RMsis APIs (GraphQL)

RMsis provides GraphQL API's on all major entities and relationships. You can use these API's to develop integrations between RMsis and other applications. This guide will help you to understand how to access and utilise these API's.

Prerequisites

This section covers primary information needed to access RMsis API's

JIRA API Authentication

1. RMsis API endpoint i.e. **`<JIRA_BASE_URL>/rest/service/latest/rmsis/graphql`** is part of RMsis plugin.
2. So, you will need to access JIRA authentication API to receive authentication token and subsequently you can access RMsis API endpoint.
3. If you do not have prior knowledge of accessing API in JIRA, please visit JIRA Server platform REST API reference

GraphQL

1. RMsis uses GraphQL as its API standard
2. GraphQL is query language for API's created by Facebook.
3. You can learn more about GraphQL via specification http://facebook.github.io/graphql/ or example implementation https://github.com/facebook/graphql
4. Some more getting started articles,
    - https://graphql.org

Resources

This section covers information about resources available in RMsis API's.

There are two main kind of resources available in RMsis, entities and relationships.

Entity

1. All distinct and independent resources in RMsis are called entity.
2. As RMsis imports some of them from JIRA, we can classify entities as internal and external.
3. Internal Entities
    a. Requirement
        i. Planned
        ii. Unplanned
    b. Release
    c. Test Case (Test Steps as its part)
    d. Test Run
4. External Entity
    a. User
    b. Project
5. In API's,
    a. For internal entities, all CRUD operations are provided, whereas,
    b. For external entities, only retrieve operation is provided (to get RMsis internal ID) and we encourage you to use JIRA Rest API's for other operations.

Relationship

1. Links between entities is called relationship
2. In API's, CRUD operations are provided for relationships

Structure

This section covers structure of API's provided in RMsis.

Request and Response Structure

RMsis follows GraphQL for request and response structure.

Request

Request to GraphQL endpoint can either be HTTP `GET` or `POST` method

1. GET Method

    a. GraphQL operation query can be provided as a query parameter
    b. `<GrapQL_Endpoint>?query=<GraphQL_Operation_Query>`
2. Post Method
    a. GraphQL endpoint consumes `application/json` media type, and operation query is provided as request body
    b. Structure of this as follows

Grapql JSON Request Body

```
{
        query: "<GraphQL_Query>" //required
        variables: { //not required
                //Map of GraphQL variables
        }
        operationName: "<Name_of_Operation>" //not required
}
```

Response

GraphQL returns media type `application/json` as response. HTTP Code 200 or 400 is provided depending upon errors in response. Structure of this response is,

GraphQL Response JSON

```
{
        data: {
                //contains data response
        },
        errors: [
                //optional list of error while executing api
                //if there is any error in response, http status code will be 404.
        ]
}
```

Operation Structure

As per GraphQL standard, API's in RMsis are divided in two operation structures, query and mutation types.

Query Type

API to retrieve any entity or relationships are part of query type. General structure for these API's are

1. For entities
    a. `get<Singular_or_Plural_Entity_Name...>`
    b. `search<Entity_Name...>`
2. For relationships
    a. `getTargetRelationshipEntities`
    b. `getSourceRelationshipEntities`

Mutation Type

API to create, update, or delete are part of mutation type. General structure for these API's are

1. For entities
    a. `create<Entity_Name>`
    b. `update<Entity_Name>`
    c. `delete<Entity_Name>`
2. For relationships
    a. `createRelationship`
    b. `deleteRelationship`

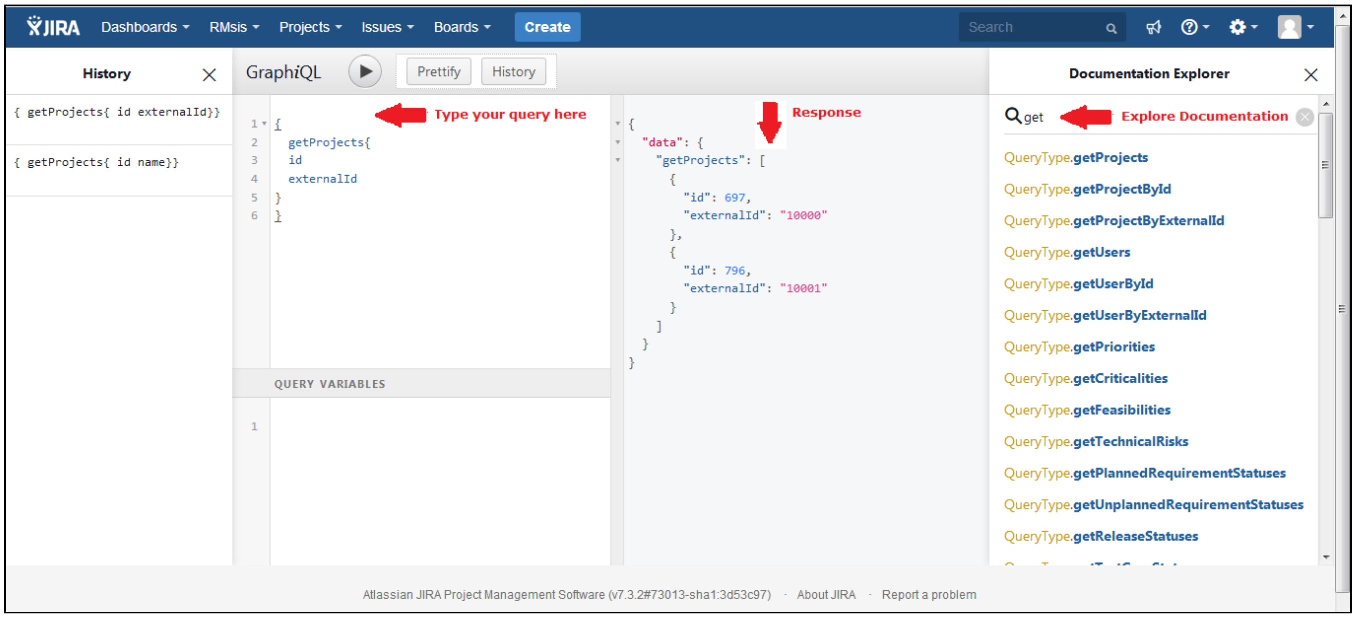
More information about Structure of API's can be obtained from GraphQL Schema file.

Permissions

Permissions granted to users are same as the ones configured for a specific user.

API Playground

RMsis provides GraphiQL Playground (https://github.com/graphql/graphiql) build along side RMsis plugin for API documentation and testing API calls against your system. It can be accessed from your **JIRA Menu Bar >> RMsis >> API Playground** (`<Jira_base_url>/secure/ApiPlayground.jspa`). A view of the GraphiQL user interface is presented below:



```
{ getProjects{ id externalId}}
{ getProjects{ id name}}
```

```
{
  getProjects{
  id
  externalId
}
}
```

```
{
  "data": {
    "getProjects": [
      {
        "id": 697,
        "externalId": "10000"
      },
      {
        "id": 796,
        "externalId": "10001"
      }
    ]
  }
}
```

QueryType.getProjects
QueryType.getProjectById
QueryType.getProjectByExternalId
QueryType.getUsers
QueryType.getUserById
QueryType.getUserByExternalId
QueryType.getPriorities
QueryType.getCriticalities
QueryType.getFeasibilities
QueryType.getTechnicalRisks
QueryType.getPlannedRequirementStatuses
QueryType.getUnplannedRequirementStatuses
QueryType.getReleaseStatuses

Samples

Few samples based on RMsis GraphQL,

Application Samples

Here are links to some applications created using RMsis GraphQL,

- https://bitbucket.org/optimizory-dev/rmsis-api-widgets
- Export RMsis data to a Word document
- Export RMsis data to an Excel document

Sample Request and Response

This section contains request and response samples based on current RMsis GraphQL implementation.

Information	Request Query	Response Code	Response JSON

Fetch list of Priority in system	`{ getPriorities { id name } }`	200	`{ "data": { "getPriorities": [ { "id": 1, "name": "Blocker" }, { "id": 2, "name": "Critical" }, { "id": 3, "name": "Major" }, { "id": 4, "name": "Minor" }, { "id": 5, "name": "Trivial" } ] } }`
Fetch list of Users available in RMsis	`{ getUsers { id externalId } }`	200	`{ "data": { "getUsers": [ { "id": 703, "externalId": "admin" }, { "id": 699, "externalId": "userone" }, { "id": 701, "externalId": "test" }, { "id": 705, "externalId": "usertwo" } ] } }`

Fetch single requirement by ID

```
{
  getRequirementById(id: 1) {
    id
    summary
    description
    externalId
    createdAt
    categories {
      id
      name
    }
    reporter {
      id
    }
    priority {
      name
    }
  }
}
```

200

```
 {
  "data": {
    "getRequirementById": {
      "id": 1,
      "summary": "Requirement 1",
      "description": "Requirement <p>description</p>",
      "externalId": null,
      "createdAt": "22 Oct 1988 05:30 AM",
      "categories": [
        {
          "id": 1001,
          "name": "Category 1"
        },
        {
          "id": 1002,
          "name": "Category 2"
        }
      ],
      "reporter": {
        "id": 24
      },
      "priority": {
        "name": "Major"
      }
    }
  }
}
```

Fetch requirement by ID

```
{
  getRequirementById(id: 222) {
    id
    summary
  }
}
```

404

```
{
  "data": {
    "getRequirementById": null
  },
  "errors": [
    {
      "message": "Requirement with ID: 2 not found",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "getRequirementById"
      ]
    }
  ]
}
```

Create planned requirement	`mutation { createPlannedRequirement(projectId: 709, summary: "New Requirement Summary") { id summary createdAt } }`	200	`{ "data": { "createPlannedRequirement": { "id": 2, "summary": "New Requirement Summary", "createdAt": "16 Apr 2018 12:55 PM" } } }`
Delete Test Case	`mutation { deleteTestCase(id: 5) }`	200	`{ "data": { "deleteTestCase": true } }`
Fetch linked artifacts of a requirement	`{ getTargetRelationshipEntities(name: REQUIREMENT_ARTIFACT, sourceId: 1074) }`	200	`{ "data": { "getTargetRelationshipEntities": [ 839 ] } }`
Fetch artifact externalID using RMsis ID	`{ getArtifactById(id: 839) { id externalId } }`	200	`{ "data": { "getArtifactById": { "id": 839, "externalId": "10023" } } }`

Known Limitations

This section covers known limitations of current implementation.

1. For external entities (i.e. JIRA entities), only internal and external ID is return
    a. we encourage you to use JIRA Rest API's for more information.
2. Meta API's for custom fields are currently not available.
3. API's for privileged operations like **version, commit,** etc. are currently not available.
4. Some operations requiring a UI context (like **move** etc.) are not yet provided.
5. API's for configuration and administration are currently not available.