Connect RIT and MATLAB

This tutorial is organized in two sections:

- Streaming Data from Rotman Interactive Trader to MATLAB
- Trading with Rotman Interactive Trader

If you do not already have the Rotman Trader Toolbox installed, you will need to get it first to use the commands taught in this tutorial. Go to the MathWorks website and use the search bar to find “Rotman Trader Toolbox” in the File Exchange. Download and install the toolbox. Alternatively, click at the following link: http://www.mathworks.com/academia/student-competitions/rotman-trading/

Streaming Data from Rotman Interactive Trader to MATLAB

This example shows how to use the ‘rotmanTrader’ functions to connect to and trade through Rotman Interactive Trader (RIT). RIT must be installed on your computer along with the Excel RTD Links. The MATLAB version (32bit vs 64bit) has to be the same as your Excel version (32bit vs 64bit). For more information visit http://rit.rotman.utoronto.ca/

Create a Connection

The first step is to create a connection to RIT. To do this, issue the ‘rotmanTrader’ command. You can name your connection something other than rit.

```
rit = rotmanTrader
```

```
rit =

  rotmanTrader with properties:

           updateFreq: 2
           lastUpdate: '27-Nov-2015 09:18:43'
          updateTimer: [1x1 timer]
           updateFcns: {}
           traderName: 'Marco Salerno'
             traderID: 'Marco'
        timeRemaining: 300
               period: 1
             yearTime: 72000
            timeSpeed: 1
       allAssetTickers: ''
   allAssetTickerInfo: ''
           allTickers: {'ALGO'}
        allTickerInfo: {'ALGO'  'Shares of ALGO Corp.'  'STOCK'  '1'  '1'  '1'}
                   pl: 0
                 cash: 0
```

Notice that the ‘rit’ connection has properties listed. These default properties are always available and update with the frequency listed in the updateFreq property. The default value is 2 seconds. Also listed is the last update timestamp in lastUpdate. To change the update frequency, set the property to a different value. For example, to change it to 1 second, enter:


```
rit.updateFreq  =  1
```

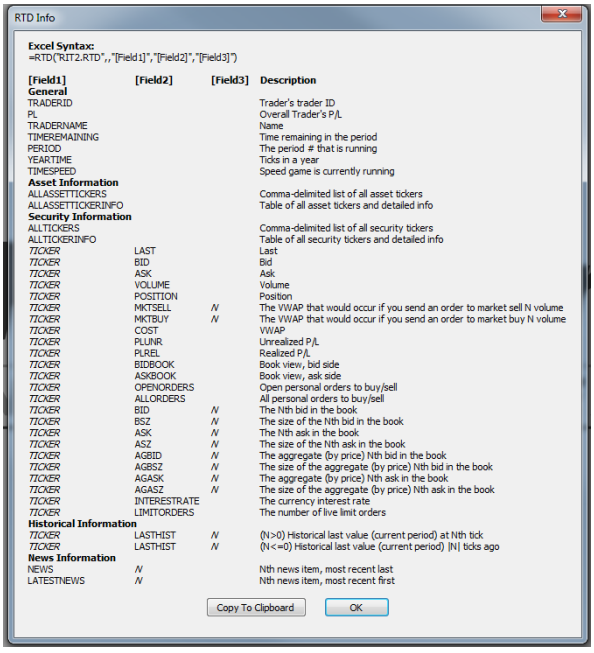
```
rit =

  rotmanTrader with properties:

          updateFreq: 1
          lastUpdate: '27-Nov-2015 09:31:53'
         updateTimer: [1x1 timer]
          updateFcns: {}
          traderName: 'Marco Salerno'
            traderID: 'Marco'
       timeRemaining: 300
              period: 1
            yearTime: 72000
           timeSpeed: 1
      allAssetTickers: ''
   allAssetTickerInfo: ''
           allTickers: {'ALGO'}
       allTickerInfo: {'ALGO'  'Shares of ALGO Corp.'  'STOCK'  '1'  '1'  '1'}
                  pl: 0
                cash: 0
```

Subscribing to RIT Data

Data is retrieved from RIT using the same server that is used for Microsoft Excel. In RIT, you can click on the RTD link in the bottom right corner. It will bring up this image with available data fields.



In Excel, the RTD function is used to return data. In MATLAB, the subscribe command is used to enter the field information for the data you wish to subscribe to. This will add the data to the rit variable we created earlier. To get the last traded price we need to enter two fields, the ticker symbol and the LAST

string separated by a |. For example, to subscribe to security ALGO and add the last price to our connection to RIT (the rit variable defined earlier), type:

```
subscribe(rit, 'ALGO|LAST')
rit
```

```
rit =

  rotmanTrader with properties:

           updateFreq: 1
           lastUpdate: '27-Nov-2015 09:41:50'
          updateTimer: [1x1 timer]
           updateFcns: {}
           traderName: 'Marco Salerno'
             traderID: 'Marco'
        timeRemaining: 283
               period: 1
             yearTime: 72000
            timeSpeed: 1
       allAssetTickers: ''
    allAssetTickerInfo: ''
           allTickers: {'ALGO'}
         allTickerInfo: {'ALGO'  'Shares of ALGO Corp.'  'STOCK'  '1'  '1'  '1'}
                    pl: 0
                  cash: 0
             algo_last: 20.0100
```

You can see that RIT now has a new property of algo_last that will update with last traded prices. Subscriptions added will show up as additional properties. To return the data, simply type:

```
rit.algo_last
```

```
ans =

   20.0100
```

Note that when subscribe is called, data is updated from RIT. You can also force an update by issuing update.

```
update(rit)
```

```
ans =

  rotmanTrader with properties:

           updateFreq: 1
           lastUpdate: '27-Nov-2015 09:43:09'
          updateTimer: [1x1 timer]
           updateFcns: {}
           traderName: 'Marco Salerno'
```



```
            traderID: 'Marco'
         timeRemaining: 283
                period: 1
              yearTime: 72000
             timeSpeed: 1
        allAssetTickers: ''
     allAssetTickerInfo: ''
           allTickers: {'ALGO'}
         allTickerInfo: {'ALGO'  'Shares of ALGO Corp.'  'STOCK'  '1'  '1'  '1'}
                   pl: 0
                 cash: 0
             algo_last: 20.0100
```

We could also add the bid and ask as well. Note the need to separate the list of two subscriptions by ";" is required.

```
subscribe(rit, {'ALGO|BID';'ALGO|ASK'})
rit
```

```
rit =

  rotmanTrader with properties:

            updateFreq: 1
            lastUpdate: '27-Nov-2015 09:46:00'
           updateTimer: [1x1 timer]
            updateFcns: {}
            traderName: 'Marco Salerno'
              traderID: 'Marco'
         timeRemaining: 283
                period: 1
              yearTime: 72000
             timeSpeed: 1
        allAssetTickers: ''
     allAssetTickerInfo: ''
           allTickers: {'ALGO'}
         allTickerInfo: {'ALGO'  'Shares of ALGO Corp.'  'STOCK'  '1'  '1'  '1'}
                   pl: 0
                 cash: 0
              algo_bid: 20.0100
              algo_ask: 20.0200
             algo_last: 20.0100
```

Working with Streaming Data

To work with streaming data, you add a function that is called each time data is updated. To do this, let's first create a function that will display the last price for ALGO.

```
fcn = @(input) disp(['ALGO Last Traded at $', num2str(input.algo_last, '%4.2f')]);
```


What we created here is a function that will print to the command window the last traded price for ALGO every time an update is called for 'rotmanTrader' (every second in this case). The input in this case is the 'rotmanTrader' variable. For example, test the function:

```
fcn(rit)
```

```
ALGO Last Traded at $20.01
```

Now add it to the list of updateFcns and it will be executed every time there is an update.

```
addUpdateFcn(rit,fcn)
rit
```

```
rit =

  rotmanTrader with properties:

            updateFreq: 1
            lastUpdate: '27-Nov-2015 10:00:51'
           updateTimer: [1x1 timer]
            updateFcns: {}
            traderName: 'Marco Salerno'
              traderID: 'Marco'
         timeRemaining: 253
                period: 1
              yearTime: 72000
             timeSpeed: 1
       allAssetTickers: ''
    allAssetTickerInfo: ''
           allTickers: {'ALGO'}
         allTickerInfo: {'ALGO'  'Shares of ALGO Corp.'  'STOCK'  '1'  '1'  '1'}
                    pl: 0
                  cash: 0
             algo_last: 20.0600
              algo_ask: 20.0600
              algo_bid: 20

ALGO Last Traded at $20.06
ALGO Last Traded at $20.06
ALGO Last Traded at $20.06
ALGO Last Traded at $20.05
ALGO Last Traded at $20.05
ALGO Last Traded at $20.04
ALGO Last Traded at $20.03
ALGO Last Traded at $20.03
ALGO Last Traded at $20.04
```

Add two more functions for updates on bids and asks.

```
askfcn = @(input) disp(['ALGO ASK Price is $',num2str(input.algo_ask,'%4.2f')]) ;
addUpdateFcn(rit,askfcn) ;
bidfcn = @(input) disp(['ALGO BID Price is $',num2str(input.algo_bid,'%4.2f')]) ;
```



```
addUpdateFcn(rit,bidfcn) ;
```

```
ALGO BID Price is $20.03
ALGO ASK Price is $20.04
ALGO Last Traded at $20.04
ALGO BID Price is $20.03
ALGO ASK Price is $20.05
ALGO Last Traded at $20.05
```

Note that updateFcns is a 1x3 cell array listing the functions that will be executed with each update.

```
rit.updateFcns{:}
```

```
ans =

    '@(input)disp(['ALGO ASK Price is $',num2str(input.algo_ask...'
    '@(input)disp(['ALGO BID Price is $',num2str(input.algo_bid...'
    '@(input)disp(['ALGO Last Traded at $',num2str(input.algo_l...'
```

To stop the updates, simply remove the function from the updateFcns array using the removeUpdateFcn and pass in function name to remove. We'll remove the Ask price updates.

```
removeUpdateFcn(rit,rit.updateFcns{1})
```

Now remove the bid and last functions too.

```
removeUpdateFcn(rit,rit.updateFcns{2})
removeUpdateFcn(rit,rit.updateFcns{1})
```

Unsubscribing

To unsubscribe from a source of data, use unsubscribe. Note that if you have any update functions that are using this data, you need to remove them first.

Unsubscribe from the bid price for ALGO. First, get the subscription list and IDs.

```
topics = getSubscriptions(rit)
```

```
topics =

    ID       Topic
    __    ___________

    1     'ALGO|LAST'
    2     'ALGO|BID'
    3     'ALGO|ASK'
```

Use the topic ID to specify which subscription to remove. To remove the Bid updates, type:

```
unsubscribe(rit,2)
rit
```



```
rit =

  rotmanTrader with properties:

           updateFreq: 1
           lastUpdate: '27-Nov-2015 10:46:14'
          updateTimer: [1x1 timer]
           updateFcns: {1x0 cell}
           traderName: 'Marco Salerno'
             traderID: 'Marco'
        timeRemaining: 283
               period: 1
             yearTime: 72000
            timeSpeed: 1
       allAssetTickers: ''
    allAssetTickerInfo: ''
           allTickers: {'ALGO'}
         allTickerInfo: {1x6 cell}
                   pl: 0
                 cash: 0
            algo_last: 20.0400
             algo_ask: 20.0400
```

The data is no longer retrieved and is removed from the RIT variable. The other subscriptions are still kept.

Cleaning Up

To properly clean up, you first need to delete the 'rotmanTrader' connection before clearing it from the workspace. This stops the updates and disconnects from Rotman Interactive Trader.

```
delete(rit)
clear rit
```

If you cleared the rit variable before issuing the delete, the update timer is still running in the background, and you may see errors/warnings. To stop it issue the following command:

```
delete(timerfind('Name','RotmanTrader'));
```


Trading with Rotman Interactive Trader

This example shows how to use the 'rotmanTrader' functions to connect to and trade through Rotman Interactive Trader (RIT). RIT must be installed on your computer along with the Excel RTD Links. For more information visit http://rit.rotman.utoronto.ca/software.asp.

Create a Connection

First create a connection to Rotman Interactive Trader and list the functions available.

```
rit = rotmanTrader; methods(rit)
```

```
Methods for class rotmanTrader:

addOrder           delete             rotmanTrader
addUpdateFcn       getOrderInfo       sell
addprop            getOrders          stopUpdates
blotterOrder       getSubscriptions   subscribe
buy                getTickerInfo      unsubscribe
cancelOrder        isOrderQueued      update
cancelOrderExpr    limitOrder
cancelQueuedOrder  removeUpdateFcn
clearQueuedOrders  restartTimer
```

To get more information on the functions, type help or doc followed by the name of the function. For example:

```
help buy
```

```
--- help for rotmanTrader/buy ---

 buy submits a market buy order to Rotman Interactive Trader.

    ID = buy(RIT,TICKER,SIZE) returns queued order ID if market
    order was successfully submitted.  RIT is the
    connection to Rotman interactive Trader.  TICKER is the
    symbol(s) as a string or cell array of strings for the
    tickers to trade.  SIZE is the quantity to buy at market.

    Example:
    rit = rotmanTrader;
    buyID = buy(rit,'CRZY',100)
    sellID = buy(rit,'TAME',-100) % negative is sell

    See also sell, limitOrder, addOrder, blotterOrder

    Published output in the Help browser
       showdemo rotmanTrader
```

Submitting Market Orders

Buy and sell market order for a single security. For both buy and sell functions, the returned value is the orderID.

```
buyID = buy(rit, 'ALGO', 10)
sellID = buy(rit, 'ALGO', 10)
```

```
buyID =

     1

sellID =

     2
```

The type of order can also be changed by changing the sign of qty. For example, submitting a buy order with a quantity of -10 changes it to a sell order.

```
buyID = buy(rit, 'ALGO', -10);
sellID =sell(rit, 'ALGO', -10);
```

Submitting Limit Orders

Limit orders can be submitted using the limitOrder function.

```
help limitOrder
```

```
help limitOrder
 --- help for rotmanTrader/limitOrder ---

 limitOrder submits a limit order to Rotman Interactive Trader.

    ID = limitOrder(RIT,TICKER,QTY,PRICE) submits a market buy
    or sell order depending upon the sign of QTY and returns
    queued ID when order is accepted.  False (0) if not
    accepted, and -1 if the case is not running.    RIT is the
    connection to Rotman interactive Trader.  TICKER is the
    symbol(s) as a string or cell array of strings for the
    securities to trade.  QTY is the quantity to submit for bid
    (buy) or ask (sell).  Price is the bid/ask price to offer.

    QTY defines the limit order as a buy limit order if
    positive.  If QTY is negative, submits a sell limit order.
```

To submit a buy limit order, a bid for ALGO at a price of $20.00 and quantity 90:

```
buyID = limitOrder(rit, 'ALGO', 90, 20.00);
```

To submit a sell limit order, an ask for ALGO at a price of $15.00 and quantity 100 (Note the (-) negative quantity used to denote a sell limit order):

```
sellID = limitOrder(rit, 'ALGO', 90, 21.00);
```


Submit Orders Using a Blotter

Create an order blotter, a table with order information.

```
help blotterOrder
```

```
 --- help for rotmanTrader/blotterOrder ---

 blotterOrder submits orders using an order blotter to Rotman
 Interactive Trader.

    ID = blotterOrder(RIT,BLOTTER) submits orders to Rotman
    Interactive Trader, through connection RIT.  BLOTTER is the
    order table specifying orders to place.

    For market orders, BLOTTER must contain the variables names
    TICKER, QUANTITY, and ACTION (with Buy/Sell values).

    For limit orders, BLOTTER must contain the variable names
    TICKER, QUANTITY, ACTION (with Buy/Sell values) and PRICE
    of the limit order.  To submit a market order with limit
    orders, PRICE must be set to 0 or NaN for the market
    orders.  Otherwise the will be submitted as market orders.
```

Create a blotter of buy and sells at the market price with:

```
ticker = {'ALGO'; 'ALGO'};
action ={'BUY'; 'SELL'};
quantity = [120; 120];
blotter = table(ticker, action, quantity)
```

```
blotter =

    ticker        action          quantity
    ______    _____________    __________

    'ALGO'    'BUY'    'SELL'    120    120
```

Submit the blotter order using:

```
blotterID = blotterOrder(rit,blotter)
```

```
blotterID =

     7     8
```

Blotter orders can also contain limit orders. Prices must be present to define a limit order. Use 0 or nan for market orders in prices.

```
price = [nan, 21.00];
blotter = table(ticker, action, quantity, price)
```

```
blotter =
```



```
    ticker    action    quantity    price
    ______    ______    ________    _____

    'ALGO'    'BUY'     120         NaN
    'ALGO'    'SELL'    120          21
```

Submit the blotter order.

```
tf = blotterOrder(rit, blotter)
```

```
tf =

     9
    10
```

```
id = getOrders(rit)
```

```
id =

    71
```

```
orderBlotter = getOrderInfo(rit,id)
```

```
orderBlotter =

    OrderID    Ticker    Type    OrderType    Quantity    Price    Status    Quantity2
    _______    ______    ____    _________    ________    _____    ______    _________

    71         ALGO      SELL    LIMIT        -120        21       LIVE      120
```

Cancelling Orders

Cancel an order by orderID.

```
cancelID = id(1);
cancelOrder(rit,cancelID)
```

Cancel an order by expression.

```
expr = 'Price <= 15.00 AND ticker = ALGO;
cancelOrderExpr(rit,expr)
```

Cancel Queued Orders

Orders are submitted to Rotman Interactive Trader and may be queued if the orders are submitted faster than the case allows. The queued orders can be queried and even deleted. Resubmit the blotter order from above, query which ones are still queued, and then cancel them.

```
blotter
```



```
blotter =

    ticker    action    quantity    price
    ______    ______    ________    _____

    'ALGO'    'BUY'     120         NaN
    'ALGO'    'SELL'    120          21
    'ALGO'    'BUY'     120          15
```

```
queuedID = blotterOrder(rit,blotter)
```

```
queuedID =

    16
    17
    18
```

```
inQueue = isOrderQueued(rit,queuedID)
```

```
inQueue =

     0
     1
     1
```

1 indicates that there are queued orders. Calling the cancelQueuedOrder function and passing in the series of 0s and 1s that we saved inQueue will remove queued orders. We can check by using isOrderQueued.

```
cancelQueuedOrder(rit,queuedID(inQueue))
isOrderQueued(rit,queuedID)
```

```
ans =

     0
     0
     0
```

```
id = getOrders(rit)
```

```
id =

   738   736
```

```
orderBlotter = getOrderInfo(rit,id)
```

```
orderBlotter =

    OrderID    Ticker     Type     OrderType    Quantity    Price    Status    Quantity2
    _______    ______    ______    _________    ________    _____    ______    _________
```

```
    738        'ALGO'    'BUY'     'LIMIT'       120        15       'LIVE'    120
    736        'ALGO'    'SELL'    'LIMIT'      -120        21       'LIVE'    120
```

One can also clear all queued orders using clearQueuedOrder.

```
blotter
queuedID = blotterOrder(rit,blotter)
inQueue = isOrderQueued(rit,queuedID)
clearQueuedOrders(rit)
isOrderQueued(rit,queuedID)
id = getOrders(rit)
orderBlotter = getOrderInfo(rit,id)
```

