Using The API

Author: Ian Wang

The BaseWebHookHandler

The BaseWebHookHandler is the center of the API. It takes incoming events and parses the raw data into various classes so that their properties can be easily accessed.

The WebHookHandler is constructed with 3 properties, the **channelSecret**, **channelAccessToken** and a **MessageSender**. The channelSecret and channelAccessToken properties are just the string values taken from the LINE@ channel page.

The MessageSender

A MessageSender is an interface for sending push/reply messages. The reason the interface is separate from the WebHookHandler is to allow custom implementations of message sending. For example, you might want to push all message requests to a central server before sending it to the LINE Messaging API, or perhaps you want to log each sent message. Using an interface allows the user to customize the functionality of the message sending functions to acheive different user goals. The interface itself only has four methods:

```
public interface MessageSender {

    public Response sendReply(String token, List<Message> replyMessages, String
metadata);
    public Response sendPush(String userId, List<Message> pushMessages, String
metadata);
    public Response linkRichMenu(String richMenuId, String userId);
    public Response UnlinkRichMenu(String userId);

}
```

There is a default **HTTPMessageSender** class that is included in the project which directly sends the messages it receives to the LINE Messaging API. This class utilizes the *httpclient* library, which is included as a dependency for the project setup.

Inheriting the BaseWebHookHandler

With the API, custom bots are created via inheritance to the BaseWebHookHandler class. There is a central event handler function which receives incoming events of all different types, and can be overriden to customize the functionality of the bot. Bellow is a sample implementation of a custom handler function which echos text messages from LINE users that message the bot:

```
@Override
protected void handleEvent(String userId, WebHookEvent event) {
    if (event.type() == WebHookEventType.MESSAGE) {
        MessageEvent messageEvent = (MessageEvent) event;
        if (messageEvent.message().type() == MessageType.TEXT) {
            TextMessage textMessage = (TextMessage) messageEvent.message();
            // Just send an echo reply
            Util.sendSingleTextPush(messageSender, userId, textMessage.getText());
        }
    }
}
```

Setting up the BaseWebHookHandler

The LINE Messaging API sends its events to a WebHook, which is usually a server that accepts HTTPS POST requests. The API does not have a built in HTTPS server, but instead has specifications on how it should take in data from HTTPS POST requests. Every interaction goes through the **handleWebHookEvent** function, and the function signature looks like so:

```
public final boolean handleWebHookEvent(String headerSignature, String body);
```

The **headerSignature** is a request header with the key **X-Line-Signature**, and it is used to verify the message source. The validation is taken care of by the WebHookHandler and does not need to be done by the user of the API. The body should be a big chunk of JSON data passed into the function as a String.

Thus, to hook up the WebHookHandler to your bot server, simply call the central function from your POST request receiver with the corresponding fields set properly.

An example is included in the project under the *APIdemo* package, and is implemented using the **httpserver** library.

Dialogue Stacks

A more streamlined method of developing bot applications, called **dialogue stacks**, are implemented in the API as well. A *Dialogue* defines how the bot server handles the incoming events of a certain user, and each user is allocated their own dialogue stack. More information can be found in the dialogue stack documentation.

Documentation

Most classes and methods in the API have docs written for them, and more complicated code have comments written alongside as well.

There is a demo written in the project as well, and it uses most functionalities of the API; Thus, it can server as a useful reference.