GraphQL API Reference (beta)

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.

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

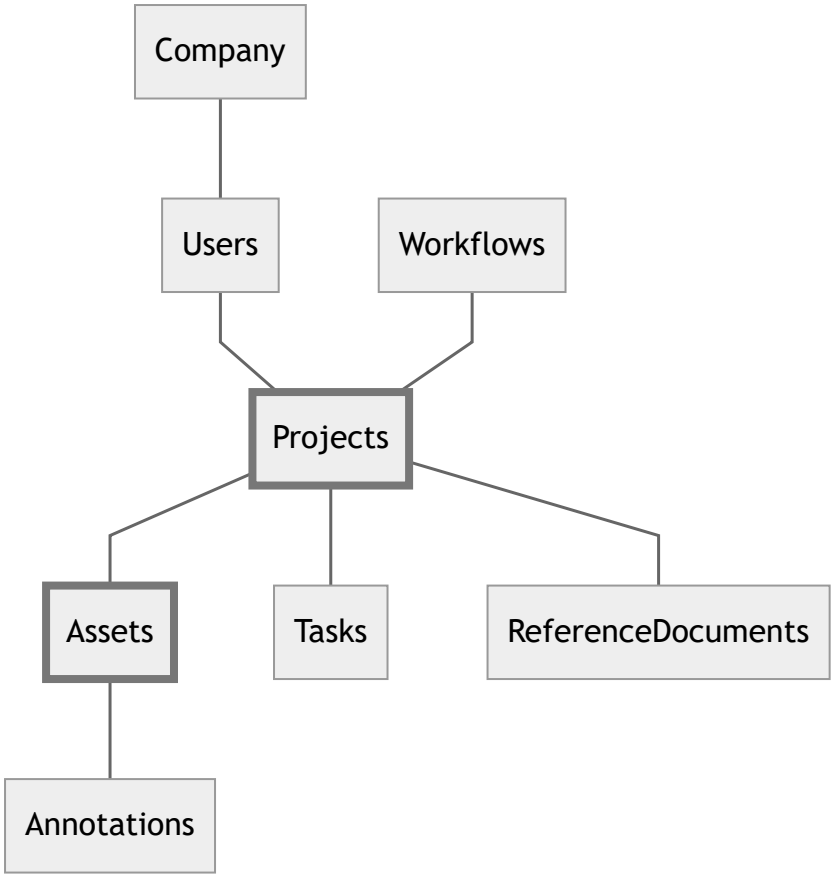
Table of contents

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

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

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

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.

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.
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.

Status	
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).
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.

Type	
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).
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.
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.

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: "XXXXXXXXXXXXXX") {
    uID
    name
    email
  }
}
```

users

Returns all active users in the system.

```
query Users {
  users {
    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: "XXXXXXXXXXXXXX") {
    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.

Mutations

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

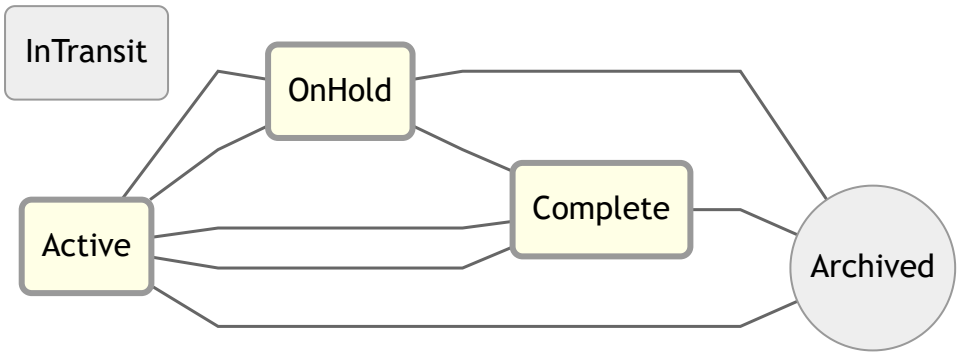
List of project's metadata entries.

`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.

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
```