
LINE Messaging API SDK for Go



Introduction

The LINE Messaging API SDK for Go makes it easy to develop bots using LINE Messaging API, and you can create a sample bot within minutes.

Documentation

See the official API documentation for more information.

- English: <https://developers.line.biz/en/docs/messaging-api/overview/>
- Japanese: <https://developers.line.biz/ja/docs/messaging-api/overview/>

Requirements

This library requires Go 1.20 or later.

Installation

```
1 $ go get -u github.com/line/line-bot-sdk-go/v8/linebot
```

Import all packages in your code

```
1 import (  
2     "github.com/line/line-bot-sdk-go/v8/linebot"  
3     "github.com/line/line-bot-sdk-go/v8/linebot/channel_access_token"  
4     "github.com/line/line-bot-sdk-go/v8/linebot/insight"  
5     "github.com/line/line-bot-sdk-go/v8/linebot/liff"  
6     "github.com/line/line-bot-sdk-go/v8/linebot/manage_audience"  
7     "github.com/line/line-bot-sdk-go/v8/linebot/messaging_api"  
8     "github.com/line/line-bot-sdk-go/v8/linebot/module"  
9     "github.com/line/line-bot-sdk-go/v8/linebot/module_attach"  
10    "github.com/line/line-bot-sdk-go/v8/linebot/shop"  
11    "github.com/line/line-bot-sdk-go/v8/linebot/webhook"  
12 )
```

Configuration

```
1 import (
2     "github.com/line/line-bot-sdk-go/v8/linebot/messaging_api"
3 )
4
5 func main() {
6     bot, err := messaging_api.NewMessagingApiAPI(
7         os.Getenv("LINE_CHANNEL_TOKEN"),
8     )
9     ...
10 }
```

Configuration with `http.Client`

Every client application allows configuration with `WithHTTPClient` and `WithEndpoint`. (For Blob client, configurations `WithBlobHTTPClient` and `WithBlobEndpoint` are also available.)

```
1 client := &http.Client{}
2 bot, err := messaging_api.NewMessagingApiAPI(
3     os.Getenv("LINE_CHANNEL_TOKEN"),
4     messaging_api.WithHTTPClient(client),
5 )
6 ...
```

Getting Started

The LINE Messaging API primarily utilizes the JSON data format. To parse the incoming HTTP requests, the `webhook.ParseRequest()` method is provided. This method reads the `*http.Request` content and returns a slice of pointers to Event Objects.

```
1 import (
2     "github.com/line/line-bot-sdk-go/v8/linebot/webhook"
3 )
4
5 cb, err := webhook.ParseRequest(os.Getenv("LINE_CHANNEL_SECRET"), req)
6 if err != nil {
7     // Handle any errors that occur.
8 }
```

The LINE Messaging API is capable of handling various event types. The Messaging API SDK automatically unmarshals these events into respective classes like `webhook.MessageEvent`, `webhook.FollowEvent`, and so on. You can easily check the type of the event and respond accordingly using a switch statement as shown below:

```

1  for _, event := range cb.Events {
2      switch e := event.(type) {
3          case webhook.MessageEvent:
4              // Do Something...
5          case webhook.StickerMessageContent:
6              // Do Something...
7      }
8  }

```

We provide code examples. - EchoBot - a simple echo bot - KitchenSink - a bot that handles many types of events - EchoBotHandler - A simple bot that automatically verifies signatures and only handles Webhook events - DeliveryHelper - InsightHelper - RichmenuHelper

Receiver

To send a message to a user, group, or room, you need either an ID

```

1  userID := event.Source.UserId
2  groupID := event.Source.GroupId
3  RoomID := event.Source.RoomId

```

or a reply token.

```

1  replyToken := event.ReplyToken

```

Create message

The LINE Messaging API provides various types of message.

```

1  bot.ReplyMessage(
2      &messaging_api.ReplyMessageRequest{
3          ReplyToken: e.ReplyToken,
4          Messages: []messaging_api.MessageInterface{
5              messaging_api.TextMessage{
6                  Text: replyMessage,
7              },
8          },
9      },
10 )

```

Send message

With an ID, you can send message using `PushMessage()`

```

1 bot.PushMessage(
2     &messaging_api.PushMessageRequest{
3         To: "U.....",
4         Messages: []messaging_api.MessageInterface{
5             messaging_api.TextMessage{
6                 Text: replyMessage,
7             },
8         },
9     },
10     nil, // x-line-retry-key
11 )

```

With a reply token, you can reply to messages using `ReplyMessage()`

```

1 bot.ReplyMessage(
2     &messaging_api.ReplyMessageRequest{
3         ReplyToken: e.ReplyToken,
4         Messages: []messaging_api.MessageInterface{
5             messaging_api.TextMessage{
6                 Text: replyMessage,
7             },
8         },
9     },
10 )

```

How to get response header and error message

You may need to store the `x-line-request-id` header obtained as a response from several APIs. In this case, please use `~WithHttpInfo`. You can get headers and status codes. The `x-line-accepted-request-id` or `content-type` header can also be obtained in the same way.

```

1 resp, _, _ := app.bot.ReplyMessageWithHttpInfo(
2     &messaging_api.ReplyMessageRequest{
3         ReplyToken: replyToken,
4         Messages: []messaging_api.MessageInterface{
5             messaging_api.TextMessage{
6                 Text: "Hello, world",
7             },
8         },
9     },
10 )
11 log.Printf("status code: (%v), x-line-request-id: (%v)", resp.
12     StatusCode, resp.Header.Get("x-line-request-id"))

```

Similarly, you can get specific error messages by using `~WithHttpInfo`.

```

1 resp, _, err := app.bot.ReplyMessageWithHttpInfo(
2     &messaging_api.ReplyMessageRequest{

```

```

3     ReplyToken: replyToken + "invalid",
4     Messages: []messaging_api.MessageInterface{
5         messaging_api.TextMessage{
6             Text: "Hello, world",
7         },
8     },
9 },
10 )
11 if err != nil && resp.StatusCode >= 400 && resp.StatusCode < 500 {
12     decoder := json.NewDecoder(resp.Body)
13     errorResponse := &messaging_api.ErrorResponse{}
14     if err := decoder.Decode(&errorResponse); err != nil {
15         log.Fatal("failed to decode JSON: %w", err)
16     }
17     log.Printf("status code: (%v), x-line-request-id: (%v), error
        response: (%v)", resp.StatusCode, resp.Header.Get("x-line-
        request-id"), errorResponse)
18 }

```

Help and media

FAQ: <https://developers.line.biz/en/faq/>

News: <https://developers.line.biz/en/news/>

Versioning

This project respects semantic versioning.

See <http://semver.org/>

Contributing

Please check CONTRIBUTING before making a contribution.

License

```

1 Copyright (C) 2016 LINE Corp.
2
3 Licensed under the Apache License, Version 2.0 (the "License");
4 you may not use this file except in compliance with the License.
5 You may obtain a copy of the License at
6
7     http://www.apache.org/licenses/LICENSE-2.0

```

8
9 Unless required by applicable law or agreed to in writing, software
10 distributed under the License is distributed on an "AS IS" BASIS,
11 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied
12 .
13 See the License **for** the specific language governing permissions and
14 limitations under the License.