Using the GraphQL API for Discussions

In this article

Fields
Objects
Interfaces
Mutations
Search

Learn how to use the GitHub Discussions GraphQL API.

The GitHub Discussions GraphQL API allows you to get, create, edit, and delete discussion posts. For more information about GitHub Discussions, see "About discussions."

This API is available for authenticated users, OAuth apps, and GitHub Apps. Access tokens require the `repo` scope for private repositories and the `public_repo` scope for public repositories. For more information, see "Scopes for OAuth apps."

Fields

Repository.discussions

List the discussions within a repository. If `categoryId` is specified, only results within that category will be returned. If `answered` is not specified, both answered and unanswered discussions will be returned.

Signature:

```
discussions(
  after: String,
  before: String,
  first: Int,
  last: Int,
  categoryId: ID = null,
  answered: Boolean = null,
  orderBy: DiscussionOrder = {field: UPDATED_AT, direction: DESC}
) : Discussion
```

DiscussionOrder

```
"""
Ways in which discussions can be ordered.
"""
input DiscussionOrder {
  """
  The field by which to order discussions.
  """
  field: DiscussionOrderField!

  """
```

```
  The direction in which to order discussions by the specified field.
  """
  direction: OrderDirection!
}
```

```
"""
Properties by which discussion connections can be ordered.
"""
enum DiscussionOrderField {
  """
  Order discussions by creation time.
  """
  CREATED_AT

  """
  Order discussions by most recent modification time.
  """
  UPDATED_AT
}
```

Repository.discussionCategories

Return the available discussion categories defined within this repository. Each repository may have up to 25 categories. For more information about discussion categories, see "About discussions."

Signature:

```
discussionCategories(
  after: String,
  before: String,
  first: Int,
  last: Int,
) : DiscussionCategoryConnection!
```

Repository.discussion

Get a discussion. Returns `null` if discussion with the specified ID does not exist.

Signature:

```
discussion(number: Int!) : Discussion
```

Repository.pinnedDiscussions

Return discussions pinned to this repository, ordered by pin position.

Signature:

```
pinnedDiscussions(
  after: String,
  before: String,
  first: Int,
  last: Int,
) : PinnedDiscussionConnection!
```

Objects

Note: For brevity, connection types are not expanded here. Each connection type mentioned in the schema follows the same pattern as other connections in the GraphQL API. For more information, see "Introduction to GraphQL."

```
query {
  repository(owner: "github", name: "some-repo") {
    discussions(first: 10) {
      # type: DiscussionConnection
      totalCount # Int!

      pageInfo {
        # type: PageInfo (from the public schema)
        startCursor
        endCursor
        hasNextPage
        hasPreviousPage
      }

      edges {
        # type: DiscussionEdge
        cursor
        node {
          # type: Discussion
          id
        }
      }

      nodes {
        # type: Discussion
        id
      }
    }
  }
}
```

Discussion

▶ Fields:

DiscussionComment

▶ Fields

DiscussionCategory

▶ Fields

PinnedDiscussion

▶ Fields:

PinnedDiscussionPattern

▶ Values

PinnedDiscussionGradient

▶ Values

Interfaces

RepositoryDiscussionAuthor

Implemented by the `User` and `Organization` types. **Note:** An `Organization` will only have discussions associated with it if it was converted from a `User`.

▶ Fields

RepositoryDiscussionCommentAuthor

Also implemented by the `User` and `Organization` types.

▶ Fields

Mutations

These mutations follow the same implementation pattern that other mutations in the GraphQL API. Each mutation accepts a single argument of an `Input` type, named after the mutation, and returns a `Payload` type containing the fields specified.

For example, this is a basic `createDiscussion` mutation that will create a new discussion:

```
mutation {
  # input type: CreateDiscussionInput
  createDiscussion(input: {repositoryId: "1234", categoryId: "5678", body: "The
body", title: "The title"}) {

    # response type: CreateDiscussionPayload
    discussion {
      id
    }
  }
}
```

createDiscussion

Input fields:

- `body: String!` The body of the new discussion.
- `title: String!` The title of the new discussion.
- `repositoryId: ID!` The ID of a repository in which to create the discussion.
- `categoryId: ID!` The ID of a `DiscussionCategory` within this repository.
- `clientMutationId: String` A unique identifier for the client performing the mutation.

Return type fields:

- `clientMutationId: String` The unique identifier provided as an input.
- `discussion: Discussion` The discussion that was created.

updateDiscussion

Input fields:

- `discussionId: ID!` The node ID of the discussion to update.
- `body: String` The new contents of the discussion body.
- `title: String` The new discussion title.
- `categoryId: ID` The node ID of a `DiscussionCategory` within the same repository to change this discussion to.
- `clientMutationId: String` A unique identifier for the client performing the mutation.

Return type fields:

- `clientMutationId: String` The unique identifier provided as an input.
- `discussion: Discussion` The discussion that was modified.

deleteDiscussion

Input fields:

- `id: ID!` The node ID of the discussion to delete.
- `clientMutationId: String` A unique identifier for the client performing the mutation.

Return type fields:

- `clientMutationId: String` The unique identifier provided as an input.
- `discussion: Discussion` The discussion that was deleted.

addDiscussionComment

Input fields:

- `body: String!` The contents of the comment.
- `discussionId: ID!` The node ID of the discussion to comment on.
- `replyToId: ID` The node ID of the discussion comment to reply to. If absent, the created comment will be a top-level comment.
- `clientMutationId: String` A unique identifier for the client performing the mutation.

Return type fields:

- `clientMutationId: String` The unique identifier provided as an input.
- `comment: DiscussionComment` The discussion comment that was created.

updateDiscussionComment

Input fields:

- `body: String!` The new contents of the comment body.
- `commentId: ID!` The node ID of the discussion comment to update.
- `clientMutationId: String` A unique identifier for the client performing the mutation.

Return type fields:

- `clientMutationId: String` The unique identifier provided as an input.
- `comment: DiscussionComment` The discussion comment that was updated.

deleteDiscussionComment

Input fields:

- `id: ID!` The node ID of the discussion comment to delete.
- `clientMutationId: String` A unique identifier for the client performing the mutation.

Return type fields:

- `clientMutationId: String` The unique identifier provided as an input.
- `comment: DiscussionComment` The discussion comment that was deleted.

markDiscussionCommentAsAnswer

Input fields:

- `id: ID!` The node ID of the discussion comment to mark as an answer.
- `clientMutationId: String` A unique identifier for the client performing the mutation.

Return type fields:

- `clientMutationId: String` The unique identifier provided as an input.
- `discussion: Discussion` The discussion that includes the chosen comment.

unmarkDiscussionCommentAsAnswer

Input fields:

- `id: ID!` The node ID of the discussion comment to unmark as an answer.
- `clientMutationId: String` A unique identifier for the client performing the mutation.

Return type fields:

- `clientMutationId: String` The unique identifier provided as an input.
- `discussion: Discussion` The discussion that includes the unmarked comment.

Search

Discussion may be returned from the top-level `search` field. To search for discussion, specify `type` as `DISCUSSION`. The `SearchResultItemConnection` type has a `discussionCount` field to report the number of returned discussions, and the `Discussion` type is added to the `SearchResultItem` union. For more information, see "Queries" and "Searching discussions."