GraphQL API Reference

Change history

Date	version	Comment
22/10/2025	0.10 alpha	First version, basic functionality, Queries.
24/11/2025	0.20 beta	Project-related mutations implemented. Metadata-related queries and mutations implemented. Project tags added to queries and mutations. User and Client-related queries and mutations implemented. Webhooks-related queries and mutations implemented.
20/12/2025	0.30 beta	Workflow management implemented. Asset and reference document download urls provided in Asset and RefDocument.
17/05/2026	0.40	Added mutations: **taskCreateUploadAssetChange()**, **workflowRun()**, and query **loggedInUser()**.

Please find the most recent **API Reference** online here.

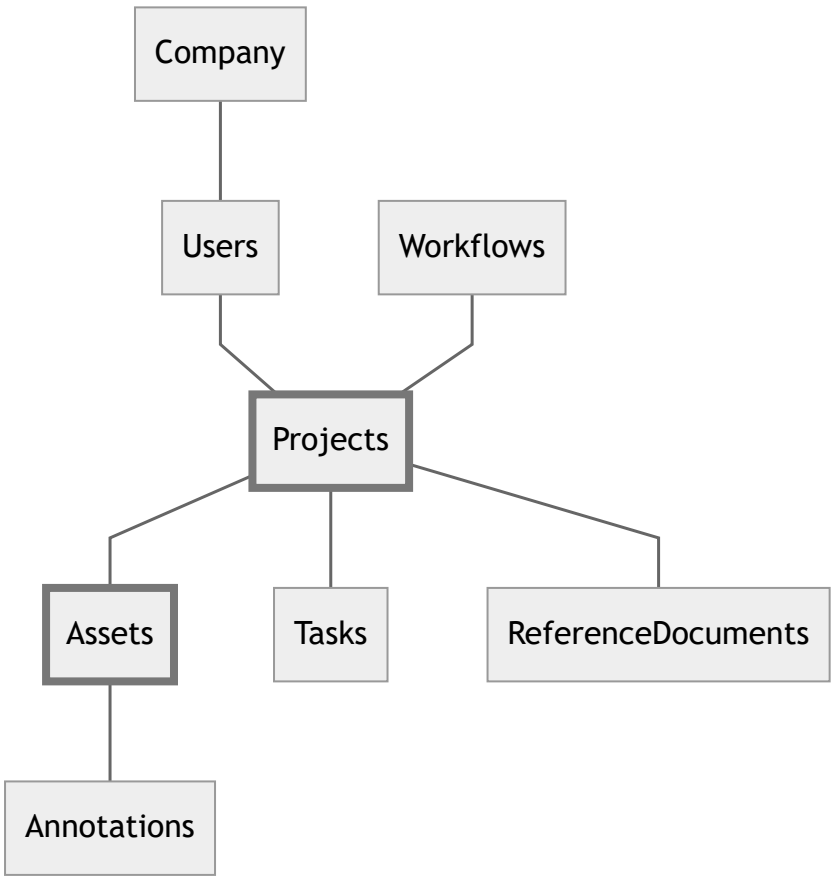
Table of contents

General

This is a GraphQL API to the Approval Studio, design review, and packaging approval SAAS. API utilizes its flow, authorization, storage, and so on.

Approval Studio basic concepts

From the business point of view, Approval Studio is based on a multitenant concept when a single user may be a part of one or more tenants (called Companies). Please see https://approvalstudio.freshdesk.com/ for detailed instructions on how to work with Approval Studio.



Authorization/authentication

The API authorization is based on an **authorization token** which should be requested prior to using any available API queries or mutations.
The flow is the following: request auth token by calling **authorizationToken()** query (see) providing Approval Studio's username/password. If ok, the method returns an authorization token that you should **provide with any other API call** as an **HTTP header** like this: `Authorization: Bearer XXXXXXXXXX`.

The token is valid for a limited amount of time, **120 minutes** by default. When the token is expired and still used you will have the error like this:

```
{
  "errors": [
    {
      "message": "Access denied for field 'projectById' on type 'QueryRoot'.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "extensions": {
        "code": "ACCESS_DENIED",
        "codes": [
          "ACCESS_DENIED"
        ]
      }
    }
  ]
}
```

Token management

A single query that provides you with the authorization token is `authorizationToken()`:

```
query {
  authorizationToken(
    name: "username@mail.com"
    password: "password$1"
    keepAliveTime: 123)
  {
```

```
        token,
        expirationDate
      }
    }
```

Sample result is :

```
{
  "data": {
    "authorizationToken": {
      "token": "eyJhbGciOiJIUzxxxxx",
      "expirationDate": "2025-10-21T20:23:05.1539332Z"
    }
  }
}
```

Now you can use this token to authorize access to the rest of the API.

Types

Project

A project, the primary system's entity, that aggregates properties, assets, reference documents, tasks etc.

```
{
  uID,
  name,
  state,
  customer,
  projectName,
  design,
  revision,
  createdAt,
  owners {
    ...
  },
  assets {
    ...
```

```
  },
  refDocs {
    ...
  },
  tasks {
    ...
  }
}
```

Fields

`uID: String!`
The unique id of the project, a GUID-like string, unique throughout the system.

`name: String`
The name of the project, a mandatory field.

`state: ProjectState`
Project state, see ProjectState.

`customer: String`
Customer name.
This is one of the descriptive project's attributes, just plain text, that could be use to categorize a project by, in this case, by a customer.

`projectName: String`
Project name.
Like a customer above, this is an optional descriptive project's attribute that point to a customer name or some kind of a customer's code.

`design: String`
Design name. Optional descriptive project's attribute.

`revision: String`
Revision number. Optional descriptive project's attribute.

`createdAt: DateTime!`
UTC date and time when the project was created.

`tags: [String]`
Optional tags associated with the project, array of plain string.

`metaData: [StringStringKeyValuePair]`
Optional metadata as a list of key-value pairs.

`reviewStatus: ProjectReviewStatus`
Review status based on the tasks; statistics. It's how many tasks are pending, how many reviewing assets/tasks approved or rejected.

`owners: [User]`
List of one or more project owners who are allowed to manage the project, create tasks, add or remove assets etc.
See User.

`assets: [Asset]`
List of assets associated with the project.
Assets are files, images or documents, that are subject of reviewing process.
The important part of asset management is version, an auto-incremental integer value, associated with every asset.
So if you need to have only the most recent version(s) you need to group assets by name and select the asset with

See Asset.

`refDocs: [RefDocument]`
List of reference documents associated with the project.
Reference documents are files of any type of URLs that contains some reference data related to the projects like detailed projects descriptions, standards, requirements etc.

See RefDocument.

`tasks: [Task]`
List of active tasks associated with the project.
See Task.

Asset

Asset is a file uploaded by a user interactively, using API or any available integration and is a subject of reviewing.
Reviewing assets is a primary purpose of the whole system.
Note: Images, documents and video assets are the same from point of view of the API.

Asset file types:

- `.pdf`, `.ai`, `.tiff`, `.jpeg`, `.gif`, `.bmp`, `.png`
- `.docx`, `.xls`, `.xlsx`, `.pptx`, `.ppt`, `.ppsx`, `.pps`, `.pot`, `.potx`, `.dotx`, `.dot`, `.odt`, `.ods`, `.rtf`, `.txt`
- `.htm` / `.html`
- `.mp4`

```
{
   uID,
   name,
   version,
   status,
   fileSize,
   createdAt
}
```

Fields

`uID: String!`
The unique id of the asset, a GUID-like string, unique throughout the system.

`name: String!`
The name of the asset, full file name like `"flyer_merchant.pdf"`.
The asset name used to group assets by versions, so when you upload an asset with the same name, it obtains a sequential version number, so the most recent asset with this name has has biggest version number.

`version: Int!`
Asset version, a consequent number.
Version is a consequential integer, assigning at asset uploading. The version is incremented for every unique asset name, for example:

> - file1.pdf - version 1
> - file1.pdf - version 2
> - file1.pdf - version 3
> - file2.pdf - version 1
> - file2.pdf - version 2

If, for instance, upload a file file2.pdf once more time, it will have version 3.

`status: AssetStatus`
Asset processing status.
Asset uploading is, generally, time-consuming process, that, possible, may end with fail.
So this status indicates that the asset is processing now, a processing completed or failed.

See AssetStatus

`reviewStatus: ReviewStatus`
Review status based on the review tasks; statistics of how many reviewers accept, reject asset or still reviewing it.

`fileSize: Int`
Size of the asset file, bytes.

`pagesCount: Int`
Number of pages in the asset file.

`createdAt: DateTime!`
UTC date and time when the asset was created.

`reviewUrl: String!`
A URL to a review tool that allows to interactive read-only review the asset.

`downloadUrl: String!`
URL to download asset's original file, as is.

`thumbnailUrl: String!`
URL to the asset's thumbnail image, first page in case of multi-page asset like pdf.

`imageUrls: [String]!`
URL to the asset's rasterized images, first page in case of multi-page asset like pdf. This is an array of URLs for every page in the asset.

Note: When **using the API from MCP,** you may need to download the asset original file or rasterized depending on your ability to read and decode original file: original PDF may have complex structure and required much effort to render; or you are unable to read the native asset format due some restrictions.

For example, you can create a kind of a **review report based on asset and requirements**. In this case you need to **choose an asset** and an appropriate **requirement document** that, usually, is uploaded as a reference document (see RefDocument).

Having a pair asset <-> requirement you need to make a decision which asset representation is better for you – original asset (field `downloadUrl`) or rasterized (field `imageUrls1`).

You may also you asset URLs for building UIs, sending mails, making snapshoots etc.

User

User is a primary unit of authorization in the system, and every entity somehow connected to some user (or multiple users).
Project, asset, reference document, task etc are created or processed on behalf of a user.

`uID: String!`
The unique id of the user.

`name: String!`
The name of the user, usually in form or “First name Last name” or only last name if the first name is not provided at user registration.

`email: String`
The user’s email, mandatory.

`clients: [Client]`
List if clients (tenants) the users belongs to.

Task

Task is created to assign a user with activity like reviewing assets or uploading or smth. else.

```
{
  uID,
  comment,
  status,
  dueDate,
  closedDate,
  createdAt,
  taskType,
  assignedTo {
    uID
    email,
```

```
    name
  }
}
```

Fields

`uID: String!`
The unique id of the task.

`comment: String`
An optional comment that a project owner may provide with the task to provide some additional information.

`status: TaskStatus`
Task activity status, see []

`dueDate: DateTime`
An optional due date that the task's creator associates with the task when creates it.

`closedDate: DateTime`
A UTC date and time of when task is closed.

`createdAt: DateTime!`
A UTC date and time when a test was created.

`taskType: TaskType`
A task type, that describes what the nature of the task: upload assets, documents, review assets etc.

`assignedTo: User`
A user this task is assigned to, see User.

RefDocument

Reference document associated with a project.
Reference documents are files of any type of URLs that project owner(s) wants to attach to a project and, optionally, share with team working on the project. Those can be requirements, standards, samples etc.
Reference documents are simpler than assets, no statuses, no processing etc.

```
{
  uID,
  name,
  size,
  createdAt
}
```

Fields

`uID: String!`
The unique id of the reference document.

`name: String!`
The name of the document or URL. When it is a file name, then the reference document is a downloadable file; when it's a URL, then it's available externally and Approval Studio

`size: Int!`
Size of the file.

`createdAt: DateTime!`
UTC date/time when the document was uploaded and assigned to a project.

`downloadUrl: String!`
URL to download reference document file, as is.

ProjectState

Project activity state.
This is an enumeration that represent possible project states.

State	
ACTIVE	Project is active, owner(s) can manage it, create tasks, upload assets etc.
ON_HOLD	This just name for projects that should not be active, but still fully functional.
COMPLETED	Project is completed, not active tasks allowed.
ARCHIVED	Project moved to archive, no any activity allowed.

State	
IN_TRANSIT	A short-living state when a project changes it’s state.

AssetStatus

Status of the asset processing.

Status	
PENDING	Asset is processing now.
PROCESSED	Asset has successfully uploaded and processed and is available to use.
FAILED	Asset processing failed due any reason. Asset not available and not visible in the list of assets.
	Note: Version number would be still increasing with every failed asset processing attempt.

TaskType

A task type, enumeration, that describes the nature of the task: upload assets, documents, review assets, assign users etc.

Type	
UPLOAD_ASSETS	Asset uploading task, assigned to a local user.
UPLOAD_REF_DOCS	Reference document uploading task, assigned to a local user.
REVIEW_ASSETS	Asset or multiple assets review task, assigned to a local user.
EXTERNAL_REVIEW_ASSETS	Asset or multiple assets review task, assigned to an external user (email).

Type	
UPLOAD_CHANGED_ASSET	New version of an asset uploading task, assigned to a local user.
UPLOAD_VIDEO	Video asset (.mp4) uploading task, assigned to a local user.
EXTERNAL_REVIEW_VIDEO	Video asset (.mp4) uploading task, assigned to an external user (email).
REVIEW_VIDEO	Video asset or multiple assets review task, assigned to a local user.
EXTERNAL_UPLOAD_ASSET	Asset uploading task, assigned to an external user (email).
EXTERNAL_UPLOAD_REFDOC	Reference document(s) uploading task, assigned to an external user (email).
ASSIGN_USER_ROLES	Workflow-related task, that require a local user to assign user roles for the given workflow.

TaskStatus

Task activity status, enumeration.

Status	
PENDING	Task is in active state, waiting to be either done or deleted.
CLOSED	Task is done.
APPROVED	Review asset task, internal or external, is done and reviewer approved the asset(s).
REJECTED	Review asset task, internal or external, is done and reviewer rejected the asset(s).

Status	
APPROVED_WITH_CHANGES	Review asset task, internal or external, is done and reviewer approved the asset(s) but left a comment what should be changed/done.

Client

Client or so called tenant. Approval Studio allows users to be internal users of multiple clients. **Note**: in most cases users belong to a single tenant. The current version of the GraphQL API is made based on assumption that users belong to a single tenant only.

`uID: String!`

The unique id of the asset.

`name: String!`

The name of the asset, file name.

`isPrimary: Boolean!`

A flag indicating this is the default client, used by default in mutations.

`created: DateTime!`

UTC-based date/time of the client creation.

Webhook

Please read detailed webhooks processing tutorial here – Approval Studio API guide

Fields

`uID: String!`

Webhook's UID.

`uRL: String!`

The webhooks URL.

`secret: String`

A unique string associated with the webhook.

The API host provides a **Secret** with every event object posted on an endpoint. The endpoint needs to check against this ID for every incoming post to avoid spamming. Keep secret safe.

`eventType: WebhookEventType`
A type of events this webhook is assigned to to.

`createdAt: DateTime!`
A UTC based webhook created date and time.

WebhookEventType

Webhook's type of event to handle.
Every time you register a new webhook you need to provide which event (or multiple events) will be a trigger.

Event code	Description
`PROJECT_CREATED`	Fires when a new project is created.
`PROJECT_EDITED`	Fires when a project's attribute changes, like name, description, due date, etc.
`PROJECT_STATE`	Fires when project state changes.
`ASSET_UPLOADED`	Fires when an asset or a new version of an existing asset is uploaded.
`ASSET_DELETED`	Fires when asset deleted.
`REF_DOC_UPLOADED`	A new reference document uploaded.
`REF_DOC_DELETED`	An existing reference document is deleted.
`ANNOTATION_ADDED`	An annotation to asset is created.
`ANNOTATION_EDITED`	An annotation is edited.
`ANNOTATION_DELETED`	An annotation is deleted.
`TASK_CREATED`	A new task of any type is created.
`TASK_COMPLETED`	A task is marked as completed.
`TASK_DELETED`	A task is deleted.

Event code	Description
TASK_APPROVED	An asset review task marked as approved.
TASK_REJECTED	An asset review task marked as rejected.
WEBHOOK_TEST	Dummy event object for the testing endpoint
ALL	Any event above will trigger the webhook invocation.

TimeLine

TimeLine represents a single entry of an activity log associated with projects.

`uID: String!`
The id of the timeline entry.

`description: String`
Textual representation of the event.

`eventType: TimeLineEvent!`
The event type.

`createdAt: DateTime!`
UTC-based event creating date and time.

`user: User`
The user instance that initiated this event, see User.

`project: Project`
The project instance this event associated with, see Project.

task: Task
The task this event associated with, see Task.

TimeLineAction

The list of events the TimeLine entry is logged for.

Value	Description
CREATE_USER	Create a new user.
USER_LOGIN	User logged in.
USER_LOGOUT	User logged out.
CREATE_PROJECT	Create a project.
UPDATE_PROJECT	Update project.
CHANGE_PROJECT_STATUS	Change project stat.
ADD_PROJECT_OWNER	Add project owner.
DELETE_PROJECT_OWNER	Delete project owner.
ADD_TASK	Create a new task.
DELETE_TASK	Delete a task.
UPLOAD_ASSET	Upload an asset.
UPLOAD_DOCUMENT	Upload a reference document.
DELETE_DOCUMENT	Delete a reference document.
ADD_ANNOTATION	Add an annotation.
ADD_COMMENT	Add a comment.
APPROVE_ASSET	Approve asset(s).
REJECT_ASSET	Reject asset(s).
EDIT_ANNOTATION	Edit an annotation.
SEND_ASSET_BY_EMAIL	Send an asset(s) by email.
CHANGE_USER_EMAIL	Change user’s email.
ADMIN_LOG	Admin log entry.
DELETE_ASSET_VERSION	User changed password.

Value	Description
USER_CHANGED_PASSWORD	Created new company.
CREATE_COMPANY	Delete asset version.
UPDATE_COMPANY_PROPERTY	Update company property.
ADMIN_USER_ADDED	Admin user added.
ADMIN_USER_REMOVED	Admin user removed.
CREATE_CLIENT_USER	Create client user.
USER_ACTIVATED	User activated.
UPDATE_USER_PROPERTY	Update user property.
USER_INVITATION_CANCELED	User invitation canceled.
NEW_ASSET_VERSION_UPLOADED	New version of asset was uploaded.
SEND_REFERENCE_DOCUMENT_BY_EMAIL	Send a reference document by email.
TASK_CANCELED_BY_NEW_VERSION	Task canceled by new version of asset upload.
DELETE_VIDEO	Delete video version.
NOTIFICATIONS_ON_PENDINGS_SENT	Notifications on pending tasks and projects sent.
UPLOAD_COMMENT_ATTACHMENT	Upload Comment Attachment
GTM_ACTION	GTM Action.
SET_ANNOTATION_COMPLETE	Set annotation complete.
SET_ANNOTATION_INCOMPLETE	Set annotation incomplete.
CLOSE_TASK	Close a task.
CLOSE_TASK_BY_USER_DEACTIVATION	Close a task due to user deactivation.
CHANG_PROJECT_OWNER_BY_USER_DEACTIVATION	Change project owner due to a user's deactivation.

Value	Description
DOWNLOAD_ASSET	Download an asset.
DOWNLOAD_REF_DOC	Download a reference document.
APPROVE_WITH_CHANGES	Approve with changes.
SEND_MESSAGE	Send a message.
DELETE_PROJECT	Delete a project.
EXTERNAL_UPLOAD_ASSET	External upload asset.
EXTERNAL_UPLOAD_REF_DOC	External upload reference document.
APP_SUMO_ACTION	AppSumo's action.
UPDATE_COMMENT_REACTION	Update Comment Reaction.
CLAIM_TASK	Claim task.
REASSIGN_TASKS_PROJECTS	Reassign tasks and projects.
COPY_ASSET_TO_DAL	Copy Asset to DAL.
COPY_DAL_ASSET_TO_PROJECT	Copy DAL asset to project.
MANUAL_WORKFLOW_TRIGGER	Manual workflow trigger.
FILE_LINK_OPENED	File link was opened.

Workflow

Represents a complete workflow configuration with triggers, actions, and roles.

Fields

`uID: String!`

A unique action identifier, string.

`name: String`
The optional name of the workflow.

`created: DateTime!`
UTC-based workflow creation date/time

`isActive: Boolean!`
Flag, that indicates if this an active workflow or not.

`actions: [WorkflowAction]`
List of workflow actions.

`roles: [WorkflowRole]`
Optional list of roles used in the workflow.

WorkflowAction

Action is what is the reaction to a trigger. Actions contain different number of fields to provide, depends on the nature of the action.
Note: **WorkflowActionInput** has the same fields but used as an input parameter.

Fields

`uID: String`
The id of the workflow's action.

`friendlyName: String`
The optional name of the workflow.

`triggerType: WorkflowTriggerType!`
The mandatory trigger type. It define which event will trigger the action.

`actionType: WorkflowReactionType!`
The mandatory action type that defines what the action represents as a reaction to a trigger and fields must be provided to make the workflow action workable.

`userUIDs: [String]`
List of user UIDs who will have the action task assigned to.

`roleUIDs: [String]`
List of user group UIDs who will have the action task assigned to.

`triggerExecutorUID: String`
UID of the user that triggers the event.

`triggerRoleUID: String`
UID of the user group that triggers the event.

`emailSubject: String`
The optional email subject whe the reaction .

`emailLanguage: String`
Language code of the email.

`password: String`
The external task's optional password, that an external user would need to enter to get access to the task.

`comment: String`
Optional comment that would be assigned to a task that is created as a reaction to a trigger.

`checklistName: String`
An optional checklist name that may be used in asset review tasks.

`checklistItems: [String]`
An optional list of items of a checklist name that may be used in asset review tasks.

`daysToDueDate: Int`
Optional amount of day tha the task, created as a reaction to the trigger, will have to be complete.

`allowDownload: Boolean`
In case of the review task, this optional flag indicates if downloading assets will be possible through UI.

`isSimpleMode: Boolean`
In case of the review task, this optional flag indicates if simplified UI be provided instead of the full professional interface.

`isCustomPayload: Boolean`
When the reaction is posting data to an external webhook, this flag indicates that

WebhookCustomFields field will be using to posting the data.

`webhookURL: String`
When the reaction is posting data to an external webhook, this field contains the webhook URL.

`webhookRequestType: String`
The HTTP method, POST, GET etc.

`webhookContentType: String`
The webhook post's content data, "application/json" by default.

`webhookFields: [String]`
The list of the fields to post to the webhook.

`webhookHeaders: [StringStringKeyValuePair]`
The optional list of the webhook post's headers, empty by default.

`webhookCustomPayload: String`
When the reaction is posting data to an external webhook and the option IsCustomPayload is set, this field contains the data to post, as is.

`contacts: String`
When the reaction is an external task, this filed contain a list of the task's recipients, emails.

`targetKanbanColumnUID: String`
When the task is moving project to a predefined Canban column, this filed is an UID of the column.

WorkflowTriggerType

Enumeration of events that can trigger workflow actions.

Trigger Type	Description
`PROJECT_CREATED`	Fires when a new project is created.
`INTERNAL_UPLOAD_ASSETS_TASK_COMPLETED`	Fires when internal user completes asset upload task.

Trigger Type	Description
`INTERNAL_UPLOAD_REF_DOCS_TASK_COMPLETED`	Fires when internal user completes reference doc upload task.
`INTERNAL_REVIEW_TASK_REJECTED`	Fires when internal reviewer rejects assets.
`INTERNAL_REVIEW_TASK_APPROVED`	Fires when internal reviewer approves assets.
`EXTERNAL_UPLOAD_ASSETS_TASK_COMPLETED`	Fires when external user completes asset upload task.
`EXTERNAL_UPLOAD_REF_DOCS_TASK_COMPLETED`	Fires when external user completes reference doc upload task.
`EXTERNAL_REVIEW_TASK_REJECTED`	Fires when external reviewer rejects assets.
`EXTERNAL_REVIEW_TASK_APPROVED`	Fires when external reviewer approves assets.
`USER_ROLES_ASSIGNED`	Fires when user roles are assigned to workflow.
`MANUAL_TRIGGER`	Fires when workflow is manually triggered.

WorkflowReactionType

Enumeration of actions that can be performed in response to triggers.

Reaction Type	Description
`CREATE_INTERNAL_UPLOAD_ASSETS_TASK`	Create task for internal user to upload assets.
`CREATE_INTERNAL_UPLOAD_REF_DOCS_TASK`	Create task for internal user to upload reference docs.

Reaction Type	Description
`CREATE_EXTERNAL_UPLOAD_ASSETS_TASK`	Create task for external user to upload assets.
`CREATE_EXTERNAL_UPLOAD_REF_DOCS_TASK`	Create task for external user to upload reference docs.
`CREATE_INTERNAL_REVIEW_TASK`	Create review task for internal user.
`CREATE_EXTERNAL_REVIEW_TASK`	Create review task for external user.
`SEND_ASSETS_BY_EMAIL`	Send assets via email.
`SEND_REF_DOCS_BY_EMAIL`	Send reference documents via email.
`COMPLETE_PROJECT`	Mark project as complete.
`SEND_DATA_WITH_WEBHOOK`	Post data to external webhook.
`ASSIGN_USER_ROLES`	Request role assignments for workflow.
`SEND_CUSTOM_EMAIL`	Send custom email notification.
`MOVE_IN_KANBAN`	Move project to specific Kanban column.
`SWITCH_TO_WORKFLOW`	Switch to different workflow.

WorkflowRole

The concept of roles: instead of hardcoding users in the actions, you may define roles like ‘Designer’, ‘Reviewer’, ‘External Reviewer’ etc and ask someone provide the concrete persons before workflow starts.

Note: **WorkflowRoleInput** is the same but used as an input parameter.

Fields

`uID: String!`

The id of the role name, a unique string like GUID, recommended to be 4-chars string.

`name: String`
Name of the role, like ‘Uploader’, ‘External Uploader’, ‘Reviewer’, ‘External Revuewer’.

`isInternal: Boolean`
An flag, indicating is this role related to a company’s internal user, that has his/her own account in the system, or and external one, available through email.

Queries

Query	Type
`projectById()`	Project
`assetById()`	Asset
`userById()`	User
`taskById()`	Task
`refDocumentById()`	RefDocument

projectById

`projectById(uid: ID!): Project`
Returns a project by a unique project id with related entities: project owners, assets, reference documents, tasks etc.
Returns Project or error if no project found.

Request:

```
query projectById {
  projectById(uid:"XXXXXXXXXXXXXXX")
  {
    uID,
    name,
    state,
    customer,
```

```
    projectName,
    design,
    revision,
    createdAt,
    owners {
      uID,
      name,
      email
    },
    assets {
      uID,
      name,
      version,
      status,
      fileSize,
      createdAt
    },
    refDocs {
      uID,
      name,
      size,
      createdAt
    },
    tasks {
      uID,
      comment,
      status,
      dueDate,
      closedDate,
      createdAt,
      taskType,
      assignedTo {
        uID
        email,
        name
      }
    }
  }
}
```

projects

```
projects(
        query: String
        uIDs: [String]
        dateFrom: DateTime
        dateTo: DateTime
        states: [ProjectState]
        metaData: [StringStringKeyValuePairInput]
        limit: Int
): [Project]
```

Arguments

`query: String`

Plain text substring to look for in the projects' attributes, optional.

`uIDs: [String]`

List of projects identifier to get. This parameter is ignored when null or empty.

`dateFrom: DateTime`

Beginning of the interval of the projects' creation UTC date/time to look for, optional, inclusive.

`dateTo: DateTime`

End of the interval of the projects' creation UTC date/time to look for, optional, exclusive.

`states: [ProjectState]`

Optional list of the project states to look for. `ACTIVE` is the default one.

`metaData: [StringStringKeyValuePairInput]`

List of project's metadata. When provided, the search result wil include only projects that contain the provided metadata.

`limit: Int`

Optional maximum number of records to return, default value is 100, maximum 1000.

assetById

Returns an asset by the given unique identifier with all the properties and associated objects. Returns Asset or error if no project found.

Request:

```
query assetById {
  assetById(uid: "XXXXXXXXXXXXXXX") {
    uID
    name
    version
    status
    fileSize
    createdAt
  }
}
```

userById

Returns an user by the given unique identifier.
Returns User or error if no user found.

```
query userById {
  userById(uid: "XXXXXXXXXXXXXXX") {
    uID
    name
    email
  }
}
```

users

Returns all active users in the system.

```
query Users {
  users {
    uID,
    email,
    name,
    clients {
      uID,
      name,
      isPrimary
    }
```

```
  }
}
```

loggedInUser

Returns a user that is currently authenticated. This is the user this GraphQL server makes requests on behalf of.

```
query LoggedInUser{
  loggedInUser{
    uID,
    email,
    name,
    clients {
      uID,
      name,
      isPrimary
    }
  }
}
```

taskById

Returns a task by the given unique identifier. Set of fields are vary depends on the task type. Returns Task or error if no task found.

Request:

```
query taskById {
  taskById(uid: "XXXXXXXXXXXXXXX") {
    uID,
      comment,
      status,
      dueDate,
      closedDate,
      createdAt,
      taskType,
      assignedTo {
        uID
        email,
```

```
        name
      }
  }
}
```

refDocumentById

Returns an reference document by a unique identifier, a file or a reference by URL in the field 'Name'.
Returns RefDocument or error if no document found.
Request:

```
query refDocumentById {
  refDocumentById(uid: "XXXXXXXXXXXXXXX") {
    uID
    name
    size
    createdAt
  }
}
```

authorizationToken

Returns an authorization token by the given username and password.

```
authorizationToken(
        name: String!
        password: String!
        keepAliveTime: Long): AuthToken
```

Parameters:
`name [String!]`
The email address associated with the user's account. Required for user authentication.

`password [String!]`
The user's password. Used in conjunction with the email for credential verification.

`keepAliveTime [Long]`

The optional keep-alive time (in minutes) for the session. If provided, this may extend the duration of the authentication token's validity. Default value is 30 minutes.

Request:

```
query {
  authorizationToken(
    name: "useremail@mail.com"
    password: "pwd!$rty"
    keepAliveTime: 300
  ) {
    token
    expirationDate
  }
}
```

webhooks

Returns list of active webhooks.

```
query  {
  webhooks {
    uID,
    uRL,
    secret,
    eventType
  }
}
```

Arguments

`uID: String`

Optional webhook UID.

timelines

Returns list of timeline events using optional filters.

```
timelines(
        query: String
        dateFrom: DateTime
        dateTo: DateTime
        projectUIDs: [String]
        limit: Int
): [TimeLine]
```

Arguments

`query: String`

Plain text substring to look for in the event descriptions, optional.

`dateFrom: DateTime`

Beginning of the interval of the events' creation UTC date/time to look for, optional, inclusive, optional.

`dateTo: DateTime`

End of the interval of the events' creation UTC date/time to look for, optional, exclusive, optional.

`projectUIDs: [String]`

List of projects identifier to get activities (timelines) for. This parameter is ignored when null or empty, optional.

`limit: Int`

Optional maximum number of records to return, default value is 100, maximum 1000, optional.

Mutations

Mutation	Type
`projectCreate()`	Project
`projectUpdate()`	Project
`projectSetState()`	Project
`assetUpload()`	Asset

Mutation	Type
assetDelete()	Asset
taskCreateUploadAsset()	Task
taskCreateUploadRefDoc	Task
taskCreateReviewAsset	Task
taskCreateUploadAssetExt	Task
taskCreateUploadRefDocsExt	Task
taskCreateReviewAssetExt()	Task
taskApprove()	Task
taskComplete()	Task
taskReject()	Task
taskDelete()	Task
refDocUpload	RefDocument
refDocAddUrl()	RefDocument
refDocDelete()	RefDocument

projectCreate

Creates a new project with the given attributes.

```
projectCreate(
        name: String!
        customer: String
        project: String
        design: String
        revision: String
        description: String
        tags: [String]
        metaData: [StringStringKeyValuePairInput]
```

```
  dueDate: DateOnly
      ): Project
```

Arguments

`name: String!`

Project's name, mandatory.

`customer: String`

Customer name, optional.

`project: String`

Design project name, optional.

`design: String`

Name of the project's design, optional.

`revision: String`

Project's revision name, optional.

`description: String`

Project's description, optional.

`tags: [String]`

List of project's tags, optional.

`metaData: [StringStringKeyValuePairInput]`

List of project's metadata entries, optional.

`dueDate: DateOnly`

Project's completion due date, optional.

projectUpdate

Updates an existing project with the given properties. You may provide one or many properties to update.

```
projectUpdate(
        uid: String!
        name: String
        customer: String
        project: String
        design: String
        revision: String
        description: String
        tags: [String]
```

```
        description: String
        metaData: [StringStringKeyValuePairInput]
        dueDate: DateOnly
): Project
```

Arguments

`uid: String!`

UID of the project to update.

`name: String`

Project's name.

`customer: String`

Customer name.

`project: String`

Design project name.

`design: String`

Name of the project's design.

`revision: String`

Project's revision name.

`description: String`

Project's description.

`tags: [String]`

List of project's tags.

`metaData: [StringStringKeyValuePairInput]`

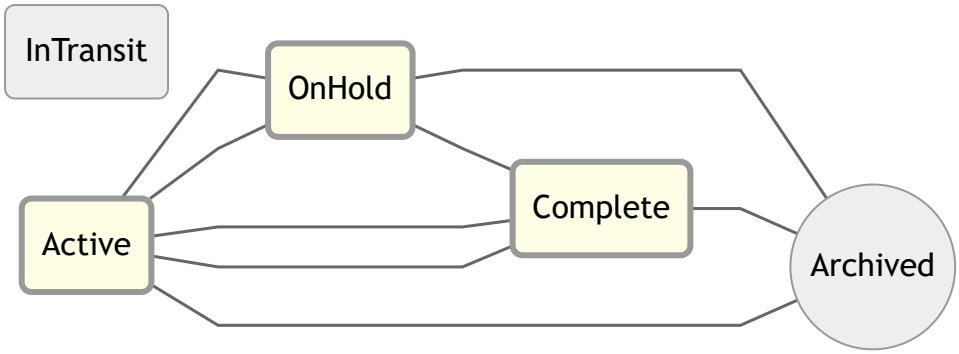
List of project's metadata entries.

`dueDate: DateOnly`

Project's completion due date.

projectSetState

Changes project's state according to the state rules:



The diagram above explains the pre-requisite check for changing the project's state.
Note: The diagram above **is a recommended way** to change project's state, **it is not mandatory to follow it**, but highly recommended.

```
projectSetState(
        uid: String!
        state: ProjectState!
): Project
```

Arguments

`uid: String!`
UID of the project to update.

`state: ProjectState!`
Project's state to set.

assetUpload

Uploads a new asset from the given URL.
ApprovalStudio will download an asset of any appropriate type, process it, and assign to a given project, so that it is available to review, send etc.
Before processing asset, the ApprovalStudio validates all the pre-requisites: files type, file storage space for your account etc. If everything ok, it set the asset status `PENDING` and returns the Asset instance. The asset meanwhile is being processed and will get status `PROCESSED` when processing completes successfully.
Note: GraphQL does not work with files, so if you need to upload a file, use the **Approval Studio REST API**.
Note: Processing files require some time, usually a couple of seconds, sometimes more, up to

minutes, so if you need to know when the uploaded asset is ready to use, pool `assetById()` query waiting for the asset status goes to `PROCESSED` or `FAILED`.

```
assetUpload(
        projectUID: String!
        url: String!
        fileName: String
): Asset
```

assetDelete

Deletes an existing asset by asset UID. When completes, the asset file with all related metadata is deleted, the account's storage space is reclaimed by the asset size.

```
assetDelete(assetUID: String!): Asset
```

taskCreateUploadAsset

Creates an **Asset Uploading Task** and assign it to a given (internal) user.
This task requires a user, the task assigned to, to upload one or more assets to the given project.

```
taskCreateUploadAsset(
        projectUID: String!
        userUID: String!
        dueDate: DateOnly
        comment: String
): Task
```

Arguments

`projectUID: String!`
Project UID the task will be associated with.

`userUID: String!`
UID of the user the task will be assigned to.

`dueDate: DateOnly`

Optional task's due date.

`comment: String`

Optional comment, plain string. A user will this this comment UI and e-mail/Slack/WhatsUp notification.

taskCreateUploadAssetChange

Creates a new **AssetChangeUpload task** for a given asset providing optional due date and comment.

This task is providing **a new version of a given asset**, and this asset, when uploaded, would be the next sequential version of this asset, having the same name and incremented version number.

For instance, you have this in a project:

> - picture.pdf (v1)
> - picture.pdf (v2) <-- current version.

Now you create this task, and when it done you will have:

> - picture.pdf (v1)
> - picture.pdf (v2)
> - picture.pdf (v3) <-- new current version.

```
taskCreateUploadAsset(
        projectUID: String!
        userUID: String!
        assetUID: String!
        dueDate: DateOnly
        comment: String
): Task
```

Arguments

`projectUID: String!`

Project UID the task will be associated with.

`userUID: String!`
UID of the user the task will be assigned to.

`assetUID: String!`
UID of the asset that is requested to upload a new version of.

`dueDate: DateOnly`
Optional task's due date.

`comment: String`
Optional comment, plain string. A user will this this comment UI and e-mail/Slack/WhatsUp notification.

taskCreateUploadRefDoc

Creates a **Reference Document Uploading Task** and assign it to a given user.
This task requires a user, the task assigned to, to upload one or more reference documents to the given project.

```
taskCreateUploadRefDoc(
        projectUID: String!
        userUID: String!
        dueDate: DateOnly
        comment: String
): Task
```

Arguments

`projectUID: String!`
Project UID the task will be associated with.

`userUID: String!`
UID of the user the task will be assigned to.

`dueDate: DateOnly`
Optional task's due date.

`comment: String`
Optional comment, plain string.

taskCreateReviewAsset

Creates a **Asset Review Task** and assign it to a given user.
Asset view task requires that the given user click on the URL he/she will get on UI or email and review the asset(s) make it either Approve or Reject.

```
taskCreateReviewAsset(
        projectUID: String!
        userUID: String!
        assetsUIDs: [String]!
        dueDate: DateOnly
        comment: String
): Task
```

Arguments

`projectUID: String!`
Project UID the task will be associated with.

`userUID: String!`
UID of the user the task will be assigned to.

`assetsUIDs: [String]!`
UIDs of the assets to review.

`dueDate: DateOnly`
Optional task's due date.

`comment: String`
Optional comment, plain string.

taskCreateReviewAssetExt

Creates an **External Asset Review Task** and assign it to a given external user (using email as a user identifier).
The same task a the Review Asset task, but assigned on a person outside the company's account and identified by e-mail address instead of User UID.

```
taskCreateReviewAssetExt(
        projectUID: String!
        email: String!
        assetsUIDs: [String]!
        dueDate: DateOnly
        comment: String
        emailSubject: String
        password: String
        isAllowDownloadAssets: Boolean
        isReadOnly: Boolean
): Task
```

Arguments

`projectUID: String!`
Project UID the task will be associated with.

`email: String!`
Email of the user the task will be assigned to.

`assetsUIDs: [String]!`
UIDs of the assets to review.

`dueDate: DateOnly`
Optional task's due date. When Due Date is provided and is overdue, Approval Studio will notify a user.

`comment: String`
Optional comment, plain string.

`emailSubject: String`
Optional email subject. If email notification are active, a user will get an email describing the task, with this email subject line.

`password: String`
Optional password, plain string.
If password is provided, the ApprovalStudio will ask a user to provide this password to get access to the review tool.

`isAllowDownloadAssets: Boolean`
Optional flag, indicating that the user will be allowed to download any assets files, associated

with the task.

`isReadOnly: Boolean`
Optional read only flag. When set, it makes the review tool read-only, giving the user permission to view only, but prevent to provide any changes, making notes or annotations.

taskCreateUploadAssetExt

Creates an **External Upload Asset Task** and assign it to a given external user (using email as a user identifier).

```
taskCreateUploadAssetExt(
        projectUID: String!
        email: String!
        dueDate: DateOnly
        comment: String
        emailSubject: String
        password: String
): Task
```

Arguments

`projectUID: String!`
Project UID the task will be associated with.

`email: String!`
Email of the user the task will be assigned to.

`dueDate: DateOnly`
Optional task's due date.

`comment: String`
Optional comment, plain string.

`emailSubject: String`
Optional notification email subject, plain string.

`password: String`
Optional password, plain string.

taskCreateUploadRefDocsExt

Creates an **External Upload Reference Document Task** and assign it to a given external user (using email as a user identifier).

```
taskCreateUploadRefDocsExt(
        projectUID: String!
        email: String!
        dueDate: DateOnly
        comment: String
        emailSubject: String
        password: String
): Task
```

Arguments

`projectUID: String!`
Project UID the task will be associated with.

`email: String!`
Email of the user the task will be assigned to.

`dueDate: DateOnly`
Optional task's due date.

`comment: String`
Optional comment, plain string.

`emailSubject: String`
Optional notification email subject, plain string.

`password: String`
Optional password, plain string.

taskApprove

Approves assets(s) associated with a given **Asset Review Task**, internal or external.

```
taskApprove(taskUID: String!): Task
```

Arguments

```
taskUID: String!
```
UID of the task to approve.

taskReject

Reject assets(s) associated with a given **Asset Review Task**, internal or external.

```
taskReject(taskUID: String!): Task
```

Arguments

```
taskUID: String!
```
UID of the task to reject.

taskComplete

Marks a given task as Completed.
This method can be used in conjunction with asset or reference doc uploading.
First you upload a file, then, when uploading is competed, you may close the task.

```
taskComplete(taskUID: String!): Task
```

Arguments

```
taskUID: String!
```
UID of the task to complete.

taskDelete

Deletes a given task.
This marks a task as deleted, it will not appear on UI nor participate in approve/reject/complete calculations.

```
taskDelete(taskUID: String!): Task
```

Arguments

```
taskUID: String!
```
UID of the task to delete.

refDocUpload

Uploads a reference document from the given URL and associate it with the given project.

```
refDocUpload(
        projectUID: String!
        url: String!
        fileName: String
): RefDocument
```

Arguments

```
projectUID: String!
```
Project ID the reference document would be associated to.

```
url: String!
```
Reference document's URL to download from.

```
fileName: String
```
Optional file name; if not provided, the file name would be taken from the URL.

refDocAddUrl

Creates a reference document as a given URL (no file downloading) and associate it with the given project.

```
refDocAddUrl(
        projectUID: String!
        url: String!
): RefDocument
```

Arguments

`projectUID: String!`
Project ID the reference document would be associated to.

`url: String!`
Reference document reference (URL) that to be added to the list of the reference documents as URL.

refDocDelete

Deletes a given reference document, file or URL.

```
refDocDelete(refDocUID: String!): RefDocument
```

Arguments

`refDocUID: String!`
ID of the reference document to delete.

webhookCreate

```
webhookCreate(
        uRL: String!
        eventType: WebhookEventType!
): Webhook
```

Creates a new webhook registration.
Arguments
uRL: String!
URL of the webhook entry point.

eventType: WebhookEventType!
ID of the reference document to delete.

webhookDelete

```
webhookDelete(uID: String!): String
```

workflowCreate

Crates a new workflow.

Arguments

`name: String`
Optional workflow name.

`sActive: Boolean = false`
Optional activity flag.

`actions: [WorkflowActionInput] = []`
List of workflow actions.

`roles: [WorkflowRoleInput] = []`
List of workflow roles, optional.

workflowUpdate

Updates and existing workflow.

Arguments

`uID: String`
Workflow UID.

`name: String`

Optional workflow name.

`isActive: Boolean = false`

Optional activity flag.

`actions: [WorkflowActionInput] = []`

List of workflow actions.

`roles: [WorkflowRoleInput] = []`

List of workflow roles, optional.

workflowDelete

Deletes and existing workflow.

Arguments

`uID: String`

Workflow UID.

workflowRun

Runs a manual action of the workflow, assigned to a given project.

Arguments

`projectUID: String`

Project UID.