Using the Explorer

In this article

About the GraphQL Explorer
Using the Altair GraphQL Client IDE
Accessing the sidebar docs
Using the variable pane
Requesting support
Troubleshooting errors

You can run queries on real GitHub data using the GraphQL Explorer, an integrated development environment in your browser that includes docs, syntax highlighting, and validation errors.

About the GraphQL Explorer

Note: If your GitHub Enterprise Cloud organization uses GitHub's IP allow list, you won't be able to use the GraphQL Explorer. Instead, we recommend using an alternative GraphQL client IDE.

GraphQL Explorer is an instance of GraphiQL, which is a "graphical interactive in-browser GraphQL IDE."

Using the Altair GraphQL Client IDE

There are many open source GraphQL client IDEs. For example, you can use Altair to access GitHub's GraphQL API. To access the GraphQL API with Altair, download and install it from altair-graphql/altair. Then, follow the configuration steps below.

Configuring Altair

1. Get an access token.
2. Launch Altair.
3. In the left sidebar, below the Altair logo, click **Set Headers**. A new window will open.
4. In the "Header key" field, enter `Authorization`.
5. In the "Header value" field, enter `Bearer TOKEN`, replacing `TOKEN` with your token from the first step.
6. Click **Save** in the bottom right corner of the window to save your authorization header.
7. In the "GraphQL Endpoint" field, enter `https://api.github.com/graphql`.

8. To load the GitHub GraphQL schema, download the public schema.
9. In Altair, click on **Docs** on the top right, then the three dots and **Load Schema...**
10. Select the file public schema that you downloaded in an earlier step.

> **Note**: For more information about why `POST` is the method, see "Forming calls with GraphQL."

You can test your access by querying yourself:

```
query {
  viewer {
    login
  }
}
```

If everything worked correctly, this will display your login. You're all set to start making queries.

Accessing the sidebar docs

All types in a GraphQL schema include a `description` field compiled into documentation. The collapsible **Docs** pane on the right side of the Explorer page allows you to browse documentation about the type system. The docs are automatically updated and will drop deprecated fields.

> The **Docs** sidebar contains the same content that is automatically generated from the schema under "GitHub GraphQL API documentation," though it is formatted differently in places.

Using the variable pane

Some example calls include variables written like this:

```
query($number_of_repos:Int!){
  viewer {
    name
     repositories(last: $number_of_repos) {
       nodes {
         name
       }
     }
   }
}
variables {
   "number_of_repos": 3
}
```

This is the correct format to submit the call using a `POST` request in a `curl` command (as long as you escape newlines).

If you want to run the call in the Explorer, enter the `query` segment in the main pane and the variables in the **Query Variables** pane below it. Omit the word `variables` from the Explorer:

```
{
   "number_of_repos": 3
```

```
}
```

Requesting support

For questions, bug reports, and discussions about GitHub Apps, OAuth apps, and API development, explore the APIs and Integrations discussions on GitHub Community. The discussions are moderated and maintained by GitHub staff, but questions posted to the forum are not guaranteed to receive a reply from GitHub staff.

Consider reaching out to GitHub Support directly using the contact form for:

- guaranteed response from GitHub Enterprise Cloud staff
- support requests involving sensitive data or private concerns
- feature requests
- feedback about GitHub Enterprise Cloud products

Troubleshooting errors

Because GraphQL is introspective, the Explorer supports:

- Intelligent typeaheads aware of the current schema
- Validation error previews as you type

If you enter a query that is not well-formed or does not pass schema validation, a popup warns you of an error. If you run the query, the error returns in the response pane.

A GraphQL response contains several keys: a `data` hash and an `errors` array.

```
{
  "data": null,
  "errors": [
    {
      "message": "Objects must have selections (field 'nodes' returns Repository
but has no selections)",
      "locations": [
        {
          "line": 5,
          "column": 8
        }
      ]
    }
  ]
}
```

It's possible you might run into an unexpected error that is not related to the schema. If this happens, the message will include a reference code you can use when reporting the issue:

```
{
  "data": null,
  "errors": [
    {
      "message": "Something went wrong while executing your query. This is most
likely a GitHub bug. Please include \"7571:3FF6:552G94B:69F45B7:5913BBEQ\" when
reporting this issue."
    }
  ]
}
```

Note: GitHub recommends checking for errors before using data in a production environment. In

GraphQL, failure is not total: portions of GraphQL queries may succeed while others fail.