

# Quinn's Oanda-Matlab Interface

## Instruction Manual

### Contents:

0: Notes

1: Versions

2: StartUp

2B: StartUp: OAPI Bot

3: Functions

3A: GetAccounts

3B: GetAccountInfo

3C: GetInstrument

3D: GetInstrumentData

3E: GetTransactionHistory

3F: GetTransactionInfo

3G: GetPrices

3H: StreamPrices

3I: GetHistory

3J: GetOrders

3K: GetTrades

3L: NewOrder

3M: ModifyTrade

3N: CloseTrade

### 0: Notes

Libraries:

Both JSONLAB and URLREAD2 are required for these functions to work, they have been included in the OAPI folders in their appropriate places with their respective licences.

JSONLab (Qianqian Fang <fangq at nmr.mgh.harvard.edu>)

URLREAD2 (Jim Hokanson)

Almost all inputs must be strings, this is because after the input has been processed all the information for the request will be sent to Oanda in the form of a URL: to make an input a string just put it inside inverted commas, E.G:

an instrument name: EUR\_USD, as a string: 'EUR\_USD'

a number of items to retrieve: 100, as a string: '100'

The functions work best with Metatrader accounts for currently unknown reasons, trades can be opened using these functions on an account using Oanda's Fxtrade platform but some information like trade history can not be retrieved.

### 1: Versions

Two versions of the Oanda-Matlab Interface exist, a version for bots and a version for humans.

The reason for this is that many functions can be made more flexible and useful (at the cost of efficiency and speed) for a human operator if that human is at the computer to respond to input requests. For example the GetAccounts function can save a specific account to the workspace for it's information to be accessed easily but only if a human operator exists to select the account, another example is the GetHistory function, this function can produce a candlestick chart of the information requested by the user only if the user is there to accept it, this extra functionality would not be useful to a trading bot and the creation of the plot would slow down it's operation. A solution to this would be to have two differently named GetHistory functions but I opted to have two entirely separate versions because this allows an algorithmic trading system to be created in one environment with a lot of human intervention and to run in another environment without any changes needing to be made, reducing the need for extra debugging and providing a level of confidence that the system you have designed will operate in practice just as it operates during experimentation.

## 2: Startup OAPI-Human

To Start and Initialize the interface, open MatLab and navigate to the folder containing the OAPI functions (I.E OAPI-Human) then type “ApiStart” in the command window.

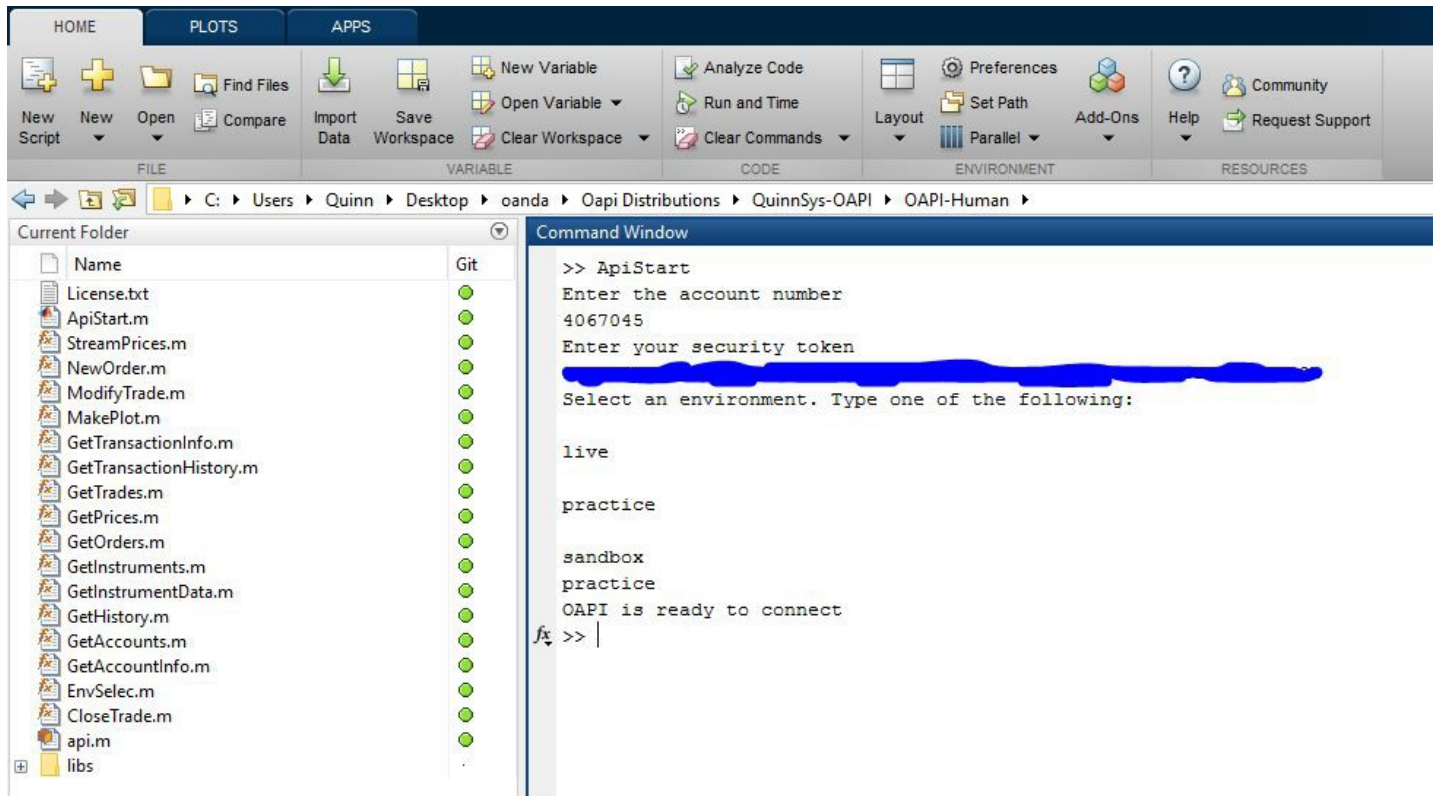
You will first be asked to enter an account number, this is identical to an accountId and will be the default account that functions use to operate.

Then you will be asked to enter your security token, this API key is associated with your whole OANDA account and will provide access to each sub-account, if you don't have a security token you can get one here:

[https://fxtrade.oanda.co.uk/account/tpa/personal\\_token](https://fxtrade.oanda.co.uk/account/tpa/personal_token)

Finally you will be asked to select an environment, typing live or practice will allow you to use either of these environments.

If all was entered correctly you will receive the message “OAPI is ready to connect” and you can begin using functions.

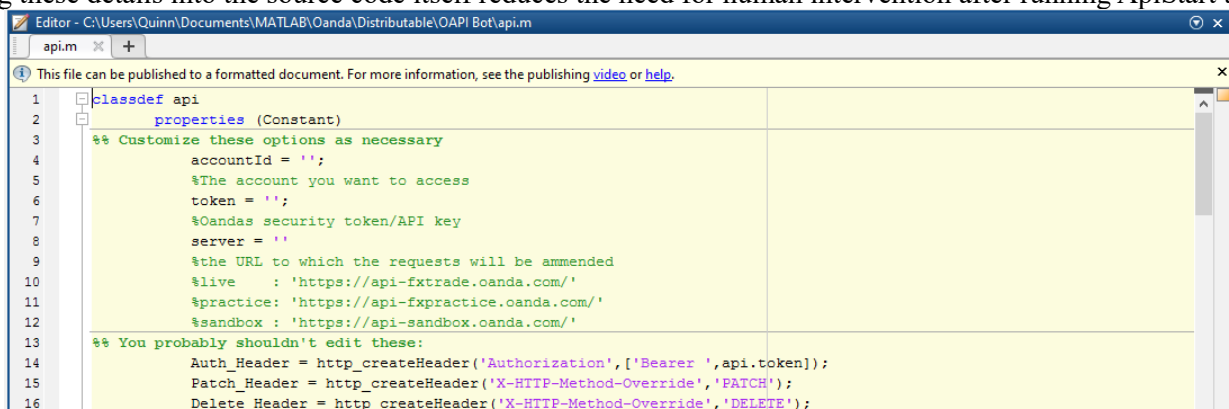


## 2B: StartUp OAPI Bot

To startup the version of OAPI meant for bots and algorithms to use, before following the steps above,

- 1: Navigate to OAPI-Bot
- 2: Open api.m
- 3: Fill in the empty apostrophes at the top of the file with your account number, security token and desired environment server
- 4: Save the file
- 5: You are now ready to run ApiStart as above

Entering these details into the source code itself reduces the need for human intervention after running ApiStart to zero.



## 3A: GetAccounts

### Syntax:

GetAccounts

### Input:

None

### Information:

This function has no input variables, simply enter “GetAccounts” into the matlab command window and the function will print the name, ID, currency and margin rate of all accounts associated with the API key supplied on startup.

The Human version of the function will then ask the user to select an account, the details of the selected account will then be outputted to the workspace.

## 3B: GetAccountInfo

### Syntax:

GetAccountInfo('#####')  
GetAccountInfo

### Input:

The only input is an accountId in inverted commas.

### Information:

This function returns the information for a selected account and has one input, the accountId of the account you want to request the information for.

This function can also be called with no input by typing just “GetAccountInfo”, this will run the GetAccounts function and return the information for the account selected.

The Bot Version requires an accountId as input and cannot run without one.

```
>> GetAccountInfo('357750')  
  
ans =  
  
        accountId: 357750  
    accountName: 'MatLab'  
        balance: 87.9691  
    unrealizedPl: 0  
    realizedPl: -0.0509  
    marginUsed: 0  
    marginAvail: 87.9691  
    openTrades: 0  
    openOrders: 0  
    marginRate: 0.0100  
accountCurrency: 'GBP'  
  
>>
```

## 3C: GetInstruments

### Syntax:

```
GetInstruments('#####');  
GetInstruments;
```

### Input:

The only input is an accountId in inverted commas.

### Information:

This function returns a list of all tradable instruments for a selected account and has one input, the accountId.

This function can also be called with no input by typing just “GetInstruments;”, this will run the GetAccounts function and return the tradable instruments for the account selected.

The Bot Version requires an accountId as input and cannot run without one.

```
>> GetInstruments('357750')  
  
ans =  
  
    'AU200_AUD'  
    'AUD_CAD'  
    'AUD_CHF'  
    'AUD_HKD'
```

## 3D: GetInstrumentData

### Syntax:

```
GetInstrumentData('#####')  
GetInstrumentData
```

### Input:

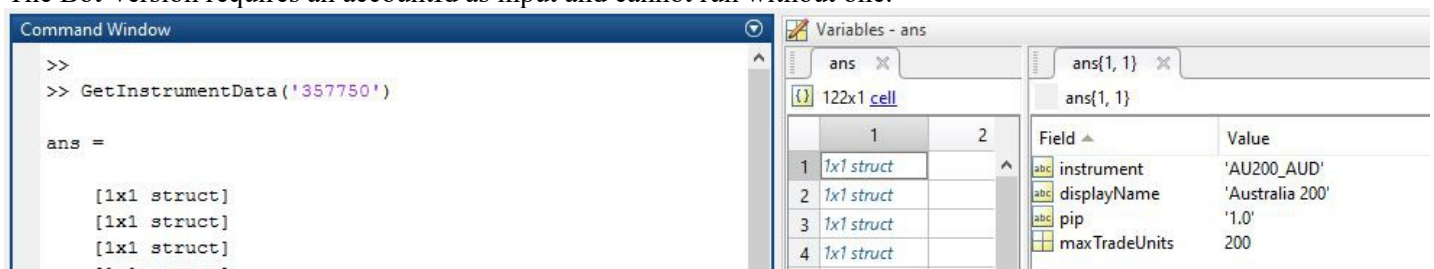
The only input is an accountId in inverted commas.

### Information:

This function returns the name, display name, pip size and maximum tradable units for every tradable instrument for the selected account.

This function can also be called with no input by typing just “GetInstrumentData”, this will run the GetAccounts function and return the information for all tradable instruments for the account selected.

The Bot Version requires an accountId as input and cannot run without one.



The screenshot shows the Command Window on the left and the Variables pane on the right. The Command Window displays the command `>> GetInstrumentData('357750')` and the output `ans =` followed by a list of four 1x1 struct arrays. The Variables pane shows the variable `ans` as a 122x1 cell array. A detailed view of `ans{1,1}` is shown, displaying the following data:

Field	Value
instrument	'AU200_AUD'
displayName	'Australia 200'
pip	'1.0'
maxTradeUnits	200

## 3E: GetTransactionHistory

### Syntax:

```
GetTransactionHistory('XXX_XXX')  
GetTransactionHistory
```

### Input:

The only input is the name of an instrument in the form: 'EUR\_USD'  
A full list of instrument names can be returned using the GetInstruments function.

### Information:

This function returns the information for the last 50 transactions on the default account (the account whose accountId the user entered during startup). The transaction history for a specific instrument can be returned by using the name of that instrument as an input.

The screenshot shows the MATLAB Command Window with the command `>> GetTransactionHistory('EUR_USD')` and the output `ans =` followed by four `[1x1 struct]` placeholders. The Variables pane on the right shows `ans{1,1}` as a 1x7 cell array. A detailed view of the first transaction is shown in a table:

Field	Value
id	'10080186970'
accountId	456977
time	'2016-01-21T14:31:02.000000Z'
type	'STOP_LOSS_FILLED'
tradeId	1.0080e+10
instrument	'EUR_USD'
units	1000
side	'buy'
price	1.0834
pl	-0.4817
interest	0
accountBalance	366.0461

## 3F: GetTransactionInfo

### Syntax:

```
GetTransactionInfo('#####')
```

### Input:

The only input is the id for the required transaction in the form: '10080240226'

### Information:

This function returns all the available information for a specific transaction, this function can only return information on transactions made by the default account, though the default accountId can be edited in the 'api' variable in the workspace.

The screenshot shows the MATLAB Command Window with the command `>> GetTransactionInfo('10080159877')` and the output `ans =` followed by a struct array with the following fields:

```
id: '10080159877'  
accountId: 456977  
time: '2016-01-21T14:11:55.000000Z'  
type: 'STOP_LOSS_FILLED'  
tradeId: 1.0080e+10  
instrument: 'GBP_JPY'  
units: 1000  
side: 'buy'  
price: 164.8810  
pl: 4.9793  
interest: -5.0000e-04  
accountBalance: 366.5278
```

The Variables pane on the right shows `ans` as a 1x1 struct with 12 fields. A detailed view of the struct is shown in a table:

Field	Value
id	'10080159877'
accountId	456977
time	'2016-01-21T14:11:55.000000Z'
type	'STOP_LOSS_FILLED'
tradeId	1.0080e+10
instrument	'GBP_JPY'
units	1000
side	'buy'
price	164.8810
pl	4.9793
interest	-5.0000e-04
accountBalance	366.5278

## 3G: GetPrices

### Syntax:

GetPrices('PairString')

### Input:

The only input is the instrument name in the form: 'EUR\_USD'

### Information:

This function returns the bid and ask price for the selected instrument at the current time.

```
Command Window
>>
>> GetPrices('EUR_USD')

ans =

    instrument: 'EUR_USD'
         time: '2016-01-25T07:23:08.529732Z'
         bid: 1.0815
         ask: 1.0816

fx >> |
```

## 3H: StreamPrices

### Syntax:

StreamPrices('PairString','time','output')

E.G:

StreamPrices('EUR\_USD','100','plot')

StreamPrices('EUR\_USD','100','data')

### Input:

This function requires three inputs, if only the first two inputs are given the third will default to 'data'.

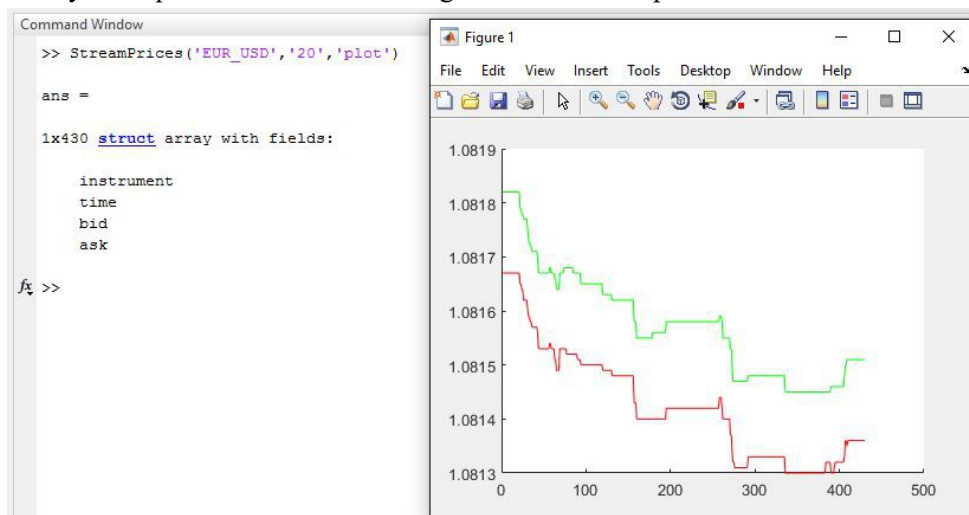
The name of the instrument to retrieve data for, format: 'EUR\_USD'

The length of time going forward to retrieve data in seconds, format: '100'

The output, either a matrix of prices or a matrix of prices and an animated live chart, format: 'data','plot'

### Information:

This function returns the bid and ask prices for a selected instrument in a stream going forward, the output can be either an animated chart or just the raw data. MatLab generally only performs one function at a time so this can't easily be used for a live bot to trade on, the algorithm would only be able to review the information at the end of the time selected. Instead call the GetPrices function with each iteration of the algorithm, this will increase efficiency as well because price information will only be requested at the time the algorithm is able to process it.



## 3I: GetHistory

### Syntax:

GetHistory('PairString','granularity','count')

E.G:

GetHistory('EUR\_USD','M5','100')

### Input:

Three inputs are required for this function.

The name of the instrument in the form: 'EUR\_USD'

The Granularity, this is the time range for the, open, close, high and low price fields in the returned data.

Format: 'M5'

More information on granularity options can be found here:

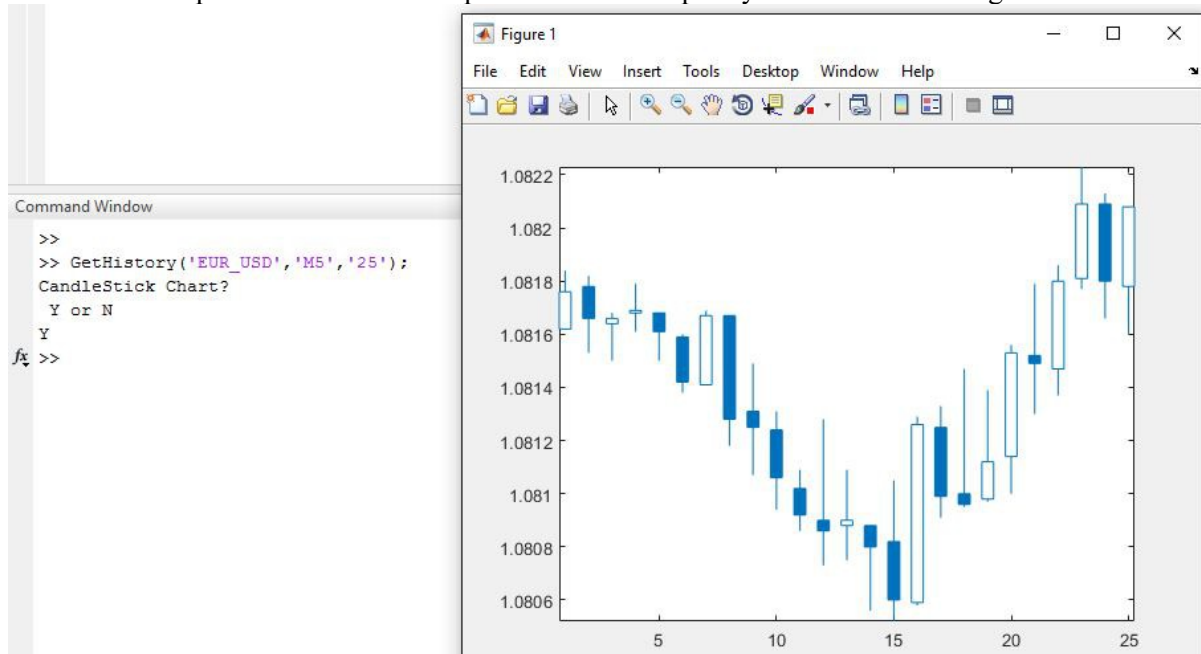
<http://developer.oanda.com/rest-live/rates/#retrieveInstrumentHistory>

The number of time spans of the selected granularity is the 'count' input, format: '100'

### Information:

This function returns the price history of a selected instrument and presents a candlestick chart of the returned data if the user accepts.

The bot version does not produce a chart and requires no further input by the user after calling the function.





## 3J: GetOrders

### Syntax:

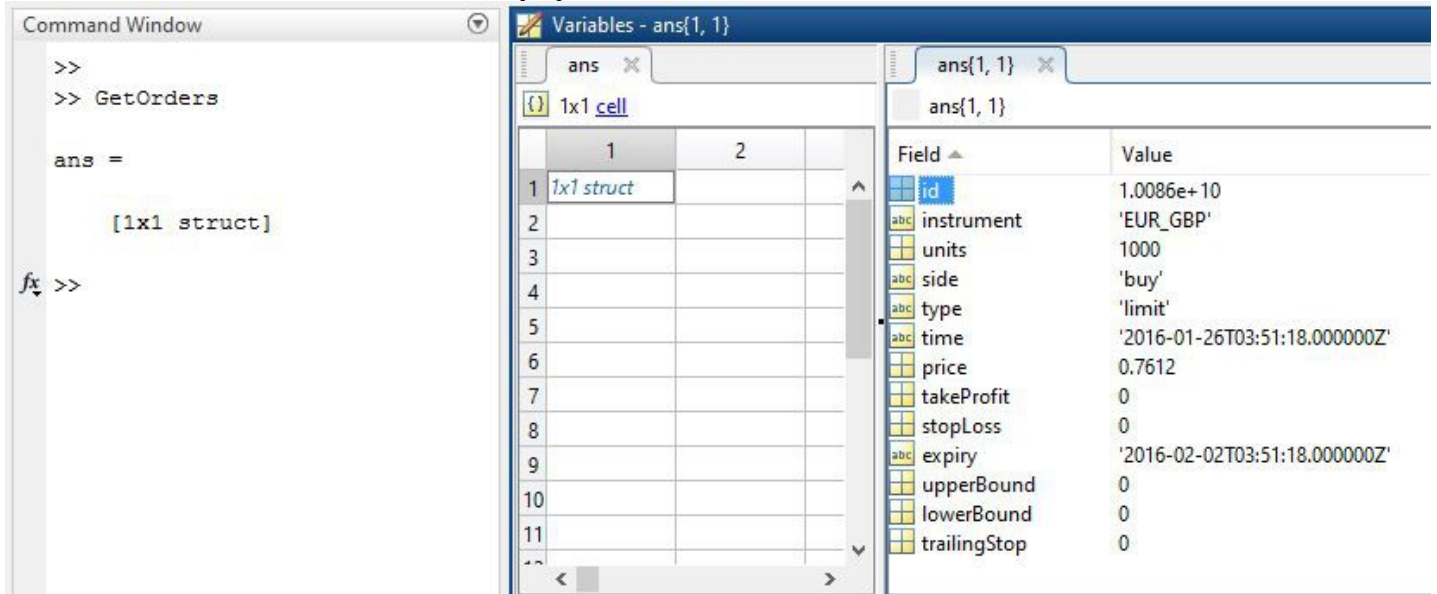
GetOrders

### Input:

This function has no input arguments.

### Information:

This function returns the information for any open orders for the default accountId.



The screenshot shows the MATLAB Command Window and Variables Editor. The Command Window displays the execution of the `GetOrders` function, which returns a 1x1 struct. The Variables Editor shows the structure of the returned object, `ans{1,1}`, which is a struct with the following fields and values:

Field	Value
<code>id</code>	<code>1.0086e+10</code>
<code>instrument</code>	<code>'EUR_GBP'</code>
<code>units</code>	<code>1000</code>
<code>side</code>	<code>'buy'</code>
<code>type</code>	<code>'limit'</code>
<code>time</code>	<code>'2016-01-26T03:51:18.000000Z'</code>
<code>price</code>	<code>0.7612</code>
<code>takeProfit</code>	<code>0</code>
<code>stopLoss</code>	<code>0</code>
<code>expiry</code>	<code>'2016-02-02T03:51:18.000000Z'</code>
<code>upperBound</code>	<code>0</code>
<code>lowerBound</code>	<code>0</code>
<code>trailingStop</code>	<code>0</code>

## 3K: GetTrades

### Syntax:

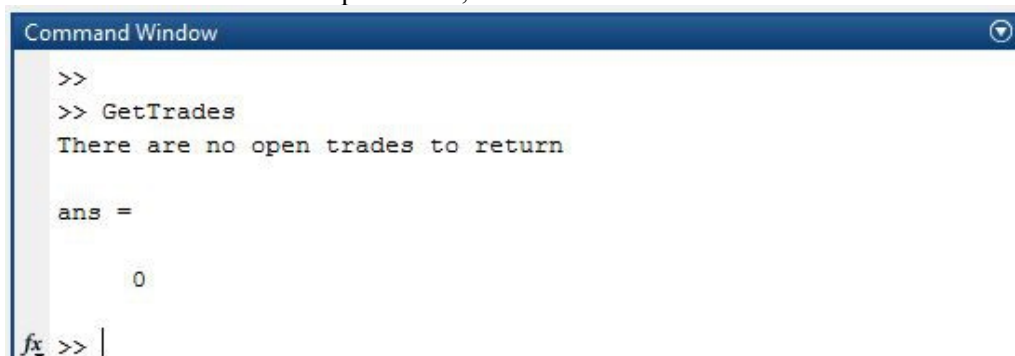
GetTrades

### Input:

This function has no input arguments

### Information:

This function returns the information for all open trades, if there are none it returns 0.



The screenshot shows the MATLAB Command Window with the following output:

```
>>  
>> GetTrades  
There are no open trades to return  
  
ans =  
  
0  
  
fx >> |
```



## 3L: NewOrder

### Syntax:

OrderBook = NewOrder('PairString','Units','Side','StopLoss','TakeProfit','TrailingStop')

### Input:

The instrument to open a new trade for: 'PairString'. Format : 'EUR\_USD'

The number of units: 'Units'. Format: '10'

The side of the trade: 'Side'. Format: 'Buy' , 'Sell'

The stop loss: 'StopLoss'. Format: '1.084'

The take profit: 'TakeProfit'. Format: '1.086'

The trailing stop in pips, up to one decimal place: 'TrailingStop'. Format: '22.7'

### Information:

This function opens new trades, opening orders that aren't at market is not currently supported but given that MatLab is capable of tracking the price an order can be stored in the MatLab script rather than on the Oanda server. I can see no reason why an algorithm would open a trade without considering the most recent information. The function will return information on the trade and I recommend storing this information in a variable called “OrderBook” so the function won't overwrite previous trade information.



The screenshot shows the MATLAB Command Window and the Variables window. The Command Window displays the execution of the `NewOrder` function, which returns a 1x1 struct named `OrderBook`. The Variables window shows the structure of `OrderBook{1, 1}`.

**Command Window:**

```
>>  
>> OrderBook = NewOrder('EUR_USD','10','buy','1.084','1.086','21.7')  
  
OrderBook =  
  
    [1x1 struct]  
  
fx >>
```

**Variables - OrderBook{1, 1}:**

Field	Value
instrument	'EUR_USD'
time	'2016-01-26T06:00:37.000000Z'
price	1.0845
tradeOpened	1x1 struct
tradesClosed	0x1 cell
tradeReduced	1x1 struct

## 3M: ModifyTrade

### Syntax:

ModifyTrade('tradeID','stopLoss','takeProfit','trailingStop')

Example:

ModifyTrade('10088074931','169.064',0,0)

### Input:

The id of the trade the user wants to modify: 'id'. Format: '10088074931'

The new stopLoss: 'stopLoss'. Format: '169.064' , 0 , '0'

The new takeProfit: 'takeProfit'. Format: '171.02' , 0 , '0'

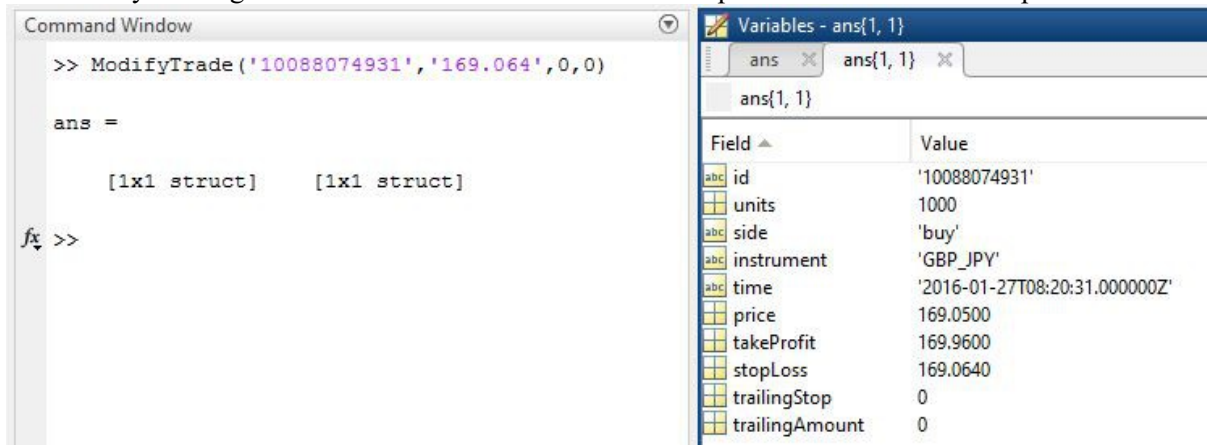
The new trailingStop in pips: 'trailingStop'. Format: '5.3' , 0 , '0'

### Information:

This function modifies currently open trades, trade IDs can be returned with the GetTrades function.

If you do not want to modify one of the three modifiable values enter either '0' or 0.

The function ends by running the GetTrades function and returns all open trades with their new parameters.



## 3N: CloseTrade

### Syntax:

CloseTrade('tradeID')

example:

CloseTrade('10088116913')

### Input:

The id of the trade the user wants to close: 'tradeID'. Format: '10088116913'

### Information:

This function just closes the selected trade.

