API Methods

Contents

Initialize Function

The first step in writing the strategy is to define the initialize function. Before the start of the strategy, the initialize() function will be called and passed in a context variable. The context is a persistent namespace for you to store variables you need to access, from anywhere in your algorithm.

Syntax:

```
def initialize(context):
  pass
```

Schedule Function

Inside the initialize function, you can schedule the strategy logic. The 'schedule_function' from zipline.api allows you to specify on what days and at what times you want a function to run. This means you can easily schedule a function to run once a day, once a month, or only execute orders 10 minutes before the market closes.

Syntax:

```
schedule_function(
        func = <<Name of the function>>,
        date_rule = <<Date rules>>,
        time_rule = <<Time rules>>
)
```

Parameters:

func: This is the first parameter to the schedule function. The name of the function to be scheduled is passed here.

date_rules: We can specify different rules to schedule the function. For example, every day, start of the week or the start of the month.

Monthly modes:

- month_start
- month_end

It accepts a days_offset parameter to offset the function execution by a specific number of trading days, from the beginning and end of the month respectively. All day calculations are done using trading days for your selected trading calendar. If the offset exceeds the number of trading days in a month, the function isn't run during that month.

Weekly modes:

- week_start
- week_end

It also accepts a days_offset parameter. If the function execution is scheduled for a market holiday and there is at least one more trading day in the week, the function will run on the next trading day. If there are no more trading days in the week, the function is not run during that week. Specifying date_rules as every_day will run it once a day.

time_rules:

- market open
- market close

It accepts hours and minutes as parameters.

Example:

The strategy function is called at the start of the week at 12 noon.

```
# Call the strategy function on the first trading day of each week at 12
noon
schedule_function(strategy,
        date_rules.week_start(days_offset=0),
        time_rules.market_open(hours=2, minutes=30))
```

Refer to this document to read more about schedule_function.

Optional Function

After the strategy has been initialized, the handle_data() function is called every minute. At every call, it passes the same context variable and a data object containing the current trading bar with open, high, low, and close (OHLC) prices, as well as volume for all the currency pairs.

Example:

```
def handle_data(context, data):
  pass
```

Data Fetching

The data.history function is used to fetch the price data for the currency pairs.

Parameters:

- assets: Currency pair symbol or iterable of currency pair symbols.
- fields: String or iterable of strings. Valid values are 'price', 'open', 'high', 'low', 'close', and 'volume'.
- bar_count: Integer number of bars of trade data.
- frequency: '1m' for minute data or '1d' for daily data. For other frequencies, use the pandas resample function.

Returns:

- pandas DataFrame indexed by the date

Example:

```
# Get the data for the past 100 days
  security_data = data.history(
                                assets = symbol(FXCM('EUR/USD')),
                                fields = 'price',
                          bar_count = 100,
                                )
```

'price' is forward-filled, returning last known price, if there is one, otherwise, NaN is returned. In the above code, we have stored the historical daily price data for the past 100 days in security_data pandas DataFrame.

Note:

data.history() can return any one of the following:

- A Pandas Series object (if the request is for single security and single field)
- A Pandas DataFrame object (multiple securities and single field - with securities in columns OR with single security and multiple fields - fields in columns)
- A Pandas Panel Data object (multiple securities AND multiple fields - securities in minor axis). Series or DataFrame objects returned will be indexed by dates and Panel Data objects will be indexed by fields. To learn more about Pandas Series, DataFrame and Panel data, see here.

Refer to this document to read more about data fetching on Blueshift.

Order Placement Functions

Once the core strategy logic is defined, you need to place orders to your broker. To place an order, strategy needs to specify an asset, and the other order parameters.

order(asset, quantity)

You can also specify market, limit orders or other order types, supported by the platform and broker you are using.

While order is the base function to trade, there are a bunch of other helper functions that do automatic order sizing. 'order_value' and 'order_target_percent' are widely used order sizing functions.

order_value(asset, value)

Place order for an asset with a specific dollar value.

order_target_percent(asset, percent)

Place order for an asset with a specific percent of current portfolio value.

To learn more about placing trade orders on Blueshift, follow this document.