Physical Prices API
User's Manual
For Java
9th March 2020

INTRODUCTION

A key part of the data solutions provided by Fastmarkets are the various REST APIs designed to provide data in a flexible, performant and secure way. For data license customers, these APIs are ideal for retrieving and processing Fastmarkets data by their own services.

This document gives details of two of these APIs with examples of how they are typically used.

THE APIs

The following public APIs are available for use to authorized customers

- **Fastmarkets Authentication API**
  This API is used to authenticate the calling service. It will also generate an access token needed when calling other Fastmarkets APIs for authorization purposes.
- **Fastmarkets Physical Prices API**
  Used to return current and historic physical prices as well as associated instrument data

FASTMARKETS AUTHENTICATION API

All the Fastmarkets APIs use well established modern security standards. This includes the use of OAuth 2.0, OpenID Connect and JSON Web Tokens.

Customers of Fastmarkets data will typically have their own services that will consume and process data. As such, authentication based on a single user's credentials are not appropriate. Instead, a unique Service Key and Service Name is issued by Fastmarkets which can be used when calling the Fastmarkets Authentication API to generate an Access Token. The returned Access Token (in the form of a JSON Web Token) is then needed for any subsequent calls to other Fastmarkets APIs (such as the Fastmarkets Physical Prices API) to verify that the calling service has the necessary permissions to view the requested data.

Generating an Access Token

Using the provided Service Key and Service Name, a POST Connect Token request is required to the Fastmarkets Authentication API as described in the example below. The response includes the Access Token itself, an Expiry time (in seconds) and the type of Access Token (this will always be of type 'Bearer').

Example Request:

```
void getAccessTokenFromAuthServer() {
 HttpHeaders headers = new HttpHeaders();
 headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
 HttpEntity < MultiValueMap < String, String >> request = new HttpEntity < > (requestParams(),
headers);
 ResponseEntity < Token > response = restTemplate.exchange(
  configProperties.getProperty("webservice.uri.auth").concat(AUTH_URL), HttpMethod.POST,
request, Token.class);
 this.token = response.getBody();
}
MultiValueMap < String, String > requestParams() {
 MultiValueMap < String, String > params = new LinkedMultiValueMap < > ();
 params.add("grant_type", configProperties.getProperty("security.oauth2.client.grant-type"));
 params.add("client_id", configProperties.getProperty("security.oauth2.client.client-id"));
 params.add("scope", configProperties.getProperty("security.oauth2.client.scope"));
 params.add("serviceKey", configProperties.getProperty("security.oauth2.client.service-key"));
 params.add("serviceName", configProperties.getProperty("security.oauth2.client.service-name"));
 return params;
}
webservice.uri.auth = https: //auth.fastmarkets.com
 static final String AUTH_URL = "/connect/token";
public class Token {
 @JsonProperty("access_token")
 private String accessToken;
 @JsonProperty("expires_in")
 private Integer expiresIn;
 @JsonProperty("token_type")
 private String tokenType;

}
```

Example Response:

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjdjZTkyOTQ4NDk0ODRkMDM4YzQ0N...eEJkMWK
geY4vumE1RY-h4pC1CT6w",
  "expires_in": 3600,
  "token_type": "Bearer"
}
```

Please note that the Access Token will expire. The expiry time is provided in the response. This is by design. Depending on requirements, a new Access Token will need to be generated in one or more of the following circumstances:

1. Before every request to an API that requires an Access Token (such as the Physical Prices API) - This could prove inefficient if making frequent data requests, but can suit stateless processing.
2. When the current Access Token is close to expiry – The 'expires in' value returned with Connect Token request provides the expiry period in seconds. Alternatively, decoding the JSON Web Token (JWT) itself will reveal a parameter called 'exp' with a timestamp value representing the expiry time.
3. On receiving a Status 401 (Unauthorized) response from an API

Fastmarkets Physical Prices API

The Physical Prices API provides price values and associated instrument data for Fastmarkets assessed prices. All price values are associated to instruments which uses a symbol as an identifier.

For more on the specification of this API and to try it out, please refer to the API's documentation page (Swagger) here: https://api.fastmarkets.com/physical/v2/documentation

Authenticating

All Fastmarkets APIs require a valid Access Token in order to retrieve permissioned data. To generate an Access Token, please refer to the Fusion Authentication API section above. The token is then added to an Authorization header parameter using the 'Bearer' prefix. For example:

Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjdjZTkyOTQ4NDk0ODRkMDM4YzQ

Retrieving a single price

To return the most recently available assessed price for a specific instrument, the Prices endpoint is used. In this example, the symbol 'MB-AL-0004' is used to return the latest available price data for 'Aluminium P1020A, in-warehouse Rotterdam duty-paid, spot $/tonne' as of 2nd March 2019.

In the response, low, mid and high price values are returned for the 1st March 2019, as this was the most recent assessment available for the specified date.

Example Request:

```
InstrumentPrice price = getInstrumentPriceByDate("MB-AL-0004", "2019-03-02");

public InstrumentPrice getInstrumentPriceByDate(String symbol, String date) {
 HttpEntity < Object > request = new HttpEntity < > (instrumentPriceByDateRequestParams(symbol,
date),
  authorizationHeader());
 String serviceUrl = serviceUri().concat(PRICE_BY_DATE);
 ResponseEntity < InstrumentPricesResponse > response = restTemplate.exchange(serviceUrl,
HttpMethod.POST, request,
  InstrumentPricesResponse.class);
}

MultiValueMap < String, String > instrumentPriceByDateRequestParams(String symbol, String date)
{
  MultiValueMap < String, String > params = new LinkedMultiValueMap < > ();
  params.add(Instrument.SYMBOLS_PARAM_KEY, symbol);
  params.add(Instrument.DATES_PARAM_KEY, date);
  return params;
 }

 --

String serviceUri() {
 String serviceUri = configProperties.getProperty("webservice.uri.physical");
 return serviceUri;
}

webservice.uri.physical = https: //api.fastmarkets.com/physical/v2

 static final String PRICE_BY_DATE = "/Prices";
```

Example Response:

```
  "instruments": [
   {
    "firstDate": "1987-04-07T00:00:00+00:00",
    "lastDate": "2019-05-21T15:00:00+00:00",
    "prices": [
     {
      "date": "2019-03-02",
      "assessmentDate": "2019-03-01T16:00:12+00:00",
      "revision": 0,
      "low": 130,
      "mid": 135,
      "high": 140
     }
    ],
    "symbol": "MB-AL-0004"
   }
  ]
}
```


If no value for the Dates parameter is included in the request, then the most recent price data is returned. A value for the Symbols parameter is always required.

Retrieving multiple prices

It is also possible to request prices for multiple instruments and multiple dates in single request using the Prices endpoint.

In the example request below, two different symbols and two different dates have been requested. In the result, there will be two price results for each of the two instruments.

Example Request:

```
InstrumentPrice price = getMultipleInstrumentPriceByDate(new String[] {
 "MB-AL-0004",
 "MB-AL-0006"
}, new String[] {
 "2019-03-02",
 "2019-03-04"
});

public InstrumentPrice getMultipleInstrumentPriceByDate(String[] symbols, String[] dates) {
 HttpEntity < Object > request = new HttpEntity < >
(instrumentMultiplePriceByDateRequestParams(symbols, dates),
  authorizationHeader());
 String serviceUrl = serviceUri().concat(PRICE_BY_DATE);
 ResponseEntity < InstrumentPricesResponse > response = restTemplate.exchange(serviceUrl,
HttpMethod.POST, request,
  InstrumentPricesResponse.class);
}

MultiValueMap < String, String > instrumentMultiplePriceByDateRequestParams(String[] symbols,
String[] dates) {
 MultiValueMap < String, String > params = new LinkedMultiValueMap < > ();
 for (String symbol: symbols) {
  params.add(Instrument.SYMBOLS_PARAM_KEY, symbol);
 }
 for (String date: dates) {
  params.add(Instrument.DATES_PARAM_KEY, date);
 }
 return params;
}
```

Example Response:


```
{
 "instruments": [
  {
   "firstDate": "1987-04-07T00:00:00+00:00",
   "lastDate": "2019-05-21T15:00:00+00:00",
   "prices": [
    {
     "date": "2019-03-02",
     "assessmentDate": "2019-03-01T16:00:12+00:00",
     "revision": 0,
     "low": 130,
     "mid": 135,
     "high": 140
    },
    {
     "date": "2019-03-04",
     "assessmentDate": "2019-03-01T16:00:12+00:00",
     "revision": 0,
     "low": 130,
     "mid": 135,
     "high": 140
    }
   ],
   "symbol": "MB-AL-0004"
  },
  {
   "firstDate": "1995-07-05T00:00:00+00:00",
   "lastDate": "2019-05-22T15:42:17+00:00",
   "prices": [
    {
     "date": "2019-03-02",
     "assessmentDate": "2019-02-27T15:06:11+00:00",
     "revision": 0, . . . . . .
```


Retrieving a range of prices

Using the Prices/History endpoint, it is possible to retrieve a series of prices over a specified time period. In this example, a request is made over a seven-day period (between 20th Feb 2019 to 27th Feb 2019). The response returns prices in descending order of date.

Example Request:

```
InstrumentPrice price = getInstrumentPriceHistory(new String[] {
 "MB-IR-0001"
}, "2019-02-20", "2019-02-27")

public InstrumentPrice getInstrumentPriceHistory(String[] symbols, String fromDate, String toDate)
{
 HttpEntity < Object > request = new HttpEntity < > (instrumentPriceHistory(symbols, fromDate,
toDate),
  authorizationHeader());
 String serviceUrl = serviceUri().concat(PRICE_HISTORY_BY_DATE);
 ResponseEntity < InstrumentPricesResponse > response = restTemplate.exchange(serviceUrl,
HttpMethod.POST, request,
  InstrumentPricesResponse.class);
}

MultiValueMap < String, String > instrumentPriceHistoryRequestParams(String[] symbols, String
fromDate, String toDate) {
 MultiValueMap < String, String > params = new LinkedMultiValueMap < > ();
 for (String symbol: symbols) {
  params.add(Instrument.SYMBOLS_PARAM_KEY, symbol);
 }
 params.add(Instrument.FROM_DATE_PARAM_KEY, fromDate);
 params.add(Instrument.TO_DATE_PARAM_KEY, toDate);
 return params;
}

static final String PRICE_HISTORY_BY_DATE = "/Prices/history";
```

Example Response:


```
{
  "instruments": [
    {
      "firstDate": "1987-01-06T00:00:00+00:00",
      "lastDate": "2019-05-22T09:17:04+00:00",
      "prices": [
        {
          "date": "2019-02-27",
          "assessmentDate": "2019-02-27T09:29:33+00:00",
          "revision": 0,
          "low": 1410,
          "mid": 1460,
          "high": 1510
        },
        {
          "date": "2019-02-26",
          "assessmentDate": "2019-02-22T10:36:29+00:00",
          "revision": 0,
          "low": 1410,
          "mid": 1460,
          "high": 1510
        },
        {
          "date": "2019-02-25",
          "assessmentDate": "2019-02-20T09:31:14+00:00",
          "revision": 0,
          "low": 1409, . . . . .
```


Retrieving average prices

Periodically, average prices for many Fastmarkets instruments are published. These are calculated values based on the underlying assessment prices over a period of a week, month or year.

Both the Prices and Prices History endpoints accept an input parameter called Price Calculation Type. There are several valid values for this parameter, the most common of which are:

- WeeklyAverage
- MonthlyAverage

- YearlyAverage

If no Price Calculation Type is specified, then the actual assessment value is returned.

NB: To find out what Price Calculation Types are available for a given instrument, use the Instrument endpoint (see section: Retrieving instruments data)

Example Request:

```
public InstrumentPrice getInstrumentPriceMonthlyAverageByDate(String symbol, String date,
String priceCalculationTypeID) {
 HttpEntity < Object > request = new HttpEntity < > (instrumentPriceByDateRequestParams(symbol,
date),
  authorizationHeader());
 String serviceUrl = serviceUri().concat(PRICE_BY_DATE);
 serviceUrl = serviceUri().concat(MONTHLY_AVE_PRICE_BY_DATE);
 ResponseEntity < InstrumentPricesResponse > response = restTemplate.exchange(serviceUrl,
HttpMethod.POST, request,
  InstrumentPricesResponse.class);
}


static final String MONTHLY_AVE_PRICE_BY_DATE = "/Prices/MonthlyAverage";
```

Example Response:


```
{
 "instruments": [
  {
   "firstDate": "2008-01-31T12:00:00+00:00",
   "lastDate": "2019-04-30T12:00:00+00:00",
   "prices": [
    {
     "date": "2019-03-02",
     "assessmentDate": "2019-02-28T12:00:00+00:00",
     "revision": 0,
     "low": 125,
     "mid": 130.31,
     "high": 135.62
    }
   ],
   "symbol": "MB-AL-0004"
  }
 ]
}
```


Optional price data fields

When requesting price data, not all available data associated to a price is returned by default. This is by design to help reduce the size of the response if requesting a large quantity of records.

However, this additional data can be returned by using the Fields input parameter. Here is a list of the available optional fields that can be added to the request:

- appraisalPrice - Value representing the whether undergoing an appraisal process at the point of assessment (Boolean)
- pricingRationale - Description of the rationale behind the assessment made by the Price Reporter (string)
- assessmentPeriod - Description of the assessment period when returning average price calculations types (string)
- lowChangeSincePrevious - Difference between low price value of previous assessment and low price value of this assessment (number)
- midChangeSincePrevious – (As above but for mid price value)
- highChangeSincePrevious – (As above but for high price value)

- lowChangeSincePreviousProportion - Difference between low price value of previous assessment and low price value of this assessment as a decimal value. 1 represents a change of 100%, -1 represents a change of -100% (number)
- midChangeSincePreviousProportion – (As above but for mid price value)
- highChangeSincePreviousProportion – (As above but for high price value)

Example Request:

```
InstrumentPrice price = getInstrumentPriceByDateWithOptionalFields("MB-AL-004", "2019-03-02",
"midChangeSincePrevious,pricingRationale")

public InstrumentPrice getInstrumentPriceByDateWithOptionalFields(String symbol, String date,
String optFields) {
 HttpEntity < Object > request = new HttpEntity < >
(instrumentPriceByDateRequestParamsWithOptionalFields(symbol, date, optFields),
  authorizationHeader());
 String serviceUrl = serviceUri().concat(PRICE_BY_DATE);
 ResponseEntity < InstrumentPricesResponse > response = restTemplate.exchange(serviceUrl,
HttpMethod.POST, request,
  InstrumentPricesResponse.class);
}

MultiValueMap < String, String > instrumentPriceByDateRequestParamsWithOptionalFields(String
symbol, String date, String optFields) {
 MultiValueMap < String, String > params = new LinkedMultiValueMap < > ();
 params.add(Instrument.SYMBOLS_PARAM_KEY, symbol);
 params.add(Instrument.DATES_PARAM_KEY, date);
 params.add(Instrument.FIELDS_PARAM_KEY, optFields);
 return params;
}
```

Retrieving instrument data

All physical prices relate to an associated instrument. The instrument consists of various attributes, all of which are available to view using the Instrument endpoint.

If no input parameters are provided, all instruments that the calling service are entitled to see are returned. The Symbols input parameter can be used return specific instruments (see example).

Example Request:

```
Instrument instrument = getInstrument("MB-STS-0236");

public Instrument getInstrument(String symbol) {
 HttpEntity < Object > request = new HttpEntity < > (instrumentRequestParams(symbol),
authorizationHeader());
 ResponseEntity < InstrumentDetailsResponse > response =
restTemplate.exchange(serviceUri().concat(INSTRUMENT_URI),
  HttpMethod.POST, request, new ParameterizedTypeReference < InstrumentDetailsResponse > ()
{});

 if (!CollectionUtils.isEmpty(response.getBody().getInstruments())) {
  Instrument instrument = response.getBody().getInstruments().iterator().next();
  return instrument;
 }
}

static final String INSTRUMENT_URI = "/Instruments";

MultiValueMap < String, String > instrumentRequestParams(String symbols) {
 MultiValueMap < String, String > params = new LinkedMultiValueMap < > ();
 params.add(Instrument.SYMBOLS_PARAM_KEY, symbols);
 return params;
}
```

Example Response:

```
{
 "instruments": [
  {
   "productId": "Broker 304 turnings",
   "description": "New York 304 turnings, broker buying, US cents per pound",
   "descriptionShort": "New York 304 turnings, broker buying, US c/lb",
   "commodityId": "STS",
   "priceType": "Price",
   "locationId": "USA-NY",
   "currencyId": "USd",
   "unitOfMeasureId": "PUND",
   "incotermId": "DLVD",
   "launchDate": "2015-10-20",
   "frequency": "Weekly",
   "sourceId": "AMM",
   "status": "Active",
   "priceCalculationTypeIds": [
    "WeeklyAverage",
    "MonthlyAverage",
    "YearlyAverage"
   ],
   "symbol": "MB-STS-0236"
```


By default, many of the attributes returned are ID values (for example: Commodity ID and Currency ID). To return the full name of these attributes, they need to be included in the Fields input parameter as they are optional.

These optional fields include:

- Commodity
- Location
- Currency
- UnitOfMeasure
- Incoterm
- Source

In this example, a request is made to include Commodity and Currency names in the response

Example Request:

```
public Instrument getInstrument(String symbol) {
 HttpEntity < Object > request = new HttpEntity < > (instrumentRequestParams(symbol),
authorizationHeader());
 ResponseEntity < InstrumentDetailsResponse > response =
restTemplate.exchange(serviceUri().concat(INSTRUMENT_URI),
  HttpMethod.POST, request, new ParameterizedTypeReference < InstrumentDetailsResponse > ()
{});

 if (!CollectionUtils.isEmpty(response.getBody().getInstruments())) {
  Instrument instrument = response.getBody().getInstruments().iterator().next();
  return instrument;
 }
}

static final String INSTRUMENT_URI = "/Instruments";

MultiValueMap < String, String > instrumentRequestParams(String symbols) {
 MultiValueMap < String, String > params = new LinkedMultiValueMap < > ();
 params.add(Instrument.SYMBOLS_PARAM_KEY, symbols);
 params.add(Instrument.FIELDS_PARAM_KEY, instrumentAdditionalFields());
 return params;
}

String instrumentAdditionalFields() {
 return "commodity,currency");
}
```

Retrieving reference data

The References endpoint is useful for obtaining details of all valid values for a particular field (such as currency codes) or to obtain the full name a specific reference value.

The following reference data is available using this endpoint:

- Currency
- UnitOfMeasure
- PriceCalculationType
- Incoterm
- Commodity
- Source

Example Request:

```
ReferenceData referenceData = getReferenceData("currency");

public ReferenceData getReferenceData(String types) {
 HttpEntity < Object > request = new HttpEntity < > (referenceDataRequestParams(types),
authorizationHeader());
 ResponseEntity < ReferenceDataResponse > response =
restTemplate.exchange(serviceUri().concat(REFERENCE_URI),
  HttpMethod.POST, request, new ParameterizedTypeReference < ReferenceDataResponse > () {});

 if (!CollectionUtils.isEmpty(response.getBody().getReferenceData())) {
  ReferenceData referenceData = response.getBody().getReferenceData().iterator().next();
  return referenceData;
 }
}

static final String ReferenceData_URI = "/References";

MultiValueMap < String, String > referenceDataRequestParams(String types) {
 MultiValueMap < String, String > params = new LinkedMultiValueMap < > ();
 params.add(ReferenceData.TYPES_PARAM_KEY, types);
 return params;
}
```

Example Response:


```
{
 "references": [
  {
   "type": "Currency",
   "items": [
    {
     "sign": "¥",
     "id": "CNY",
     "description": "China Yuan"
    },
    {
     "sign": "$",
     "id": "ARS",
     "description": "Argentine peso"
    },
    {
     "sign": "£",
     "id": "GBP",
     "description": "British Pounds"
    }, . . . .
```