Using the Graph API

The Graph API is the primary way to get data in and out of Facebook's social graph. It's a low-level HTTP-based API that is used to query data, post new stories, upload photos and a variety of other tasks that an app might need to do. This guide explains how to accomplish all these things in the Graph API.

Basics

We cover the basics of Graph API terminology and structure in our Graph API overview. The following sections explain more about the different operations that can be performed using the API.

Reading

All nodes and edges in the Graph API can be read simply with an HTTP `GET` request to the relevant endpoint. For example, if you wanted to retrieve information about the current user, you would make an HTTP `GET` request as below:

```
GET /v2.5/me HTTP/1.1
Host: graph.facebook.com
```

Most API calls must be signed with an access token. You can determine which permissions are needed in this access token by looking at the Graph API reference for the node or edge that you wish to read. You can also use the Graph API Explorer to quickly generate tokens in order to play about with the API and discover how it works.

> The `/me` node is a special endpoint that translates to the `user_id` of the person (or the `page_id` of the Facebook Page) whose access token is currently being used to make the API calls. If you had a user access token, you could retrieve all of a user's photos by using:
>
> ```
> GET graph.facebook.com
>   /me/photos
> ```

The response you receive varies based on the node or edge that you are reading, but it responses take the following general form:

```
{
   "fieldname": {field-value},
```

```
    ....
}
```

Choosing Fields

By default, not all fields in a node or edge are returned when you make a query. You can choose the fields or edges that you want returned with the `fields` query parameter. This is really useful for making your API calls more efficient and fast.

For example, the following Graph API call `https://graph.facebook.com/bgolub?fields=id,name,picture` will only return the id, name, and picture in Ben's profile:

```
GET graph.facebook.com
  /bgolub?
    fields=id,name,picture
```

Ordering

You can order certain data sets chronologically. For example you may sort a photo's comments in reverse chronological order using the key `reverse_chronological`:

```
GET graph.facebook.com
  /{photo-id}?
    fields=comments.order(reverse_chronological)
```

`order` must be one of the following values:

- `chronological`
- `reverse_chronological`

URL Lookup

Most objects can be discovered using their IDs, but there are times where this is not possible and all you have is a URL to identify something.

You can use the URL node that was introduced in v2.1 to return the IDs of Open Graph Object URLs, or to find data associated with an App Link URL.

Learn more about finding App Link data with the Index API in our documentation for App Links.

Modifying API Requests

Some API endpoints can be used with extra parameters that will modify the request. The nature of what these modifiers do varies, so we have documented each of them separately in each of the API reference documents where appropriate. Below is a list of common modifiers that can be used with most endpoints.

Name	Description	Type
`return_ssl_resources`	Used when you require a picture to be returned over a secure connection (HTTPS) to avoid mixed content warnings in browsers.	`bool`
`locale`	Used if your app needs the ability to retrieve localized content in the language of a particular locale (when available).	`Locale`

Other modifiers for paging and introspection are shown below.

Making Nested Requests

The field expansion feature of the Graph API allows you to effectively nest multiple graph queries into a single call. Certain resources, including most of Ads API, are unable to utilize field expansion on some or all connections.

For example, in a single call, you can ask for the first N photos of the first K albums. The response maintains the data hierarchy so developers do not have to weave the data together on the client.

This is different from the batch Requests feature which allows you to make multiple, potentially unrelated, Graph API calls in a single request.

Here is the general format that field expansion takes:

```
GET graph.facebook.com
  /{node-id}?
    fields=<first-level>{<second-level>}
```

`<first-level>` in this case would be one or more (comma-separated) fields or edges from the parent node. `<second-level>` would be one or more (comma-separated) fields or edges from the first-level node.

There is no limitation to the amount of nesting of levels that can occur here. You can also use a `.limit(n)` argument on each field or edge to restrict how many objects you want to get.

This is easier to understand by seeing it in action, so here's an example query that will retrieve up to five albums created by someone, and the last five posts in their feed.

```
GET graph.facebook.com
  /me?
    fields=albums.limit(5),posts.limit(5)
```

We can then extend this a bit more and for each album object, also retrieve the first two photos, and people tagged in each photo:

```
GET graph.facebook.com
  /me?
    fields=albums.limit(5){name, photos.limit(2)},posts.limit(5)
```

Now we can extend it again by retrieving the name of each photo, the picture URL, and the people tagged:

```
GET graph.facebook.com
  /me?
    fields=albums.limit(5){name, photos.limit(2){name, picture, tags.limit(2)}},posts.limit(5)
```

You can see how field expansion can work across nodes, edges, and fields to return really complex data in a single request.

Traversing Paged Results

When you make an API request to a node or edge, you usually don't receive all of the results of that request in a single response. This is because some responses could contain thousands of objects so most responses are paginated by default.

Cursor-based Pagination

Cursor-based pagination is the most efficient method of paging and should always be used where possible. A cursor refers to a random string of characters which marks a specific item in a list of data. Unless this item is deleted, the cursor will always point to the same part of the list, but is be invalidated if an item is removed. Therefore, your app shouldn't store any older cursors or assume that they will still be valid.

When reading an edge that supports cursor pagination, you will see the following JSON response:

```
{
  "data": [
     ... Endpoint data is here
  ],
  "paging": {
```

```
    "cursors": {
      "after": "MTAxNTExOTQ1MjAwNzI5NDE=",
      "before": "NDMyNzQyODI3OTQw"
    },
    "previous": "https://graph.facebook.com/me/albums?limit=25&before=NDMyNzQyODI3OTQw"
    "next": "https://graph.facebook.com/me/albums?limit=25&after=MTAxNTExOTQ1MjAwNzI5NDE="
  }
}
```

A cursor-paginated edge supports the following parameters:

- `before` : This is the cursor that points to the start of the page of data that has been returned.
- `after` : This is the cursor that points to the end of the page of data that has been returned.
- `limit` : This is the maximum number of objects that *may* be returned. A query may return fewer than the value of `limit` due to filtering. Do not depend on the number of results being fewer than the `limit` value to indicate your query reached the end of the list of data, use the absence of `next` instead as described below. For example, if you set `limit` to 10 and 9 results are returned, there may be more data available, but one item was removed due to privacy filtering. Some edges may also have a maximum on the `limit` value for performance reasons. In all cases, the API returns the correct pagination links.
- `next` : The Graph API endpoint that will return the next page of data. If not included, this is the last page of data. Due to how pagination works with visibility and privacy, it is possible that a page may be empty but contain a 'next' paging link. Stop paging when the 'next' link no longer appears.
- `previous` : The Graph API endpoint that will return the previous page of data. If not included, this is the first page of data.

> Don't store cursors. Cursors can quickly become invalid if items are added or deleted.

Time-based Pagination

Time pagination is used to navigate through results data using Unix timestamps which point to specific times in a list of data.

When using an endpoint that uses time-based pagination, you will see the following JSON response:

```
{
  "data": [
     ... Endpoint data is here
  ],
  "paging": {
    "previous": "https://graph.facebook.com/me/feed?limit=25&since=1364849754",
    "next": "https://graph.facebook.com/me/feed?limit=25&until=1364587774"
  }
}
```

A time-paginated edge supports the following parameters:

- `until` : A Unix timestamp or `strtotime` data value that points to the end of the range of time-based data.
- `since` : A Unix timestamp or `strtotime` data value that points to the start of the range of time-based data.
- `limit` : This is the maximum number of objects that *may* be returned. A query may return fewer than the value of `limit` due to filtering. Do not depend on the number of results being fewer than the `limit` value to indicate your query reached the end of the list of data, use the absence of `next` instead as described below. For example, if you set `limit` to 10 and 9 results are returned, there may be more data available, but one item was removed due to privacy filtering. Some edges may also have a maximum on the `limit` value for performance reasons. In all cases, the API returns the correct pagination links.
- `next` : The Graph API endpoint that will return the next page of data.
- `previous` : The Graph API endpoint that will return the previous page of data.

> For consistent results, specify both `since` and `until` parameters. Also, it is recommended that the time difference is a maximum of 6 months.

Offset-based Pagination

Offset pagination can be used when you do not care about chronology and just want a specific number of objects returned. This should only be used if the edge does not support cursor or time-based pagination.

An offset-paginated edge supports the following parameters:

- `offset` : This offsets the start of each page by the number specified.
- `limit` : This is the maximum number of objects that *may* be returned. A query may return fewer than the value of `limit` due to filtering. Do not depend on the number of results being fewer than the `limit` value to indicate your query reached the end of the list of data, use the absence of `next` instead as described below. For example, if you set `limit` to 10 and 9 results are returned, there may be more data available, but one item was removed due to privacy filtering. Some edges may also have a maximum on the `limit` value for performance reasons. In all cases, the API returns the correct pagination links.
- `next` : The Graph API endpoint that will return the next page of data. If not included, this is the last page of data. Due to how pagination works with visibility and privacy, it is possible that a page may be empty but contain a 'next' paging link. Stop paging when the 'next' link no longer appears.
- `previous` : The Graph API endpoint that will return the previous page of data. If not included, this is the first page of data.

Note that if new objects are added to the list of items being paged, the contents of each offset-based page will change.

> Offset based pagination is not supported for all API calls. To get consistent results, we recommend you to paginate using the previous/next links we return in the response.

Making Large Requests

Some Graph API endpoints can take parameters that are very large. If your requests end up being larger than a couple thousand characters, you may start seeing server errors since our servers will reject very large `GET` requests.

As a best practice, for large requests use a `POST` request instead of a `GET` request and add a `method=GET` parameter. If you do this, the `POST` will be interpreted as if it were a `GET`.

Making Multiple Requests

The standard version of the Graph API is designed to make it easy to get data for an individual object and to browse connections between objects. It also includes a limited ability to retrieve data for a few objects at the same time.

If your app needs the ability to access significant amounts of data at once, or you need to make changes to several objects at once, it is often more efficient to batch your queries rather than make multiple individual HTTP requests.

To enable this, the Graph API supports a number of functions, including multiple-ID lookup and batching. Batch requests are explained in a separate guide but you can read more about multiple ID lookup below.

Multiple ID Read Requests

You can make a single `GET` request that retrieves multiple nodes by using the `?ids` endpoint with the object IDs of those nodes. For example, to lookup the Facebook Developers page and the current session user in a single request, you could use the following Graph API call:

```
GET graph.facebook.com
  /?ids=platform,me
```

Which is equivalent to the following individual API requests:

```
GET graph.facebook.com
  /platform

GET graph.facebook.com
  /me
```

The returned data will look something like this:

```
{
  "me": {
    "id": "1234567890"
    ... // Other fields
  },
  "platform": {
    "id": "19292868552"
    ... // Other fields
```

```
  }
}
```

This can also work with edges as long as all the requested IDs have the requested edge. For example:

```
GET graph.facebook.com
  /photos?ids={user-id-a},{user-id-b}
```

Is equivalent to the following individual API requests:

```
GET graph.facebook.com
  /{user-id-a}/photos

GET graph.facebook.com
  /{user-id-b}/photos
```

The returned data will look something like this:

```
{
  "{user-id-a}": {
    "data": [
      {
        "id": "12345",
        "picture": "{photo-url}",
        "created_time": "2014-07-15T15:11:25+0000"
      }
      ... // More photos
    ]
  },
  "{user-id-b}": {
    "data": [
      {
        "id": "56789",
        "picture": "{photo-url}",
        "created_time": "2014-01-15T12:24:47+0000"
      }
      ... // More photos
    ]
  },
}
```

Note that while field filtering, field expansion, and limiting will work with these multiple-ID lookups, you will get an error if one of the objects does not have any of the requested fields. For example, if you were to lookup a Page and a User using this method, and then filter on `fields=id,picture,gender` the request would fail because Pages do not have a `gender` field.

Introspection

The Graph API supports introspection of nodes. This enables you to see all of the edges a node has without knowing its type ahead of time. To get this information, add `metadata=1` to the Graph API request:

```
GET graph.facebook.com
  /{node-id}?
    metadata=1
```

The resulting JSON will include a `metadata` property that lists all the supported edges for the given node:

```
{
   "name": {node-name},
   "metadata": {
      "connections": {
         "feed": "http://graph.facebook.com/me/feed",
         "picture": "https://graph.facebook.com/me/picture",
         ....
      }
```

```
      "fields": [
        {
          "name": "id",
          "description": "The user's Facebook ID. No `access_token` required. `string`."
        },
        ....
      ],
      "type": "user"
    }
}
```

Publishing

Most nodes in the Graph API have edges that can be publishing targets, such as `/{user-id}/feed` or `/{album-id}/photos`. All Graph API publishing is done with an HTTP `POST` request to the relevant edge with any necessary parameters included. For example, to publish a post on behalf of someone, you would make an HTTP `POST` request as below:

```
POST graph.facebook.com
  /{user-id}/feed?
    message={message}&
    access_token={access-token}
```

All publishing calls must be signed with an access token. You can determine which permissions are needed in this access token by looking at the Graph API reference for the node to which you wish to publish.

There are a large number of edges that can be publishing targets. Details can be found in the reference doc for each node.

The Common Scenarios for Graph API guide contains additional information for a few common publishing scenarios.

Making Multiple Requests
You can chain together publish calls with read calls by using Batch Requests. For additional information, see Making Multiple API Requests.

Updating

All Graph API updating is done with an HTTP `POST` request to the relevant node with any updated parameters included:

```
POST graph.facebook.com
  /{node-id}?
    {updated-field}={new-value}&
    access_token={access-token}
```

All update calls must be signed with an access token with the same permissions needed for publishing to that endpoint, as per the Graph API reference for the node that you wish to update.

Deleting

Delete nodes from the graph by sending HTTP `DELETE` requests to them:

```
DELETE graph.facebook.com
  /{node-id}?
    access_token=...
```

Generally speaking, an app can only delete content it created. Check the reference guide for the node or edge for more information.

To support clients that do not support all HTTP methods, you can alternatively issue a `POST` request to an endpoint with the additional argument `method=delete` to override the HTTP method. For example, you can delete a comment by issuing the following request:

```
POST graph.facebook.com
  /{comment-id}?
    method=delete
```

Searching

You can search over many public objects in the social graph with the `/search` endpoint. The syntax for search is:

```
GET graph.facebook.com
  /search?
    q={your-query}&
    type={object-type}
```

All Graph API search queries require an access token included in the request. You need a user access token to execute a search.

Available Search Types

We support search for the following types:

Type	Description	`q` value
`user`	Search for a person (if they allow their name to be searched for).	Name
`page`	Search for a Page.	Name
`event`	Search for an event.	Name
`group`	Search for a Group.	Name
`place`	Search for a place. You can narrow your search to a specific location and distance by adding the `center` parameter (with latitude and longitude) and an optional `distance` parameter (in meters):	Name
`placetopic`	Returns a list of possible place Page topics and their IDs. Use with `topic_filter=all` parameter to get the full list.	None
`ad_*`	A collection of different search options that can be used to find targeting options.	See Targeting Options

Example:

```
GET graph.facebook.com
  /search?
    q=coffee&
    type=place&
    center=37.76,-122.427&
    distance=1000
```

Handling Errors

Requests made to our APIs can result in a number of different error responses. The following topic describes the recovery tactics and provides a list of error values with a map to the most common recovery tactic to use.

Receiving Error Codes

The following represents a common error response resulting from a failed API request:

```
{
        "error": {
                "message": "Message describing the error",
                "type": "OAuthException",
                "code": 190,
                "error_subcode": 460,
                "error_user_title": "A title",
                "error_user_msg": "A message",
                "fbtrace_id": "EJplcsCHuLu"
        }
}
```

- `message`: A human-readable description of the error.
- `code`: An error code. Common values are listed below, along with common recovery tactics.
- `error_subcode`: Additional information about the error. Common values are listed below.
- `error_user_msg`: The message to display to the user. The language of the message is based on the locale of the API request.
- `error_user_title`: The title of the dialog, if shown. The language of the message is based on the locale of the API request.
- `fbtrace_id`: Internal support identifier. When reporting a bug related to a Graph API call, include the `fbtrace_id` to help us find log data for debugging.

Error Codes

Code or Type	Name	What To Do
OAuthException		If no subcode is present, this means that the login status or access token has expired, been revoked, or is otherwise invalid. Get a new access token. If a subcode is present, see the subcode.
102	API Session	If no subcode is present, this means that the login status or access token has expired, been revoked, or is otherwise invalid. Get a new access token. If a subcode is present, see the subcode.
1	API Unknown	Possibly a temporary issue due to downtime. Wait and retry the operation. If it occurs again, check you are requesting an existing API.
2	API Service	Temporary issue due to downtime. Wait and retry the operation.
4	API Too Many Calls	Temporary issue due to throttling. Wait and retry the operation, or examine your API request volume.
17	API User Too Many Calls	Temporary issue due to throttling. Wait and retry the operation, or examine your API request volume.

Code or Type	Name	What To Do
10	API Permission Denied	Permission is either not granted or has been removed. Handle the missing permissions.
190	Access token has expired	Get a new access token.
200-299	API Permission (Multiple values depending on permission)	Permission is either not granted or has been removed. Handle the missing permissions.
341	Application limit reached	Temporary issue due to downtime or throttling. Wait and retry the operation, or examine your API request volume.
368	Temporarily blocked for policies violations	Wait and retry the operation.
506	Duplicate Post	Duplicate posts cannot be published consecutively. Change the content of the post and try again.
1609005	Error Posting Link	There was a problem scraping data from the provided link. Check the URL and try again.

Authentication Error Subcodes

Code	Name	What To Do
458	App Not Installed	The user has not logged into your app. Reauthenticate the user.
459	User Checkpointed	The user needs to log in at https://www.facebook.com or https://m.facebook.com to correct an issue.
460	Password Changed	On iOS 6 and above, if the person logged in using the OS-integrated flow, they should be directed to Facebook OS settings on the device to update their password. Otherwise, they must login to the app again.
463	Expired	Login status or access token has expired, been revoked, or is otherwise invalid. Handle expired access tokens.
464	Unconfirmed User	The user needs to log in at https://www.facebook.com or https://m.facebook.com to correct an issue.
467	Invalid access token	Access token has expired, been revoked, or is otherwise invalid. Handle expired access tokens.

Debugging API Requests

Graph API Debug Mode

When Debug Mode is enabled, Graph API response may contain additional fields that explain potential issues with the request.

To enable debug mode, use the `debug` query string parameter. For example:

```
GET graph.facebook.com
  /v2.3/me/friends
    access_token=...&
    debug=all
```

If `user_friends` permission was not granted, this will produce the following response:

```
{
  "data": [
  ],
  "__debug__": {
    "messages": [
      {
        "message": "Field friends is only accessible on User object, if user_friends permission is granted
        "type": "warning"
      },
      {
        "link": "https://developers.facebook.com/docs/apps/changelog#v2_0",
        "message": "Only friends who have installed the app are returned in versions greater or equal to v2
        "type": "info"
      }
    ]
  }
}
```

The `debug` parameter value can be set to "all" or to a minimal requested severity level that corresponds to `type` of the message:

Debug Param Value	What Will Be Returned
all	All available debug messages.
info	Debug messages with type *info* and *warning*.
warning	Only debug messages with type *warning*.

Debug information, when available, is returned as a JSON object under the `__debug__` key in the `messages` array. Every element of this array is a JSON object that contains the following fields:

Field	Datatype	Description
message	String	The message.
type	String	The message severity.
link	String	[Optional] A URL pointing to related information.

You can also use Debug Mode with Graph API Explorer.

Determining Version used by API Requests

When you're building an app and making Graph API requests, you might find it useful to be able to determine what version of the API you're getting a response from. For example, if you're making calls without a version specified, the API version that responds may not be known to you.

The Graph API supplies a request header with any response called `facebook-api-version` that indicates the exact version of the API that generated the response. For example, a Graph API call that generates a request with v2.0 will have the following HTTP header:

```
facebook-api-version:v2.0
```

This `facebook-api-version` header allows you to determine whether API calls are being returned from the version that you expect.

Debug Info for Reporting Bugs

If you need to report a bug in the Graph API, we include some additional request headers that you should send with your bug report to help us pinpoint and reproduce your issue. These request headers are `X-FB-Debug`, `x-fb-rev`, and `X-FB-Trace-ID`.


Give Feedback
pañol
Português (Brasil)
facebook for developers
Log In