Java

Java and API-NG

This page contains some code snippets of Java interaction with API-NG. This example shows how to use Java to send requests to list the event types, find the next horse racing market and then placing a bet with an invalid stake to trigger an error. The code referred to here is available at https://github.com/betfair/API-NG-sample-code/tree/master/java.

In the sample code we use the json-rpc and rescript protocols. All requests are sent and received using json format. To post a request we prepare the json string using Java objects and then we serialize the object using the gson library. The example requests contain some predefined data that can be modified depends on user needs.

Prerequisites

This is a maven project containing Java sample code to connect to the Betfair API-NG application.
It requires:
- Apache Maven 3
- Java 7

How to run it

You first build the project with:
mvn clean package

You can again use Apache Maven to run the application passing the app key, session token and the method (json-rpc or rescript):
mvn exec:java -Dexec.mainClass="com.betfair.aping.ApiNGDemo" -Dexec.args="<YOUR APP KEY> <YOUR SESSION TOKEN> <METHOD>"

<YOUR APP KEY>: a valid app key
<YOUR SESSION TOKEN>: a valid Betfair session token
<METHOD>: *json-rpc* or *rescript*

example: *mvn exec:java -Dexec.mainClass="com.betfair.aping.ApiNGDemo" -Dexec.args="myAppKey mySessionToken json-rpc"*

An addition code snippets for **Error Handling - non HTTP 200 responses** is included below, but does not form part of the code included the the git repository.

Code Snippet

Creating the request - JSON-RPC

```
String requestString;
//Handling the JSON-RPC request
JsonrpcRequest request = new JsonrpcRequest();
request.setId("1");
request.setMethod(ApiNGDemo.getProp().getProperty("SPORTS_APING_V1_0") + operation);
request.setParams(params);

requestString =  JsonConverter.convertToJson(request);
if(ApiNGDemo.isDebug())
         System.out.println("\nRequest: "+requestString);

//We need to pass the "sendPostRequest" method a string in util format:  requestString
HttpUtil requester = new HttpUtil();
return requester.sendPostRequestJsonRpc(requestString, operation, appKey, ssoToken);
```

Creating the request - Rescript

```
String requestString;
        //Handling the Rescript request
        params.put("id", 1);

        requestString =  JsonConverter.convertToJson(params);
        if(ApiNGDemo.isDebug())
            System.out.println("\nRequest: "+requestString);

        //We need to pass the "sendPostRequest" method a string in util format:  requestString
        HttpUtil requester = new HttpUtil();
        String response = requester.sendPostRequestRescript(requestString, operation, appKey, ssoToken);
        if(response != null)
            return response;
        else
            throw new APINGException();
```

Calling API-NG

```
String jsonRequest = param;
HttpPost post = new HttpPost(URL);
String resp = null;
try {
            post.setHeader(HTTP_HEADER_CONTENT_TYPE, ApiNGDemo.getProp().getProperty("APPLICATION_JSON"));
            post.setHeader(HTTP_HEADER_ACCEPT, ApiNGDemo.getProp().getProperty("APPLICATION_JSON"));
            post.setHeader(HTTP_HEADER_ACCEPT_CHARSET, ApiNGDemo.getProp().getProperty("ENCODING_UTF8"));
            post.setHeader(HTTP_HEADER_X_APPLICATION, appKey);
            post.setHeader(HTTP_HEADER_X_AUTHENTICATION, ssoToken);

            post.setEntity(new StringEntity(jsonRequest, ApiNGDemo.getProp().getProperty("ENCODING_UTF8")));

            HttpClient httpClient = new DefaultHttpClient();

            HttpParams httpParams = httpClient.getParams();
            HttpConnectionParams.setConnectionTimeout(httpParams, new Integer(ApiNGDemo.getProp().getProperty
("TIMEOUT")).intValue());
            HttpConnectionParams.setSoTimeout(httpParams, new Integer(ApiNGDemo.getProp().getProperty
("TIMEOUT")).intValue());

            resp = httpClient.execute(post, reqHandler);

        } catch (UnsupportedEncodingException e1) {
            //Do something

        } catch (ClientProtocolException e) {
            //Do something

        } catch (IOException ioE){
            //Do something

        }

        return resp;
```

Find Horse Racing event type id

```
MarketFilter marketFilter;
            marketFilter = new MarketFilter();
            Set<String> eventTypeIds = new HashSet<String>();

            System.out.println("1.(listEventTypes) Get all Event Types...\n");
            List<EventTypeResult> r = jsonOperations.listEventTypes(marketFilter, applicationKey, sessionToken);
            System.out.println("2. Extract Event Type Id for Horse Racing...\n");
            for (EventTypeResult eventTypeResult : r) {
                if(eventTypeResult.getEventType().getName().equals("Horse Racing")){
                    System.out.println("3. EventTypeId for \"Horse Racing\" is: " + eventTypeResult.
getEventType().getId()+"\n");
                    eventTypeIds.add(eventTypeResult.getEventType().getId().toString());
                }
            }
```

Get next available horse races:

```
System.out.println("4.(listMarketCataloque) Get next horse racing market in the UK...\n");
            TimeRange time = new TimeRange();
            time.setFrom(new Date());

            Set<String> countries = new HashSet<String>();
            countries.add("GB");

            Set<String> typesCode = new HashSet<String>();
            typesCode.add("WIN");

            marketFilter = new MarketFilter();
            marketFilter.setEventTypeIds(eventTypeIds);
            marketFilter.setMarketStartTime(time);
            marketFilter.setMarketCountries(countries);
            marketFilter.setMarketTypeCodes(typesCode);

            Set<MarketProjection> marketProjection = new HashSet<MarketProjection>();
            marketProjection.add(MarketProjection.COMPETITION);
            marketProjection.add(MarketProjection.EVENT);
            marketProjection.add(MarketProjection.EVENT_TYPE);
            marketProjection.add(MarketProjection.MARKET_DESCRIPTION);
            marketProjection.add(MarketProjection.RUNNER_DESCRIPTION);

            String maxResult = "1";

            List<MarketCatalogue> marketCatalogueResult = jsonOperations.listMarketCatalogue(marketFilter,
marketProjection, MarketSort.FIRST_TO_START, maxResult,
                    applicationKey, sessionToken);
```

Get list of runners in the market:

```
System.out.println("5. Print static marketId, name and runners....\n");
            printMarketCatalogue(marketCatalogueResult.get(0));
            /**
             * ListMarketBook: get list of runners in the market, parameters:
             * marketId:  the market we want to list runners
             *
             */
            System.out.println("6.(listMarketBook) Get volatile info for Market including best 3 exchange
prices available...\n");
            String marketIdChosen = marketCatalogueResult.get(0).getMarketId();

            PriceProjection priceProjection = new PriceProjection();
            Set<PriceData> priceData = new HashSet<PriceData>();
            priceData.add(PriceData.EX_ALL_OFFERS);
            priceData.add(PriceData.EX_BEST_OFFERS);
            priceData.add(PriceData.EX_TRADED);
            priceData.add(PriceData.SP_AVAILABLE);
            priceData.add(PriceData.SP_TRADED);

            //In this case we don't need these objects so they are declared null
            OrderProjection orderProjection = null;
            MatchProjection matchProjection = null;
            String currencyCode = null;

            List<String> marketIds = new ArrayList<String>();
            marketIds.add(marketIdChosen);

            List<MarketBook> marketBookReturn = jsonOperations.listMarketBook(marketIds, priceProjection,
                    orderProjection, matchProjection, currencyCode, applicationKey, sessionToken);
```

Place a bet:

```
long selectionId = 0;
            if ( marketBookReturn.size() != 0 ) {
                Runner runner = marketBookReturn.get(0).getRunners().get(0);
                selectionId = runner.getSelectionId();
                System.out.println("7. Place a bet below minimum stake to prevent the bet actually " +
                        "being placed for marketId: "+marketIdChosen+" with selectionId: "+selectionId+"...
\n\n");
                List<PlaceInstruction> instructions = new ArrayList<PlaceInstruction>();
                PlaceInstruction instruction = new PlaceInstruction();
                instruction.setHandicap(0);
                instruction.setSide(Side.BACK);
                instruction.setOrderType(OrderType.LIMIT);

                LimitOrder limitOrder = new LimitOrder();
                limitOrder.setPersistenceType(PersistenceType.LAPSE);
                //API-NG will return an error with the default size=0.01. This is an expected behaviour.
                //You can adjust the size and price value in the "apingdemo.properties" file
                limitOrder.setPrice(getPrice());
                limitOrder.setSize(getSize());

                instruction.setLimitOrder(limitOrder);
                instruction.setSelectionId(selectionId);
                instructions.add(instruction);

                String customerRef = "1";

                PlaceExecutionReport placeBetResult = jsonOperations.placeOrders(marketIdChosen, instructions,
customerRef, applicationKey, sessionToken);

                // Handling the operation result
                if (placeBetResult.getStatus() == ExecutionReportStatus.SUCCESS) {
                    System.out.println("Your bet has been placed!!");
                    System.out.println(placeBetResult.getInstructionReports());
                } else if (placeBetResult.getStatus() == ExecutionReportStatus.FAILURE) {
                    System.out.println("Your bet has NOT been placed :*( ");
                    System.out.println("The error is: " + placeBetResult.getErrorCode() + ": " + placeBetResult.
getErrorCode().getMessage());
                    System.out.println("Sorry, more luck next time\n\n");
                }
            } else {
                System.out.println("Sorry, no runners found\n\n");
            }
```

Error Handling - non HTTP 200 responses

The below guidance is for customers who are looking to handle additional HTTP responses other than the HTTP 200 success code when using Rescript requests.

1. In *RescriptResponseHandler* we need to modify the handleResponse method so that an HttpResponseException is thrown for unsuccessful response

```
public class RescriptResponseHandler implements
ResponseHandler<String> {


private static final String ENCODING_UTF_8 = "UTF-8";


    public String handleResponse(HttpResponse
response) throws ClientProtocolException, IOException {


        StatusLine statusLine =
response.getStatusLine();


        HttpEntity entity =
response.getEntity();


        String responseString =
EntityUtils.toString(entity,ENCODING_UTF_8);


        if
(statusLine.getStatusCode() != 200 ) {


           throw
new HttpResponseException(statusLine.getStatusCode(), responseString);


        }

        return entity == null ?
null : responseString;


    }

}
```

2. In *HttpUtil. sendPostRequest()* we catch the HttpResponseException, convert the Json response and eventually throw an APINGException containing the error details:

```
private String sendPostRequest(String param, String operation,
String appKey, String ssoToken, String URL, ResponseHandler<String>
reqHandler) throws APINGException {


        String jsonRequest =
param;

..….


……

resp = httpClient.execute(post, reqHandler);


        } catch
(UnsupportedEncodingException e1) {
```

```
//Do something



        }

catch(HttpResponseException ex){


ResponseError container = JsonConverter.convertFromJson(ex.getMessage(), new
TypeToken<ResponseError>() {}.getType());


APINGException  apingException=container.getDetail().getAPINGException();


if(apingException==null){

apingException= new APINGException();

apingException.setErrorCode(container.getFaultcode());

apingException.setErrorDetails(container.getFaultstring());


}

throw apingException;


         }


        catch
(ClientProtocolException e) {

//Do something



        } catch (IOException
ioE){


//Do something


        }

        return resp;


    }
```

3. ***Notice that the error handling code from sendPostRequest uses a new class:*** **ResponseError, which we are using to deserialize the error response**

```
public class ResponseError {


    private Detail detail;

    private String  faultcode;

    private String  faultstring;

    public String getFaultstring() {


        return faultstring;
```

```
    }

    public void setFaultstring(String faultstring)
{

        this.faultstring =
faultstring;

    }

    public String getFaultcode() {

        return faultcode;

    }

    public void setFaultcode(String faultcode) {

        this.faultcode =
faultcode;

    }

    public Detail getDetail() {

        return detail;

    }

    public void setDetail(Detail detail) {

        this.detail = detail;

    }

}

public class Detail {

    private APINGException APINGException;

    public APINGException getAPINGException() {

        return APINGException;

    }

    public void setAPINGException(APINGException
aPINGException) {

        APINGException =
aPINGException;

    }
```