FXCM Quick Start-Up guide
REST API – JAVA

1. Introduction

Application programming interface, or the APIs, have become increasingly popular with the rise of automated trading strategies. Many retail brokers now provide APIs that enable traders to directly connect their screening software with their brokerage account to share real time prices and to place orders.

FXCM provides multiple APIs which you can learn more about if you click on this link https://www.fxcm.com/uk/algorithmic-trading/. One of these APIs is called REST API and is the one commonly used among the algorithmic traders.

FXCM provides a REST API to interact with its trading platform. Among others, it allows the retrieval of historical data as well as streaming data. In addition, it allows to place different types of orders and to read out account information. The overall goal is to allow the implementation of automated, algorithmic trading programs.

In the following paragraphs, you learn how to use Java and Maven in order to connect to the API and use the API for different trading purposes.

2. Prerequisites

2.1 **Demo account.** You can open such an account by following the step on the page below. https://www.fxcm.com/uk/algorithmic-trading/api-trading/.

2.2 **Spring tool suite.** You can download it from here https://spring.io/tools and create a simple Maven project.

2.3. **Install the necessary dependencies** and then reference them in your code. The list of libraries is the following:

- Socket.io library https://github.com/fxcm/fxcm-api-rest-nodejs-example.

Maven

```
<!-- https://mvnrepository.com/artifact/io.socket/socket.io-client -->
<dependency>
    <groupId>io.socket</groupId>
    <artifactId>socket.io-client</artifactId>
    <version>1.0.0</version>
</dependency>
```

Gradle

```
// https://mvnrepository.com/artifact/io.socket/socket.io-client
@Grapes(
    @Grab(group='io.socket', module='socket.io-client', version='1.0.0')
)
```

- Apache Commons IO 2.6 https://mvnrepository.com/artifact/commons-io/commons-io/2.6

Maven

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
</dependency>
```

Gradle

```
compile group: 'commons-io', name: 'commons-io', version: '2.6'
```

3. First Steps

After complete all the steps from Stage 2, it is time to create a simple class and connect to the server.

Firstly, declare all the parameters we need in order to establish the socket connection. Do not forget to put your account id and access token in the fields specified.

```
public class App {
    static String trading_api_host_demo = "api-demo.fxcm.com";
    static String trading_api_port = "443";
    static String demo_connection = "https://" + trading_api_host_demo + ":"+ trading_api_port
    static String login_token = "YOUR_API_ACCESS_TOKEN_GOES_HERE";
    static Socket server_socket;
    static String bearer_access_token;
    static String charset = java.nio.charset.StandardCharsets.UTF_8.name();
    static String accountID="YOUR_ACCOUNT_ID_GOES_HERE";
```

Once this is completed, create a main method and put the following inside

```
public static void main(String[] args) throws URISyntaxException {
        IO.Options options = new IO.Options();
        options.forceNew = true;

        final OkHttpClient client = new OkHttpClient();
        options.webSocketFactory = client;
        options.callFactory = client;
        options.query = "access_token="+ login_token;
```

```
        //final Socket socket = IO.socket(url + ":" + port, options);
        server_socket = IO.socket(demo_connection, options);
        server_socket.on(Socket.EVENT_CONNECT, new Emitter.Listener() {
            @Override
            public void call(Object... args) {
                bearer_access_token = "Bearer" + server_socket.id() + login_token;
                System.out.println("connected, Server Socket ID: " + server_socket.id());
                System.out.println("bearer token: " + bearer_access_token);

            }
        });
        server_socket.io().on(io.socket.engineio.client.Socket.EVENT_CLOSE, new Emitter.Listener()
{
            @Override
            public void call(Object... args) {
                System.out.println("engine close");
                client.dispatcher().executorService().shutdown();
            }
        });
        server_socket.open();
    }
```

After running the code above as a Java Application, the result should be:

```
Connected, Server Socket ID: eP6m66M0KtHdNxZhAAZy
bearer token: Bearer eP6m66M0KtHdNxZhAAZy63568b9f0f3ac589f862ccf3ce679e5cf62ef43b
engine close
```

Once we established connection with the server, we can make get or post request to the database. Below I am going to outline an example for each type.

4. Using the REST API JAVA Client

In order to make a request to the database we need to complete three steps. The first one is to create a method which executes a get/post request. The second one is to create a method where the proper parameters are added to the get/post request and the third one is to call that method right after the connection is established.

4.1. Retrieve Historical Prices

sendGetRequest()

```
public static String sendGetRequest(String path, String param) throws Exception {
            final String path2 = demo_connection + path +"?" + param;
            final URL url = new URL(path2);

            final HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            // Set the appropriate header fields in the request header.
            connection.setRequestMethod("GET");

            connection.setRequestProperty("Authorization", bearer_access_token);
            connection.setRequestProperty("Accept", "application/json");
            connection.setRequestProperty("host", trading_api_host_demo);
            connection.setRequestProperty("port", trading_api_port);
            connection.setRequestProperty("path", path +"/?"+ param);
```

```
            connection.setRequestProperty("User-Agent", "request");
            connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded'");

            final String response = getResponseString(connection);
            final int responseCode = connection.getResponseCode();

            System.out.println("\nSending 'GET' request: " + path2);
            System.out.println("Response Code : " + responseCode);

            if (responseCode == 200) {
                  return response;
            } else {
                  throw new Exception(response);
            }
      }
```

getHistoricalData()

```
static void getHistoricalData() throws Exception
      {
            String priceList = sendGetRequest("/candles/1/m1", "num=5");
            System.out.println(priceList);

      }
```

Main method

```
public static void main(String[] args) throws URISyntaxException {

• • •
        server_socket.on(Socket.EVENT_CONNECT, new Emitter.Listener() {
            @Override
            public void call(Object... args) {
                bearer_access_token = "Bearer" + server_socket.id() + login_token;
                System.out.println("connected, Server Socket ID: " + server_socket.id());
                System.out.println("bearer token: " + bearer_access_token);

                getHistoricalData();

            }
        });
• • •
```

Result

```
connected, Server Socket ID: PRGXu7KbFuiIIhVFAAfA
bearer token: Bearer PRGXu7KbFuiIIhVFAAfA63568b9f0f3ac589f862ccf3ce679e5cf62ef43b

Sending 'GET' request: https://api-demo.fxcm.com:443/candles/1/m1?num=5
Response Code : 200
{"response":{"executed":true,"error":""},"instrument_id":"1","period_id":"m1","candles":[[15353738
40,1.16341,1.1636,1.16366,1.1634,1.16352,1.16373,1.16377,1.16352,404],[1535373900,1.1636,1.16361,1
.16365,1.16345,1.16373,1.16374,1.16375,1.16358,331],[1535373960,1.1636,1.1638,1.16381,1.1636,1.163
73,1.16393,1.16393,1.16373,304],[1535374020,1.1638,1.1638,1.16392,1.1638,1.16393,1.16393,1.16406,1
.1639,356],[1535374080,1.16381,1.16362,1.16393,1.16359,1.16394,1.16374,1.16406,1.1637,256]]}

engine closed
```

4.2.Send Market Order

sendPostRequest()

```
public static String sendPostRequest(String path, String param)
	{
		URL url;
		HttpURLConnection connection = null;
		try {
			//Create connection
			url = new URL(demo_connection + path);
			connection = (HttpURLConnection)url.openConnection();
			connection.setRequestMethod("POST");
			connection.setRequestProperty("Content-Type", "application/x-www-form-
urlencoded");

			connection.setRequestProperty("Content-Length", "" +
Integer.toString(param.getBytes().length));
			connection.setRequestProperty("Content-Language", "en-US");

			connection.setRequestProperty("Authorization", bearer_access_token);
			connection.setRequestProperty("Accept", "application/json");
			connection.setRequestProperty("host", trading_api_host_demo);
			connection.setRequestProperty("port", trading_api_port);
			connection.setRequestProperty("path", path);
			connection.setRequestProperty("User-Agent", "request");


			connection.setUseCaches (false);
			connection.setDoInput(true);
			connection.setDoOutput(true);

			//Send request
			DataOutputStream wr = new DataOutputStream (connection.getOutputStream ());
			wr.writeBytes (param);
			wr.flush ();
			wr.close ();

			//Get Response
			final String response = getResponseString(connection);

			System.out.println("\nSending 'POST' request: " + path);
			System.out.println("Response : " + response);

			return response.toString();

		} catch (Exception e) {

			e.printStackTrace();
			return null;

		} finally {

			if(connection != null) {
				connection.disconnect();
			}
		}
	}
```

getResponseString()

```
private static String getResponseString(HttpURLConnection conn) throws IOException {

		try (BufferedReader reader = new BufferedReader(new
InputStreamReader(conn.getInputStream(), StandardCharsets.UTF_8)))
		{
			final StringBuilder stringBuffer = new StringBuilder();
			String line = "";
			while ((line = reader.readLine()) != null) {
				stringBuffer.append(line);
				stringBuffer.append('\r');
			}
			reader.close();
			return stringBuffer.toString();
		}
	}
```

sendMarketOrder()

```
static void sendMarketOrder() throws Exception
	{
		if (accountID == "")
		{
			System.out.println("Account ID missing");
		}
		else
		{
			String cmd = "account_id=" + accountID +
					"&symbol=EUR%2FUSD" +
					"&is_buy=true" +
					"&rate=0" +

					"&amount=2" +
					"&order_type=AtMarket" +
					"&time_in_force=GTC";

			sendPostRequest("/trading/open_trade", cmd);
		}
	}
```

Main method

```
public static void main(String[] args) throws URISyntaxException {
...
        server_socket.on(Socket.EVENT_CONNECT, new Emitter.Listener() {
            @Override
            public void call(Object... args) {
                bearer_access_token = "Bearer" + server_socket.id() + login_token;
                System.out.println("connected, Server Socket ID: " + server_socket.id());
                System.out.println("bearer token: " + bearer_access_token);

                sendMarketOrder();

            }
        });
...
```

Result

```
connected, Server Socket ID: WXEDw-xk51GIbRDGAAfI

bearer token: Bearer WXEDw-xk51GIbRDGAAfI63568b9f0f3ac589f862ccf3ce679e5cf62ef43b

Sending 'POST' request: /trading/open_trade

Response : {"response":{"executed":false,"error":"{\"code\":3,\"message\":\"Amount should be
divisible by 100\",\"parameters\":[\"100\"]}"},"data":{}}

engine closed
```

5. Conclusion

This is the REST API JAVA Client which makes one GET and one POST HTTP Requests.

When constructing your HTTP requests and methods, refer to the REST API documentation here or to the examples provided in different languages on our GitHub page.

If you the information provided on these pages or in the links above is not sufficient for your query, proceed to contacting the API Team at api@fxcm.com.