# Using global node IDs

You can get global node IDs of objects via the REST API and use them in GraphQL operations.

## In this article

You can access most objects in GitHub (users, issues, pull requests, etc.) using either the REST API or the GraphQL API. With a recent update, you can find the **global node ID** of many objects from within the REST API and use these IDs in your GraphQL operations.

**Note:** In REST, the global node ID field is named `node_id`. In GraphQL, it's an `id` field on the `node` interface. For a refresher on what "node" means in GraphQL, see "Introduction to GraphQL."

## Putting global node IDs to use

You can follow three steps to use global node IDs effectively:

1. Call a REST endpoint that returns an object's `node_id`.

2. Find the object's type in GraphQL.

3. Use the ID and type to do a direct node lookup in GraphQL.

Let's walk through an example.

## 1. Call a REST endpoint that returns an object's node ID

If you request the authenticated user:

```
$ curl -i -u username:token https://api.github.com/user
```

you'll get a response that includes the `node_id` of the authenticated user:

```
{
  "login": "octocat",
  "id": 1,
  "avatar_url": "https://github.com/images/error/octocat_happy.gif",
  "gravatar_id": "",
  "url": "https://api.github.com/users/octocat",
  "html_url": "https://github.com/octocat",
  "followers_url": "https://api.github.com/users/octocat/followers",
  "following_url": "https://api.github.com/users/octocat/following{/other_user}",
  "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/octocat/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
  "organizations_url": "https://api.github.com/users/octocat/orgs",
  "repos_url": "https://api.github.com/users/octocat/repos",
  "events_url": "https://api.github.com/users/octocat/events{/privacy}",
  "received_events_url": "https://api.github.com/users/octocat/received_events",
  "type": "User",
  "site_admin": false,
  "name": "monalisa octocat",
  "company": "GitHub",
  "blog": "https://github.com/blog",
  "location": "San Francisco",
  "email": "octocat@github.com",
  "hireable": false,
  "bio": "There once was...",
  "public_repos": 2,
  "public_gists": 1,
  "followers": 20,
  "following": 0,
  "created_at": "2008-01-14T04:33:35Z",
  "updated_at": "2008-01-14T04:33:35Z",
  "private_gists": 81,
  "total_private_repos": 100,
  "owned_private_repos": 100,
  "disk_usage": 10000,
  "collaborators": 8,
  "two_factor_authentication": true,
  "plan": {
    "name": "Medium",
    "space": 400,
    "private_repos": 20,
    "collaborators": 0
  },
  "node_id": "MDQ6VXNlcjU4MzIzMQ=="
}
```

## 2. Find the object type in GraphQL

In this example, the `node_id` value is `MDQ6VXNlcjU4MzIzMQ==`. You can use this value to query the same object in GraphQL.

You'll need to know the object's *type* first, though. You can check the type with a simple GraphQL query:

```
query {
  node(id:"MDQ6VXNlcjU4MzIzMQ==") {
     __typename
  }
}
```

This type of query—that is, finding the node by ID—is known as a "direct node lookup."

When you run this query, you'll see that the `__typename` is `User`.

## 3. Do a direct node lookup in GraphQL

Once you've confirmed the type, you can use an inline fragment to access the object by its ID and return additional data. In this example, we define the fields on `User` that we'd like to query:

```
query {
  node(id:"MDQ6VXNlcjU4MzIzMQ==") {
   ... on User {
      name
      login
    }
  }
}
```

This type of query is the standard approach for looking up an object by its global node ID.

## Using global node IDs in migrations

When building integrations that use either the REST API or the GraphQL API, it's best practice to persist the global node ID so you can easily reference objects across API versions. For more information on handling the transition between REST and GraphQL, see "Migrating from REST to GraphQL."