4. Streaming Data

This section is about **real-time or streaming data**.

In [1]:

```
import fxcmpy
import pandas as pd
import datetime as dt
con = fxcmpy.fxcmpy(config_file='fxcm.cfg')
```

4.1. Basic Approach

Streaming the current market data is straightforward, one just need to **subscribe** to the instrument of interest.

In [2]:

```
con.subscribe_market_data('EUR/USD')
```

A list with the subscribed instruments is returned by the `con.get_subscribed_symbols()` method.

In [3]:

```
con.get_subscribed_symbols()
```

Out[3]:

```
['EUR/USD']
```

One can also check the subscription for a specified instruments.

In [4]:

```
con.is_subscribed('EUR/USD')
```

Out[4]:

```
True
```

After the subscription, `fxcmpy` collects the data in a `pandas DataFrame`. The `con.get_prices()` method returns that `DataFrame` object.

In [5]:

```
con.get_prices('EUR/USD')
```

Out[5]:

	Bid	Ask	High	Low
2018-06-06 10:02:42.588	1.17718	1.17743	1.17751	1.17095
2018-06-06 10:02:45.767	1.17717	1.17743	1.17751	1.17095
2018-06-06 10:02:47.469	1.17716	1.17742	1.17751	1.17095
2018-06-06 10:02:47.737	1.17716	1.17741	1.17751	1.17095
2018-06-06 10:02:53.435	1.17716	1.17742	1.17751	1.17095
2018-06-06 10:02:54.864	1.17718	1.17742	1.17751	1.17095
2018-06-06 10:02:56.277	1.17719	1.17742	1.17751	1.17095
2018-06-06 10:02:57.461	1.17719	1.17744	1.17751	1.17095
2018-06-06 10:03:20.470	1.17720	1.17744	1.17751	1.17095
2018-06-06 10:03:21.441	1.17720	1.17745	1.17751	1.17095
2018-06-06 10:03:23.930	1.17721	1.17746	1.17751	1.17095
2018-06-06 10:03:27.619	1.17722	1.17747	1.17751	1.17095
2018-06-06 10:03:32.414	1.17721	1.17747	1.17751	1.17095
2018-06-06 10:03:34.200	1.17721	1.17746	1.17751	1.17095
2018-06-06 10:03:35.389	1.17721	1.17745	1.17751	1.17095

You can also fetch only the last available quotes via `con.get_last_price()`.

In [6]:

```
con.get_last_price('EUR/USD')
```

Out[6]:

```
Bid     1.17729
Ask     1.17754
High    1.17756
Low     1.17095
Name: 2018-06-06 10:05:03.066000, dtype: float64
```

To prevent the resulting `DataFrame` from growing too large, the number of entries can be limited by the class attribute `max_prices`. It is set to 10,000 by default.

In [7]:

```
con.get_max_prices()
```

Out[7]:

```
10000
```

You can change the value of `max_prices` with `set_prices()`.

In [8]:

```
con.set_max_prices(1000)
```

`None` as value for max_prices means that there is no limit for the number of prices in the `DataFrame`.

To stop the stream and delete (!) the `DataFrame`, use `con.unsubscribe_market_data()`.

In [9]:

```
con.unsubscribe_market_data('EUR/USD')
```

In [10]:

```
con.get_prices('EUR/USD')
```

Out[10]:

```
  Bid Ask High Low
```

4.2. Callback Functions

One can add an iterable (e.g. `tuple` or `list` object) with **callback functions** to the subscription, as in the example below.

The function `print_data()` simply prints selected quotes as soon as they are retrieved.

In [11]:

```
def print_data(data, dataframe):
    print('%3d | %s | %s, %s, %s, %s, %s'
          % (len(dataframe), data['Symbol'],
             pd.to_datetime(int(data['Updated']), unit='ms'),
             data['Rates'][0], data['Rates'][1], data['Rates'][2],
             data['Rates'][3]))
```

In [12]:

```
con.subscribe_market_data('EUR/USD', (print_data,))
```

```
  1 | EUR/USD | 2018-06-06 10:06:29.800000, 1.17725, 1.17749, 1.17756, 1.17095
  2 | EUR/USD | 2018-06-06 10:06:34.421000, 1.17725, 1.17748, 1.17756, 1.17095
```

In [13]:

```
con.unsubscribe_market_data('EUR/USD')
```

In [14]:

```
con.close()
```