
A Facebook Graph API SDK In Golang



This is a Go package that fully supports the Facebook Graph API with file upload, batch request and marketing API. It can be used in Google App Engine.

API documentation can be found on godoc.

Feel free to create an issue or send me a pull request if you have any “how-to” question or bug or suggestion when using this package. I’ll try my best to reply to it.

Install

If `go mod` is enabled, install this package with `go get github.com/huandu/facebook/v2`. If not, call `go get -u github.com/huandu/facebook` to get the latest master branch version.

Note that, since go1.14, incompatible versions are omitted unless specified explicitly. Therefore, it’s highly recommended to upgrade the import path to `github.com/huandu/facebook/v2` when possible to avoid any potential dependency error.

Usage

Quick start

Here is a sample that reads my Facebook first name by uid.

```
1 package main
2
3 import (
4     "fmt"
5     fb "github.com/huandu/facebook/v2"
6 )
7
8 func main() {
9     res, _ := fb.Get("/538744468", fb.Params{
10         "fields": "first_name",
11         "access_token": "a-valid-access-token",
12     })
13     fmt.Println("Here is my Facebook first name:", res["first_name"])
14 }
```

The type of `res` is `fb.Result` (a.k.a. `map[string]interface{}`). This type has several useful methods to decode `res` to any Go type safely.

```

1 // Decode "first_name" to a Go string.
2 var first_name string
3 res.DecodeField("first_name", &first_name)
4 fmt.Println("Here's an alternative way to get first_name:", first_name)
5
6 // It's also possible to decode the whole result into a predefined
  struct.
7 type User struct {
8     FirstName string
9 }
10
11 var user User
12 res.Decode(&user)
13 fmt.Println("print first_name in struct:", user.FirstName)

```

If a type implements the `json.Unmarshaler` interface, `Decode` or `DecodeField` will use it to unmarshal JSON.

```

1 res := Result{
2     "create_time": "2006-01-02T15:16:17Z",
3 }
4
5 // Type `*time.Time` implements `json.Unmarshaler`.
6 // res.DecodeField will use the interface to unmarshal data.
7 var tm time.Time
8 res.DecodeField("create_time", &tm)

```

Read a graph user object with a valid access token

```

1 res, err := fb.Get("/me/feed", fb.Params{
2     "access_token": "a-valid-access-token",
3 })
4
5 if err != nil {
6     // err can be a Facebook API error.
7     // if so, the Error struct contains error details.
8     if e, ok := err.(*Error); ok {
9         fmt.Printf("facebook error. [message:%v] [type:%v] [code:%v] [
10             subcode:%v] [trace:%v]",
11             e.Message, e.Type, e.Code, e.ErrorSubcode, e.TraceID)
12         return
13     }
14     // err can be an unmarshal error when Facebook API returns a
15     // message which is not JSON.
16     if e, ok := err.(*UnmarshalError); ok {
17         fmt.Printf("facebook error. [message:%v] [err:%v] [payload:%v]"
18             ,

```

```

17         e.Message, e.Err, string(e.Payload))
18     }
19     return
20 }
21 return
22 }
23
24 // read my last feed story.
25 fmt.Println("My latest feed story is:", res.Get("data.0.story"))

```

Read a graph search for page and decode slice of maps

```

1 res, _ := fb.Get("/pages/search", fb.Params{
2     "access_token": "a-valid-access-token",
3     "q":            "nightlife,singapore",
4 })
5
6 var items []fb.Result
7
8 err := res.DecodeField("data", &items)
9
10 if err != nil {
11     fmt.Printf("An error has happened %v", err)
12     return
13 }
14
15 for _, item := range items {
16     fmt.Println(item["id"])
17 }

```

Use App and Session

It's recommended to use [App](#) and [Session](#) in a production app. They provide more control over all API calls. They can also make code clearer and more concise.

```

1 // Create a global App var to hold app id and secret.
2 var globalApp = fb.New("your-app-id", "your-app-secret")
3
4 // Facebook asks for a valid redirect URI when parsing the signed
5 // request.
6 // It's a newly enforced policy starting as of late 2013.
7 globalApp.RedirectUri = "http://your.site/canvas/url/"
8
9 // Here comes a client with a Facebook signed request string in the
10 // query string.
11 // This will return a new session from a signed request.
12 session, _ := globalApp.SessionFromSignedRequest(signedRequest)

```

```

11
12 // If there is another way to get decoded access token,
13 // this will return a session created directly from the token.
14 session := globalApp.Session(token)
15
16 // This validates the access token by ensuring that the current user ID
17 // is properly returned. err is nil if the token is valid.
18 err := session.Validate()
19
20 // Use the new session to send an API request with the access token.
21 res, _ := session.Get("/me/feed", nil)

```

By default, all requests are sent to Facebook servers. If you wish to override the API base URL for unit-testing purposes - just set the respective `Session` field.

```

1 testSrv := httptest.NewServer(someMux)
2 session.BaseURL = testSrv.URL + "/"

```

Facebook returns most timestamps in an ISO9601 format which can't be natively parsed by Go's `encoding/json`. Setting `RFC3339Timestamps true` on the `Session` or at the global level will cause proper RFC3339 timestamps to be requested from Facebook. RFC3339 is what `encoding/json` natively expects.

```

1 fb.RFC3339Timestamps = true
2 session.RFC3339Timestamps = true

```

Setting either of these to true will cause `date_format=Y-m-d\TH:i:sP` to be sent as a parameter on every request. The format string is a PHP `date()` representation of RFC3339. More info is available in this issue.

Use paging field in response

Some Graph API responses use a special JSON structure to provide paging information. Use `Result.Paging()` to walk through all data in such results.

```

1 res, _ := session.Get("/me/home", nil)
2
3 // create a paging structure.
4 paging, _ := res.Paging(session)
5
6 var allResults []Result
7
8 // append first page of results to slice of Result
9 allResults = append(allResults, paging.Data()...)
10
11 for {
12     // get next page.

```

```

13  noMore, err := paging.Next()
14  if err != nil {
15      panic(err)
16  }
17  if noMore {
18      // No more results available
19      break
20  }
21  // append current page of results to slice of Result
22  allResults = append(allResults, paging.Data()...)
23  }

```

Read Graph API response and decode result in a struct

The Facebook Graph API always uses snake case keys in API response. This package can automatically convert from snake case to Go's camel-case-style struct field names.

For instance, to decode the following JSON response...

```

1  {
2    "foo_bar": "player"
3  }

```

One can use the following struct.

```

1  type Data struct {
2      FooBar string // "FooBar" maps to "foo_bar" in JSON automatically
3  }

```

The decoding of each struct field can be customized by the format string stored under the `facebook` key or the `"json"` key in the struct field's tag. The `facebook` key is recommended as it's specifically designed for this package.

Following is a sample that shows all possible field tags.

```

1  // define a Facebook feed object.
2  type FacebookFeed struct {
3      Id          string          `facebook:",required"` //
4      //          this field must exist in response.
5  }

```

mind
the
", "

```

5     Story      string
6     FeedFrom   *FacebookFeedFrom `facebook:"from"`           //
           use customized field name "from".
7     CreatedTime string           `facebook:"created_time,required"` //
           both customized field name and "required" flag.
8     Omitted    string           `facebook:"-"`             //
           this field is omitted when decoding.
9 }
10
11 type FacebookFeedFrom struct {
12     Name string `json:"name"`           // the "json" key also
           works as expected.
13     Id string  `facebook:"id" json:"shadowed"` // if both "facebook"
           and "json" key are set, the "facebook" key is used.
14 }
15
16 // create a feed object direct from Graph API result.
17 var feed FacebookFeed
18 res, _ := session.Get("/me/feed", nil)
19 res.DecodeField("data.0", &feed) // read latest feed

```

before
 "
 required
 ".

Send a batch request

```

1  params1 := Params{
2      "method": fb.GET,
3      "relative_url": "me",
4  }
5  params2 := Params{
6      "method": fb.GET,
7      "relative_url": uint64(100002828925788),
8  }
9  results, err := fb.BatchApi(your_access_token, params1, params2)
10
11 if err != nil {
12     // check error...
13     return
14 }
15
16 // batchResult1 and batchResult2 are response for params1 and params2.
17 batchResult1, _ := results[0].Batch()
18 batchResult2, _ := results[1].Batch()
19
20 // Use parsed result.

```

```
21 var id string
22 res := batchResult1.Result
23 res.DecodeField("id", &id)
24
25 // Use response header.
26 contentType := batchResult1.Header.Get("Content-Type")
```

Using with Google App Engine

Google App Engine provides the `appengine/urlfetch` package as the standard HTTP client package. For this reason, the default client in `net/http` won't work. One must explicitly set the HTTP client in `Session` to make it work.

```
1 import (
2     "appengine"
3     "appengine/urlfetch"
4 )
5
6 // suppose it's the AppEngine context initialized somewhere.
7 var context appengine.Context
8
9 // default Session object uses http.DefaultClient which is not allowed
10 // to use
11 // in appengine. one has to create a Session and assign it a special
12 // client.
13 session := globalApp.Session("a-access-token")
14 session.HttpClient = urlfetch.Client(context)
15
16 // now, the session uses AppEngine HTTP client now.
17 res, err := session.Get("/me", nil)
```

Select Graph API version

See Platform Versioning to understand the Facebook versioning strategy.

```
1 // This package uses the default version which is controlled by the
2 // Facebook app setting.
3 // change following global variable to specify a global default version
4 .
5 fb.Version = "v3.0"
6
7 // starting with Graph API v2.0; it's not allowed to get useful
8 // information without an access token.
9 fb.Api("huan.du", GET, nil)
10
11 // it's possible to specify version per session.
```

```
9 session := &fb.Session{}
10 session.Version = "v3.0" // overwrite global default.
```

Enable appsecret_proof

Facebook can verify Graph API Calls with `appsecret_proof`. It's a feature to make Graph API call more secure. See [Securing Graph API Requests](#) to know more about it.

```
1 globalApp := fb.New("your-app-id", "your-app-secret")
2
3 // enable "appsecret_proof" for all sessions created by this app.
4 globalApp.EnableAppsecretProof = true
5
6 // all calls in this session are secured.
7 session := globalApp.Session("a-valid-access-token")
8 session.Get("/me", nil)
9
10 // it's also possible to enable/disable this feature per session.
11 session.EnableAppsecretProof(false)
```

Debugging API Requests

Facebook has introduced a way to debug Graph API calls. See [Debugging API Requests](#) for more details.

This package provides both a package level and per session debug flag. Set `Debug` to a `DEBUG_*` constant to change debug mode globally, or use `Session#SetDebug` to change debug mode for one session.

When debug mode is turned on, use `Result#DebugInfo` to get `DebugInfo` struct from the result.

```
1 fb.Debug = fb.DEBUG_ALL
2
3 res, _ := fb.Get("/me", fb.Params{"access_token": "xxx"})
4 debugInfo := res.DebugInfo()
5
6 fmt.Println("http headers:", debugInfo.Header)
7 fmt.Println("facebook api version:", debugInfo.FacebookApiVersion)
```

Monitoring API usage info

Call `Result#UsageInfo` to get a `UsageInfo` struct containing both app and page-level rate limit information from the result. More information about rate limiting can be found [here](#).

```
1 res, _ := fb.Get("/me", fb.Params{"access_token": "xxx"})
2 usageInfo := res.UsageInfo()
3
4 fmt.Println("App level rate limit information:", usageInfo.App)
5 fmt.Println("Page level rate limit information:", usageInfo.Page)
6 fmt.Println("Ad account rate limiting information:", usageInfo.
    AdAccount)
7 fmt.Println("Business use case usage information:", usageInfo.
    BusinessUseCase)
```

Work with package `golang.org/x/oauth2`

The `golang.org/x/oauth2` package can handle the Facebook OAuth2 authentication process and access token quite well. This package can work with it by setting `Session#HttpClient` to OAuth2's client.

```
1 import (
2     "golang.org/x/oauth2"
3     oauth2fb "golang.org/x/oauth2/facebook"
4     fb "github.com/huandu/facebook/v2"
5 )
6
7 // Get Facebook access token.
8 conf := &oauth2.Config{
9     ClientID:     "AppId",
10    ClientSecret:  "AppSecret",
11    RedirectURL:   "CallbackURL",
12    Scopes:        []string{"email"},
13    Endpoint:      oauth2fb.Endpoint,
14 }
15 token, err := conf.Exchange(oauth2.NoContext, "code")
16
17 // Create a client to manage access token life cycle.
18 client := conf.Client(oauth2.NoContext, token)
19
20 // Use OAuth2 client with session.
21 session := &fb.Session{
22     Version:      "v2.4",
23     HttpClient:   client,
24 }
25
26 // Use session.
27 res, _ := session.Get("/me", nil)
```

Control timeout and cancelation with Context

The `Session` accept a `Context`.

```
1 // Create a new context.
2 ctx, cancel := context.WithTimeout(session.Context(), 100 * time.
    Millisecond)
3 defer cancel()
4
5 // Call an API with ctx.
6 // The return value of `session.WithContext` is a shadow copy of
    original session and
7 // should not be stored. It can be used only once.
8 result, err := session.WithContext(ctx).Get("/me", nil)
```

See this Go blog post about context for more details about how to use `Context`.

Change Log

See CHANGELOG.md.

Out of Scope

1. No OAuth integration. This package only provides APIs to parse/verify access token and code generated in OAuth 2.0 authentication process.
2. No old RESTful API and FQL support. Such APIs are deprecated for years. Forget about them.

License

This package is licensed under the MIT license. See LICENSE for details.