser** hook is used to log out the external user. Implement any required functionality that is needed when the user logs out inside this hook. It gets Express request and response object and the session object as arguments. Session object will contain all the variables stored in user's session.

Ex.

```
logOutUser(req, res, session) {
  return ssoLogout(req, res, session);
}
```

In the above example, "ssoLogout" is implemented in the integration code, for example, it can support an external web page to be displayed when the user logs out.

EXTERNAL CONTENT ITEM ACCESS CHECK

Content access in Anark Collaborate can be managed by an external system, for example, PLM. This is useful when content access needs to be enforced based on the access control rules specified in an external system. External content access check is enabled by a tenant preference setting, "Access Control Web Service URL" which can be specified in System Preferences by an admin. Setting takes a web service URL that will be invoked by Anark Collaborate to check whether the user has access to the content before any user can view or collaborate with a specific content. Based on the response of the external system web service, a user will be granted or denied content access.

The **checkContentAccess** hook should be used to implement the web service invocation for checking content access for a specific user. It gets content item identifier, user metadata object and session as input arguments. Content item identifier can be used to get additional content item metadata. User metadata object will contain the user properties. Session object will contain all the variables stored in user's session. Implementation should return a Promise, if there are no errors, resolved promise should have value true if the user has access, otherwise false.

Ex.

```
checkContentAccess (contentId, user, session) {
  // Invoke external web service for access check
}
```

HANDLE EXPIRED SESSION

Session timeout can be specified in Anark Collaborate, **handleSessionExpiry** hook will be invoked when the session expires. User session is passed as an input argument, session object can be retrieved as shown in the example below. Implement any required functionality that needs to happen on session expiry, and it should return a Promise.

Ex.

```
handleSessionExpiry(session) {
  const sessionObj = JSON.parse(session.session);
  // Implement any needed logic
}
```

EXTERNAL ACCESS CHECK FOR SEARCH

Content item search can support external access checks. When a user performs a search, only content items that they have access is shown in the search results by invoking the external web service. External access check is enabled by a tenant preference setting, “Search Content Access Web Service URL” which can be specified in System Preferences (Search tab) by an admin.

The **checkSearchItemsAccess** hook should be used to implement the web service invocation to check whether the user has access to search result content items. The first argument *itemType* will be “content”. The second argument *items* will be an array of content items with metadata and properties that matched the search keyword, for which access checks should be performed on. User metadata object will contain the user properties. Session object will contain all the variables stored in user’s session. Implementation should return a promise, if there are no errors, promise should be resolved with an object with key value pairs, where key is the content item identifier (_id property) for each of the content items passed into this hook and the value should be set to the access check result for that content item (true or false).

Ex.

```
checkSearchItemsAccess(itemType, items, user, session) {
  // Invoke external web service for access check
}
```


CONTENT LINKING AND EMBEDDING

Anark Collaborate supports URL linking which allows external applications to load content item URLs with optional parameters targeting specified components in the content item. This is useful for indexing specific content item components with custom commands, for example to highlight a 3D part in an assembly that matches a part number or showing a particular document component and scrolling to a designated page number. Customers using custom code widgets in their template can specify what commands are available and what those command do when executed. More information can be found in the Template API documentation. Anark Collaborate also supports embedding content from external systems. By embedding the View app in an iframe, customers can issue custom commands through JavaScript's Post Message. Note: content linking and content embedding are only supported when viewing content through the "view" app as such:

https://<your.company.com>/view/<the_content_id>/

LINKING SYNTAX

To issue a command through URL linking, the URL must be formatted as above with additional trailing query parameters. Three parameters are supported,

- *Command* - the action to be executed.
- *Message* - details and supporting information for the action to execute correctly.
    - Users can specify multiple messages as key-value pairs. Keys and values are defined as <key>~<value>.
    - Multiple pairs can be separated with a comma.
- *componentId* - (Optional) the component identifier to load.

For example, if your template supports the command selectBalloon, then the URL

https://<your.company.com>/view/1e165dfde2/?command=selectBalloon&message=text~3a,view~front&componentId=8742

will tell Anark Collaborate to load content with id "1e165dfde2", load component with id "8742", and select the inspection balloon in the view "front" with text "3a".

EMBEDDED SYNTAX

To issue a command for an embedded content, the following syntax must be followed.

- *Command* - the action to be executed.
- *Message* - details and supporting information for the action to execute correctly. Multiple messages are formatted as an object.
- *componentId* - (Optional) the component identifier to load

For example, if your template supports the command selectBalloon, from an external system the following JavaScript Post Message will load content with id "1e165dfde2", load component with id "8742", and select the inspection balloon in the view "front" with text "3a".

```
window.postMessage({
    command: "selectBalloon",
    message: {
      text: "3a",
```

```
      view: "front",
    },
    componentId: "8742"
}, "*");
```


PUBLISHING A DATA COMPONENT

Anark Collaborate supports a “data” component in a content item, which allows users to publish data (Ex. PLM object metadata, BOM, change notice etc.) in JSON format. JSON data should adhere to the published schema that supports attribute (key/value pairs), attribute group (object with key/value pairs), URL object (link and display name property) and a table object (columns and rows of data). An attribute group or a table column supports a single URL or an array of URLs.

Anark JSON schema for data publishing is defined using JSON Schema specification (https://json-schema.org/), which is a widely adopted format for describing JSON data. Please refer to the provided schema definition file and sample data when building your own data components.

In its most basic form, JSON data should consist of attributes (key-value pairs). The following is an example of a simple document with all the primitive data types supported.

```
{
    "str": "foo",
    "int": 1234,
    "float": 35.4,
    "bool": true,
    "nil": null
}
```

Data documents may also include attribute group (nested object) with key-value pairs.

```
{
    "str": "foo",
    "nested": {
        "str": "foo",
        "int": 1234,
        "float": 35.4,
        "bool": true,
        "nil": null
    }
}
```

Arrays and deeply nested objects are currently not supported. The following examples are invalid.

```
{
    "arrays": ["are", "not", "supported"],
    "deeply": {
        "nested": {
            "objects": {
                "are": {
                    "not": "supported"
                }
            }
        }
```


```
    }
}
```

STRUCTURED DATA

Data components allow for a growing list of formal data structures for describing different types of information. These data formats have a special __TYPE__ property which is used to identify both the type and structure of the data.

URL

The URL data structure allows human readable text to be associated with a URL (href). This is useful for creating clickable text and hyperlinks.

```
{
    "sampleUrl": {
        "__TYPE__": "url",
        "displayName": "Anark Corporation",
        "href": "https://www.anark.com"
    }
}
```

Table

Tables allow for columns and rows of data. Column definitions are used to describe the number of columns and the type of data within each column. Each row should contain the data corresponding to the column definitions.

```
{
    "sampleTable": {
        "__TYPE__": "table",
        "columns": [
            {"name": "Name", "type": "string"},
            {"name": "Age", "type": "number"},
            {"name": "Has pets", "type": "boolean"}
        ],
        "rows": [
            ["John Smith", 32, false],
            ["Jane Doe", 27, true]
        ]
    }
}
```


Tables can also contain structured URLs.

```
{
    "sampleTable": {
        "__TYPE__": "table",
        "columns": [
            {"name": "Website", "type": "url"}
        ],
        "rows": [
            [{
                "__TYPE__": "url",
                "displayName": "Anark Corporation",
                "href": "https://www.anark.com"
            }]
        ]
    }
}
```

A single column can also contain an array of URLs.

```
{
    "sampleTable": {
        "__TYPE__": "table",
        "columns": [
            {"name": "External Links", "type": "array"}
        ],
        "rows": [
            [
                [
                    {"__TYPE__": "url", ...},
                    {"__TYPE__": "url", ...},      <-- row 1
                    {"__TYPE__": "url", ...},
                ]
            ],
            [
                [
                    {"__TYPE__": "url", ...},
                    {"__TYPE__": "url", ...},      <-- row 2
                ]
            ]
        ]
    }
}
```


SCHEMA VALIDATION

Please refer to https://json-schema.org/implementations.html#validators for information on the available validators in different languages to validate the JSON payload against a JSON schema definition. There are also online tools available, these tools make it easy to verify that your data is formatted correctly, we recommend using https://www.jsonschemavalidator.net.

MISCELLANEOUS

Other APIs are as follows.

LOG A MESSAGE

POST /api/log

Add a message to the message logs.

Request:

Property	Property Description
level	Specifies the message severity level. Allowed values are info, warn, and error.
message	Specifies the message to add to logs.

Example request:

```
{
  "level": "info",
  "message": "message",
}
```

Response Status Codes:

204 Success

400 Bad Request (validation failed)

401 Unauthorized (authentication failed)