



RIT VBA API Tutorial

Table of Contents

Introduction.....	2
Introduction to Excel VBA (Developer).....	3
VBA API Commands for RIT	10
VBA API Initialization	11
Algorithmic Trading Example - Arbitrage	21

Introduction

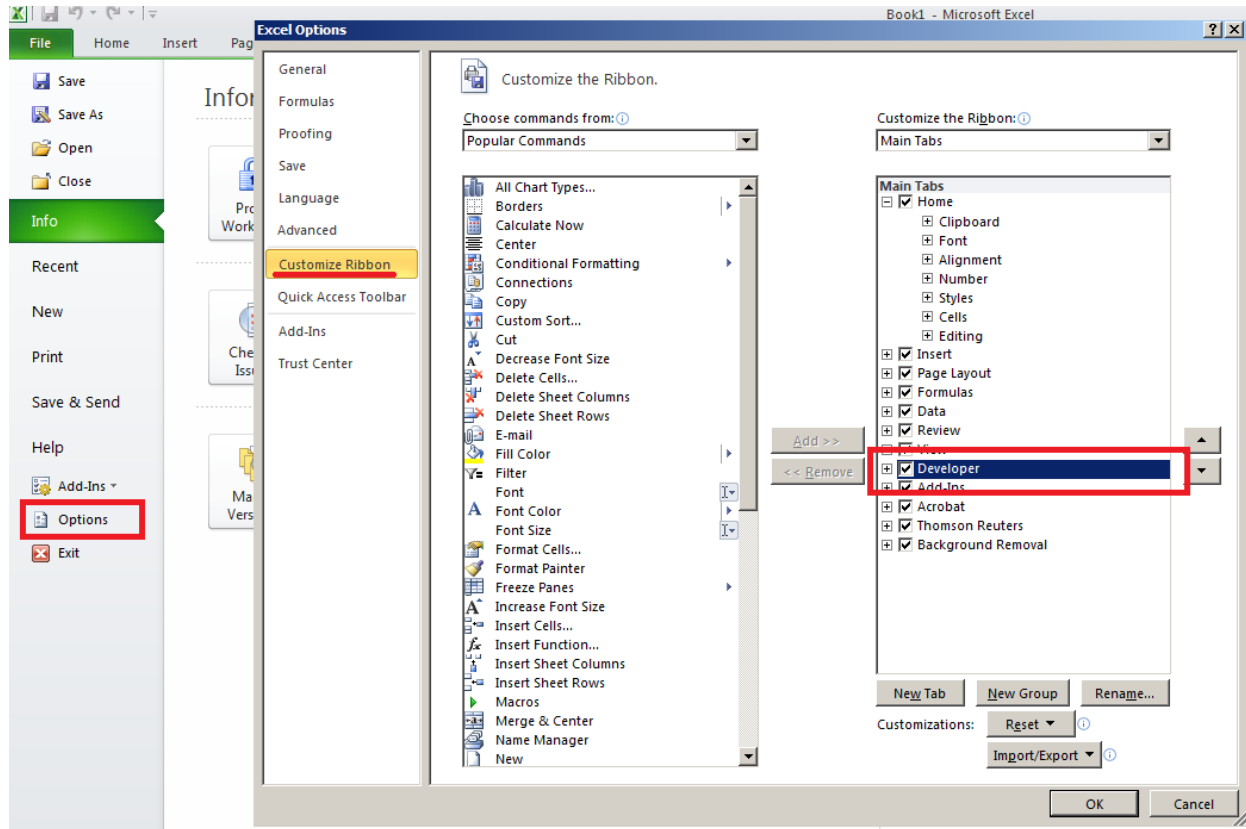
The Rotman Interactive Trader allows users to query for market data and submit trading instructions through a Microsoft Excel Visual Basic for Applications (VBA) API as well as through a REST API. The purpose of this is to allow for program “algorithmic” trading, where the computer executes trades based on a pre-defined set of instructions or parameters.

This tutorial document focuses on interacting with the Excel VBA API and assumes that the user has no previous knowledge of VBA, and begins by discussing the concepts of programming before in-depth trading algorithms are introduced. Those who are already familiar with VBA should skip to the section entitled “VBA API commands for RIT”.

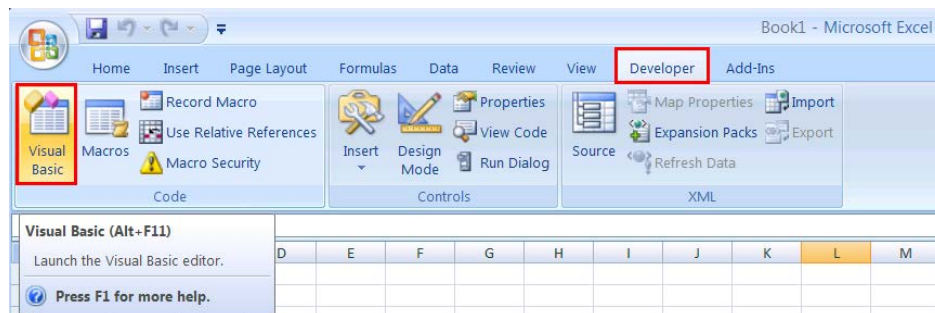
This document also does not discuss the strategies behind algorithmic trading. Rather, it introduces the user to the tools that are available through the RIT Excel VBA API. Users are encouraged to explore possible strategies and techniques and use the building blocks here to implement them.

Introduction to Excel VBA (Developer)

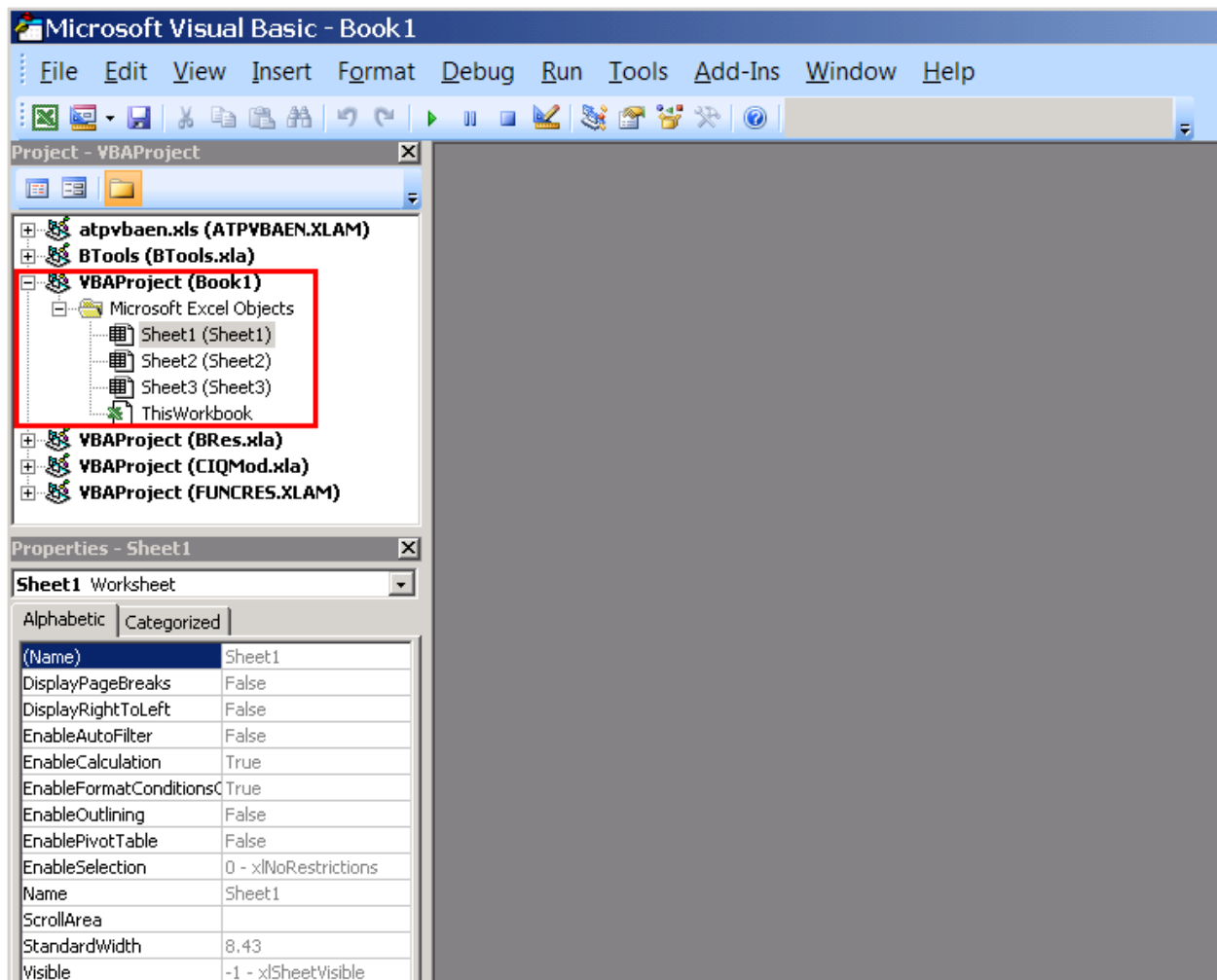
To access the VBA editor in Excel, first ensure that it is turned on by clicking on “File” on the top-left corner of the screen, then click on “Options”. Once the “Excel Options” window is opened, choose “Customize Ribbon” on the left menu bar, and ensure that “Developer” on the right side is checked. Once this is checked, the Developer Tab will appear in the original list of Excel tabs.



You can access the VBA editor by clicking on the “Visual Basic” icon within the Developer tab.
Hint: You can access this at anytime with the shortcut Alt+F11



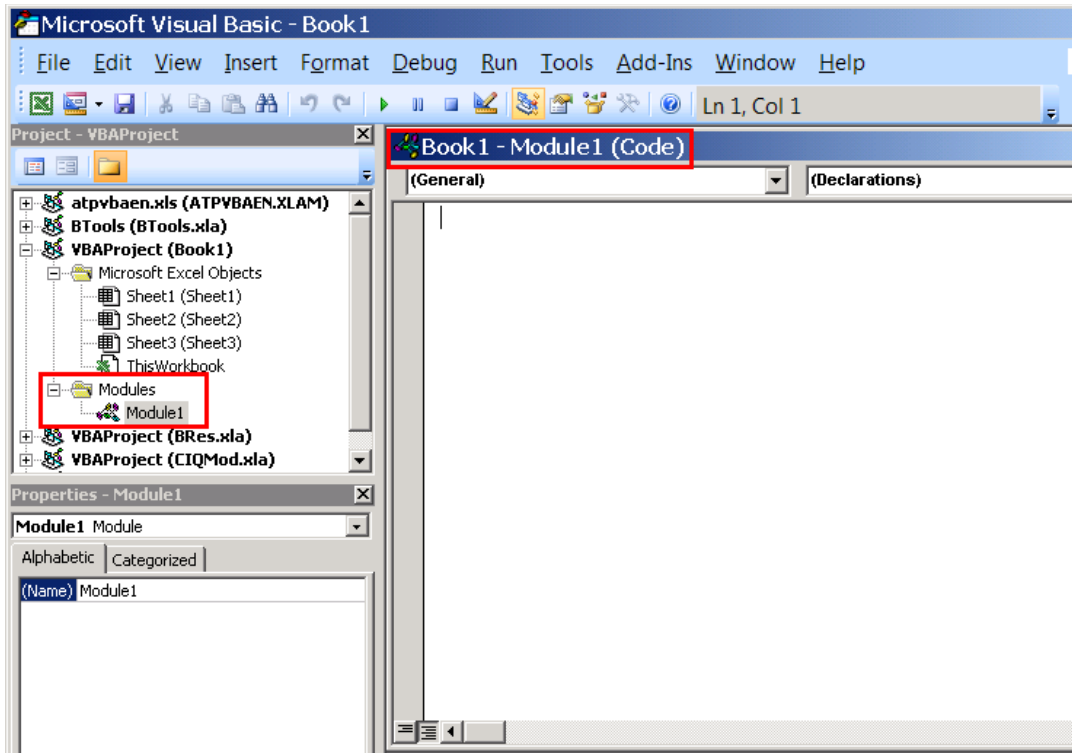
The VBA editor will display all of the loaded Excel projects and add-ins. What is relevant is the VBAProject (Book1) that you are currently working on. Note: Book1 refers to the name of your excel spreadsheet file and will change as you change your filename.



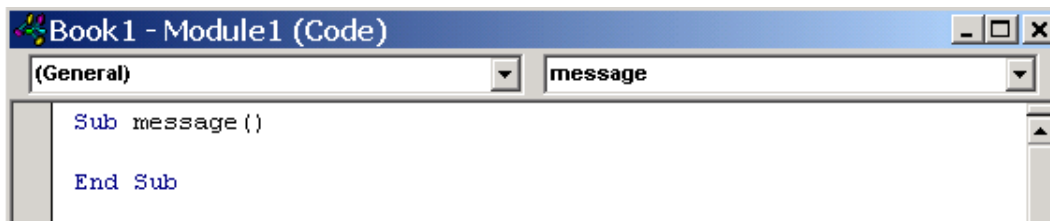
We will begin by writing some basic procedures in your Book1.xls. In order to do this, create a module in your book by going to **Insert -> Module**.

Module1 will be added to your Book1 project and a code window will open on the right hand side allowing you to input your programming code.

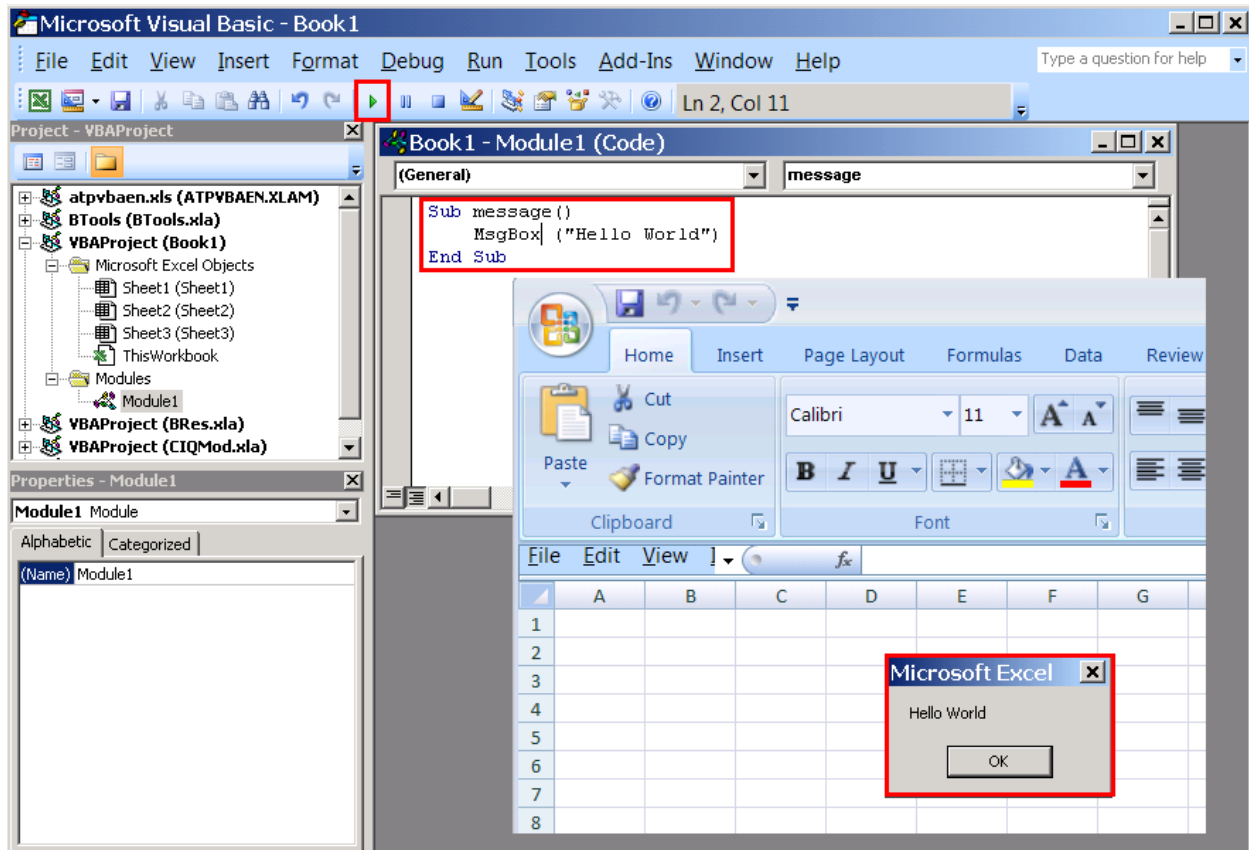
The first step is to write a very simple procedure. A procedure is a set of programming lines that are run by the computer whenever instructed to do so. Procedures are defined with the lines “sub <procedure>” and “end sub” enclosing them. We will define a procedure named “message” by inputting “Sub message” into the code window. As soon as you type “Sub message” (without quotes) and press enter, VBA will automatically format the text by adding brackets after message and add “End Sub” to the next line.



We have just created a procedure called “message”. When this procedure is run, it will execute the code. In this case, it will do nothing since we have not written any code between the beginning of the procedure (sub) and end of the procedure (end sub).

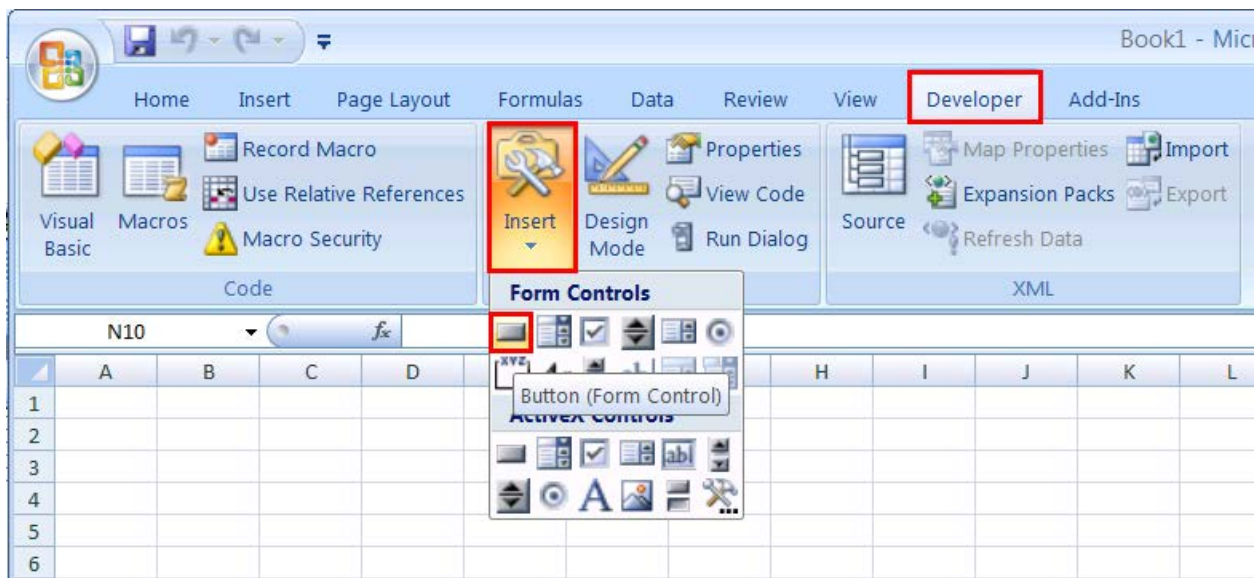


We will start with a basic set of code that references the built-in VBA function “MsgBox”. To do this, type “MsgBox (“Hello World”)” into the code window between your (Sub) and (end sub). The “MsgBox” command will cause a pop-up message box to show up in Excel when the code is executed. After you have typed the code into the window, click on the “Play” button in the VBA editor, your code will execute and a pop-up message in Excel should appear.

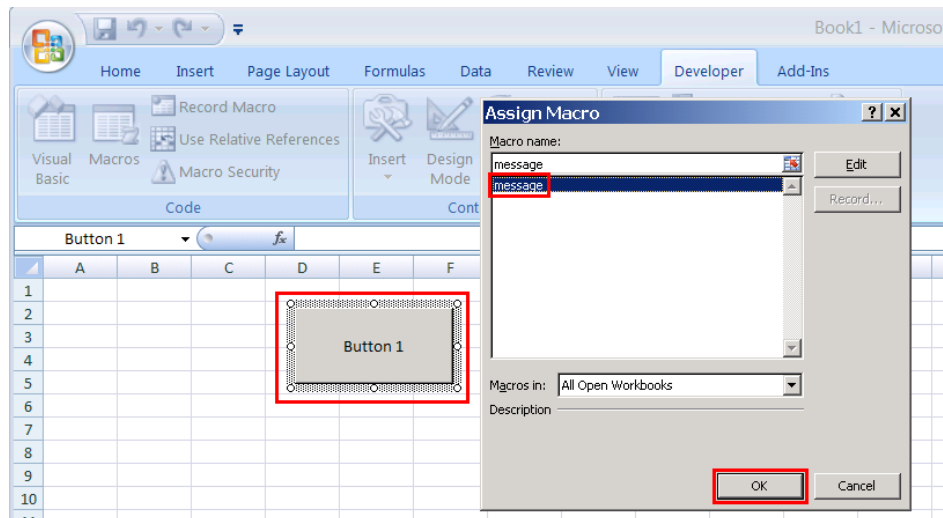


You have just completed writing and running a procedure in VBA. Obviously running the procedure from the VBA editor is rather cumbersome, so the next step involves linking the macro to an Excel button so that it is easier to run the procedure.

To create the Macro button, go back to the Developer tab in Excel and click on Insert, and then select the first option "Button".



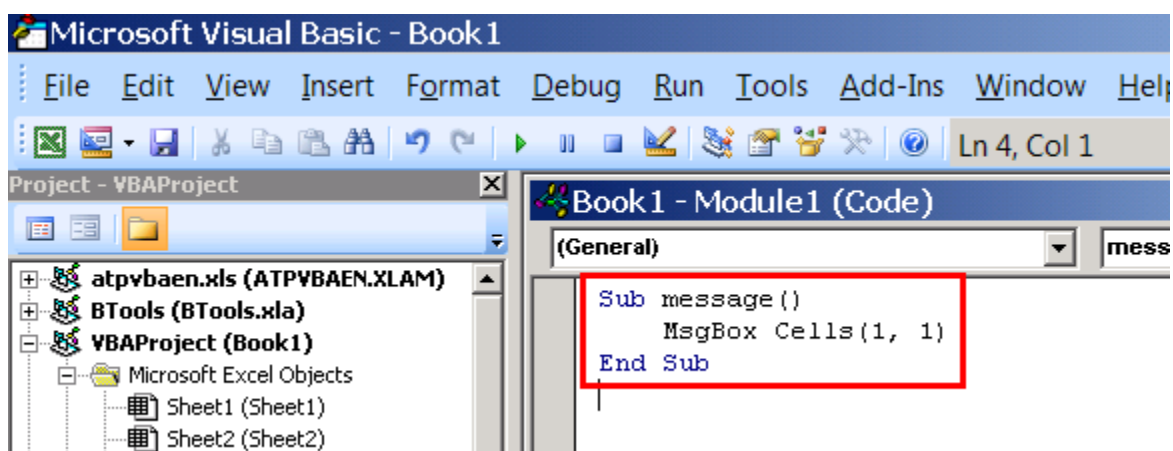
When you move your mouse over the spreadsheet, the mouse cursor will become a crosshair instead of an arrow. Click and drag anywhere on the spreadsheet to draw the button. Once you finish drawing the button, the “Assign Macro” form will appear, select “message” (the name of your macro you just wrote), then click OK. Now that you have assigned the procedure “message” to the button, the procedure will be executed each time you click the button. *Note: If you change the name of your procedure, do not forget to re-assign your Macro. In order to re-assign the macro, you will only need to right click on the button and then select “Assign Macro”*



Once that is complete, left-click on the button and your “Hello World” message box should appear. If you ever want to edit this object (resize, redirect, etc.) right click on it and a context menu will appear allowing you adjust the box.

To understand a little bit more behind the programming, we will revisit the code and modify it to be slightly more complex. In the Visual Basic Editor, we are going to modify the code to read “MsgBox Cells(1,1)” instead of “MsgBox (“Hello World”)”.

Much like Microsoft Excel, VBA assumes that any text wrapped in “quotes” is plain text, whereas anything not wrapped in “quotes” is a function, procedure, or operation. Since there are no quotes around “Cells(1,1)”, it will not say “Hello Cells(1,1)”, instead, it will follow the command of Cells(1,1).



The Cells(x,y) command is a function in Excel that instructs VBA to replace itself with the data from the spreadsheet row x, column y. Essentially the way VBA interprets this set of code is:

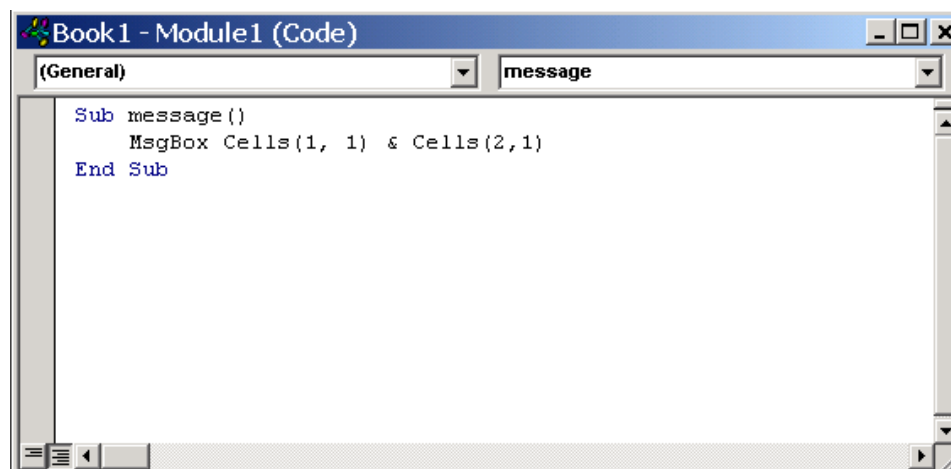
MsgBox("x") means "Create a message box with the text x"

Replacing ("x") with Cells(1,1) means we will use the data from the cell located in row 1, column 1.

MsgBox Cells(1,1) means "Create a message box with the data from row 1, column 1"

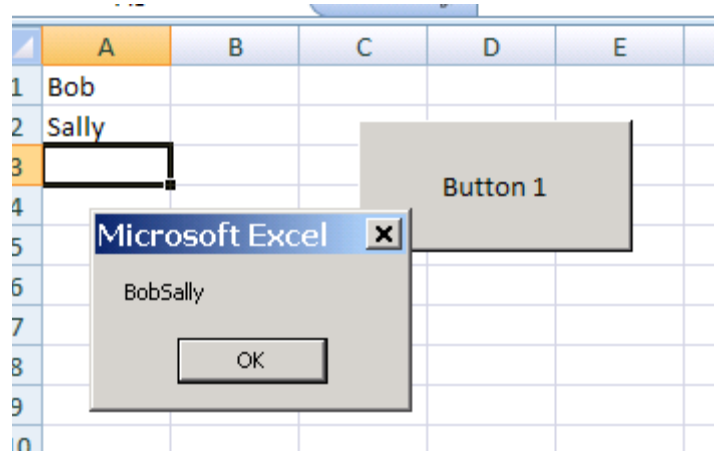
Now go to the Cell A1 in the current Excel Sheet1 and type in "Bob". Click on your Macro button, the result should be a message box that says "Bob". *Hint: If you want to reference cells from other sheets, you can do this by typing Sheet3.Cells(1,1). This will now use the data from cell A1 on Sheet3.*

We can make this more complex by adding an equation into the procedure. Go back to the VBA editor and change your code to the following:



```
Book1 - Module1 (Code)
(General) message
Sub message ()
  MsgBox Cells(1, 1) & Cells(2,1)
End Sub
```

Go to your Excel Sheet and type "Sally" into Cell A2, and click your macro button. The result should be:



To clean this up a little bit, we will make another adjustment to the code by adding the word "and" between the two references. This is accomplished as follows:


```

Book1 - Module1 (Code)
(General) message
Sub message ()
    MsgBox Cells(1, 1) & " and " & Cells(2, 1)
End Sub

```

Notice the quotes around the word “and”, as well as the space between the quotes and the word “ and ”. Without the spaces, the message box would simply say “BobandSally”. Alternatively without the “quotes” around <and>, VBA would think “and” is a command instead of using it as “text”.

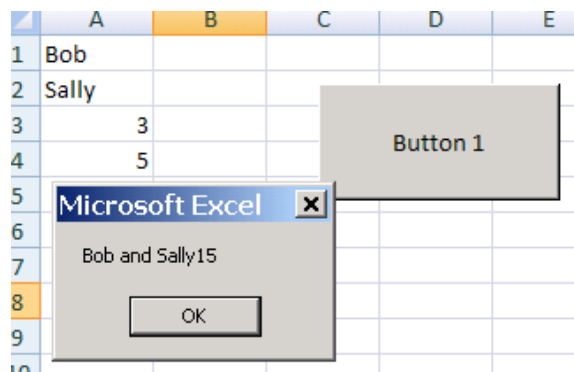
The last code adjustment that we will make is to add a mathematical equation to our message box. This is accomplished as follows:

```

Book1 - Module1 (Code)
(General) message
Sub message ()
    MsgBox Cells(1, 1) & " and " & Cells(2, 1) & Cells(3, 1) * Cells(4, 1)
End Sub

```

Type the values “3” and “5” into cells A3 and A4 and run your procedure by clicking the button. The result should be “Bob and Sally15”. Since we used the asterisk “*” between Cells(3,1) and Cells(4,1), VBA is instructed to multiply the values from these two cells, and then append them as text to the rest of the text.



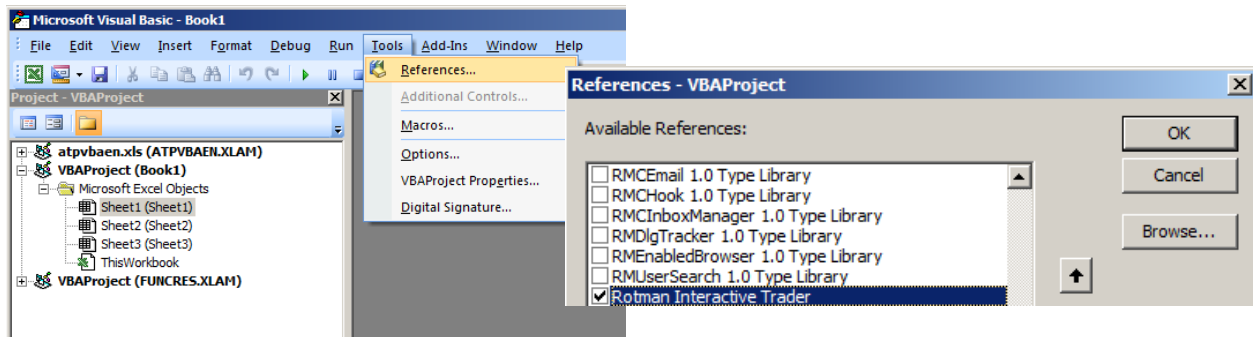
This concludes the basic VBA training that you will need in order to access the RIT API. You are now able to write a simple set of instructions (a procedure) in VBA using a predesigned function (MsgBox) and execute it via the Button that was created. In the next section, you will use the skills that you have learned, and apply them to trading!

VBA API Commands for RIT

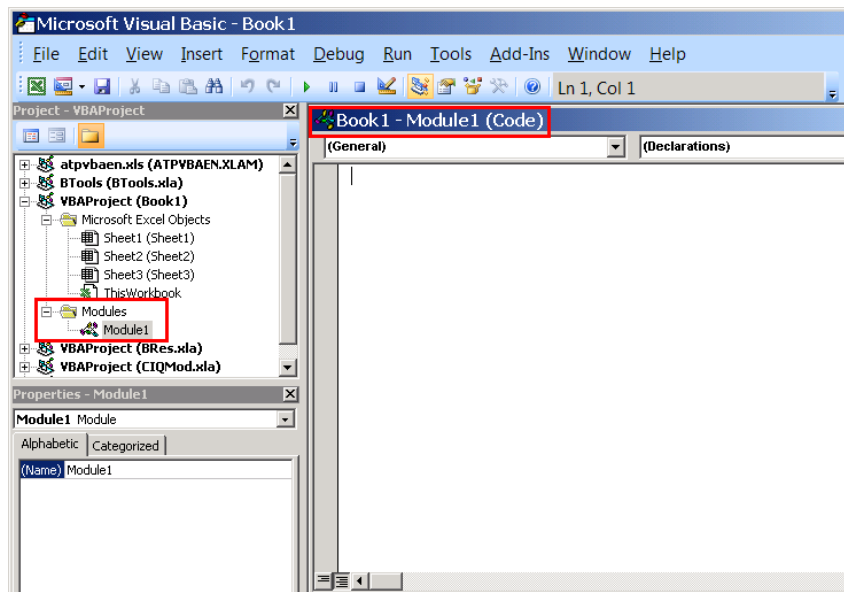
Setting up RIT VBA API configuration

Application Programming Interface (API) commands in Excel VBA can both retrieve information from and perform actions on the Rotman Interactive Trader (RIT).

To begin, start with a NEW spreadsheet and access VBA. In order to access RIT's built-in VBA commands, you will need to add it as a reference to your VBA project by going to: **Tools -> References**



When the Reference window appears, scroll down and check the item "Rotman Interactive Trader". This step loads the Rotman commands and functions into VBA so that you can reference them. Next, create a module in your file by going to **Insert -> Module**.



VBA API Initialization

Then, initialize a new Rotman Interactive Trader API object using the following code:

```
Dim API As RIT2.API
Set API = New RIT2.API
```

Once the RIT API object is initialized, you can start writing API commands. In general, the syntax for an API command is made up of 3 main parts: the object, the method, and the parameter(s) (optional), as demonstrated in the following sample code:

API.CancelOrder (order_id)
↑ ↑ ↑
Object Method Parameter

In this example, *API* is the **object** that actions are performed on. The **method**, *CancelOrder*, is the action to perform on *API* (in this case, the action is to cancel an order). The **parameter**, *order_id*, specifies details of the action (here, it specifies the order ID of the particular order to cancel).

Depending on the action that a method performs, it may or may not require a parameter. In the example above, *API.CancelOrder* requires a parameter to specify which order to cancel. In the following sections you will see examples of methods which do not require a parameter. These methods perform general actions. There are also examples demonstrating the use of more than one parameter, separated by a comma.

Other than performing actions, methods can also return a result (called the return value). It can be stored in a variable or a cell in an Excel worksheet for later reference. The example *API.CancelOrder* does not have a return value.

Submitting an Order

The following command adds an order to RIT.

General command Syntax:

API.AddOrder(*ticker, size of trade, price of trade, buy/sell, lmt/mkt*)

Parameters:

Parameter	Description	Possible Values
ticker	Ticker symbol of a stock	"ALGO", "CRZY", Range("A1"), etc.
size of trade	Bid size or ask size	500, 10, Range("A1"), Cells(2,3), etc.
price of trade	Bid price or ask price*	10.00, 15.25, Range("A1"), Cells(3, 4), etc.
buy/sell	Buy or sell an order	Buy order: API.BUY or 1** Sell order: API.SELL or -1**
lmt/mkt	Type of an order	Limit orders: API.LMT or 1** Market orders: API.MKT or 0**

* When inputting a market order, the price of trade must be specified with an arbitrary number. This number will be ignored as all market orders transact at the market price. See example in sample code 2.

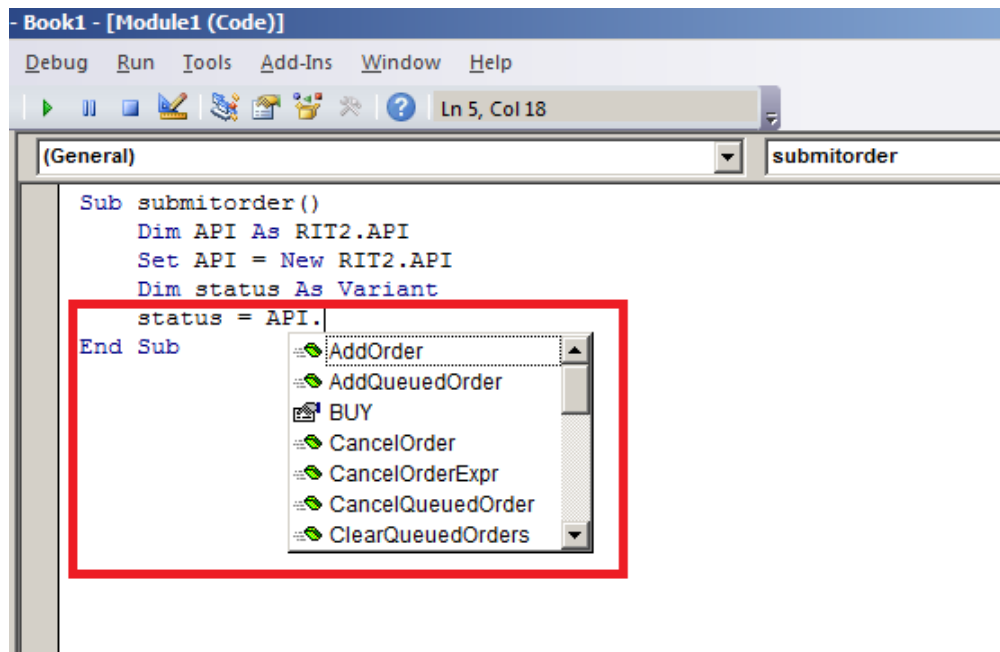
While you can code the buy and sell parameters directly with API.BUY and API.SELL, or indirectly with 1 and -1, if you are referencing cells you must use 1 (for buy) and -1 (for sell). **You will get an error if you reference cells containing the corresponding text values API.BUY and API.SELL. The same applies to referencing lmt_mkt parameters. See example in sample code 3.

Let's start by simply submitting a buy order. This can be accomplished with the following code:

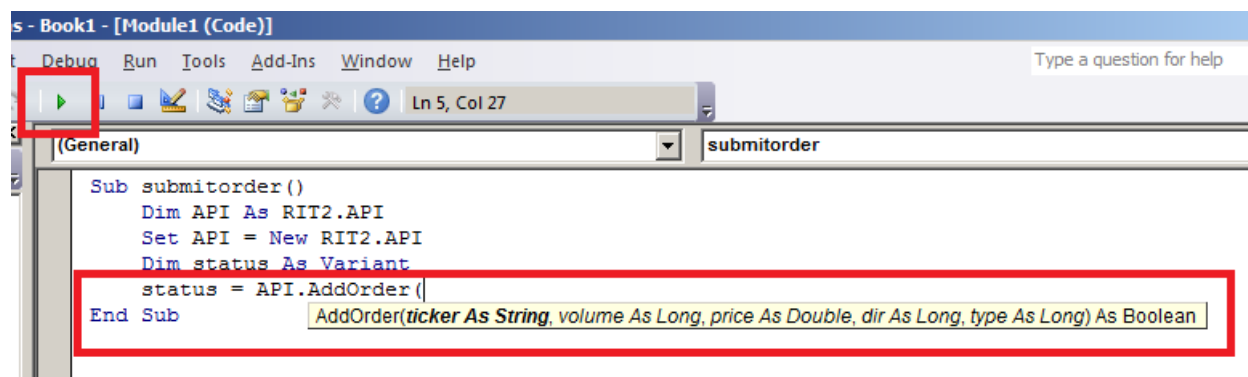
```
Sub submitorder()  
    Dim API As RIT2.API  
    Set API = New RIT2.API  
    Dim status as Variant  
    status = API.AddOrder("CRZY", 1000, 5, API.BUY, API.LMT)  
End Sub
```

Note that the example is setup assuming that students are trading a case with a stock "CRZY". If you are trading a different case, you will need to change the ticker otherwise the command will not work since the security "CRZY" does not exist.

As you type the beginning of the command "API", you will notice that a dropdown box will appear showing all of the different API commands that you can access.



You will also notice that as you type in the `API.AddOrder` command, a tooltip will show you the different command line parameters that are required for the `API.AddOrder` command.



Once you have completed the code, you can click on the red Play button in order to run the procedure. Click the button a few times and visit your RIT Client, you should see limit orders placed at \$5.00 to buy shares of CRZY.

Return Value: True or False

There are a few sample codes you can try in order to practice submitting different types of orders. Please feel free to try them.

Sample Code 1 – Limit Order:

Submit a limit buy order for the stock CRZY with size 1000, at a price of \$5.00. Assign True to the variable status if the order is successful, and assign False otherwise. Use “Range” to call cells that contain volume and price information. (So in this case, you should type 1000 in cell A1, and type 5 in

cell A2 as they are referenced for volume and price, respectively.) Note that Alternative 2 uses 1 instead of API.BUY and 1 instead of API.LMT.

Alternative 1:

```
Dim status as variant
status = API.AddOrder("CRZY", Range("A1"), Range("A2"),
API.BUY, API.LMT)
```

Alternative 2:

```
Dim status as variant
status = API.AddOrder("CRZY", Range("A1"), Range("A2"), 1, 1)
```

Sample Code 2 - Market Order:

Submit a market sell order for the stock CRZY with the size found in the cell A1 at the market price. Assign True to the variable status if the order is successful, assign False otherwise. Note that the sell price is specified here (with an arbitrary number, 1) even though it is ignored.

Alternative 1:

```
Dim status as variant
status = API.AddOrder("CRZY", Range("A1"), 1, API.SELL,
API.MKT)
```

Alternative 2:

```
Dim status as variant
status = API.AddOrder("CRZY", Range("A1"), 1, -1, 0)
```

Sample Code 3 - Referencing Cells for buy_sell:

Submit an order for the stock CRZY with the size found in the cell A1 at the market price. Assign True to the variable status if the order is successful, assign False otherwise. Whether the market order is to sell or buy depends on the value in the cell A2. Note that if a cell reference is used for the buy_sell parameter, the number value must be used in the cells. In other words, the cell A2 must contain 1 or -1. The strings "API.BUY" or "API.SELL" will not work.

Referencing cells for the lmt_mkt parameter follows the same pattern. The cell being referenced must contain 0 or 1 instead of the text "API.LMT" or "API.MKT".

```
Dim status as variant
status = API.AddOrder("CRZY", Range("A1"), 1, Range("A2"), 0)
```

Sample Code 4 – Using AddQueuedOrder:

Similar to AddOrder, you can also use AddQueuedOrder to submit a limit or market, buy or sell order. While all the parameters for AddQueuedOrder are the same as for AddOrder, the difference lies in the return value. While AddOrder returns True/False, AddQueuedOrder will return -1 (for failure to submit an order when the case is inactive) or an internal queued order ID* (for successful order submission).

```
Dim status as variant
status = API.AddQueuedOrder("CRZY", 1000, 5, API.BUY, API.LMT)
```

*When an order is submitted using either AddOrder or AddQueuedOrder API command, the RIT Server ‘queues’ an order before processing it in the system. Hence, when each order is queued, an internal queued order ID is first provided, and is converted later to an order ID when it appears on the Market Depth Book. This entire order submission process is generally completed in a fraction of a second when there are not many orders. However, one may choose to specifically use AddQueuedOrder in order to retrieve an internal queued order ID and cancel it individually before an order is processed. For more detailed information, please refer to ‘Sample Code 3 – Using CancelQueuedOrder’ under the ‘Cancelling an Order’ section below.

In addition, you can use the IsOrderQueued command to see if any particular order is currently queued. The command requires an internal queue ID as an input, and returns “True” for the order that is queued (at the moment), and “False” for any orders that are not queued (i.e. whether the order has been queued previously but successfully submitted, or simply the order has failed to be queued).

```
Dim status as variant
status = API.AddQueuedOrder("CRZY", 1000, 5, API.BUY, API.LMT)
API.IsOrderQueued(status)
```

From the above example, the IsOrderQueued command will return “False” because by the time that the VBA code reaches the “API.IsOrderQueued(status)” line, the order has been already queued and submitted from the API.AddQueuedOrder command. Hence, the command will return “False” since the order is not queued anymore. If there are several orders submitted by the API code, the IsOrderQueued command may return “True” if it is still queued.

Cancelling an Order

The following command cancels an order based on the order ID specified by the parameter.

General command Syntax:

```
API.CancelOrder (order_id)
```

Parameters:

Parameter	Description	Possible Values
Order_id	Order ID*	3142, 2323, Range("A1"), etc.

*Order IDs can be retrieved via the RTD functions – refer to the “Grabbing Ticker Specific Data Fields” section from the *RIT - User Guide - RTD Documentation.pdf*.

Return Value: None

There are a few code samples you can try in order to practice cancelling orders. Please make sure that you have submitted orders before you try cancelling them.

Sample Code 1:

Cancel the order with the Order ID 1500. Usually, you would make this more robust by linking the value to a cell location with the Cells(x,y) or Range(“mx”) functions as in Sample Code 2.

```
Sub cancelorder()
    Dim API As RIT2.API
    Set API = New RIT2.API
    API.CancelOrder (1500)
End Sub
```

Sample Code 2:

Cancel the order specified in cell A1

```
API.CancelOrder (Range("A1"))
```

Sample Code 3 – Using CancelQueuedOrder:

You can use CancelQueuedOrder to cancel an order that is ‘queued’ on the RIT Server before it appears on the Market Depth Book. Once you retrieve an internal order ID using AddQueuedOrder API command (from the ‘Sample Code 4 – Using AddQueuedOrder’ under ‘Submitting an Order’ section), you can use the following command to cancel it:

```
API.CancelQueuedOrder (internal queued order ID)
```

In case you would like to cancel all queued orders, you can use the following command:

```
API.ClearQueuedOrders
```

Again, please note that the above API commands only cancel the queued orders before they appear on the Market Depth Book. In order to cancel the orders that are submitted and visible on the Market Depth Book, please use the API.CancelOrder commands from above or follow the Cancel Order Expression instructions below.

Cancel Order Expression

The following command cancels all orders that satisfy the expression specified in the parameter.

General command Syntax:

```
API.CancelOrderExpr (order_expr)
```

Parameters:

Parameter	Description	Possible Values*
order_expr	Order expression	"Price > 20.00", "Volume = 400", "ticker = 'CRZY'", "Price > 20.00 AND Volume = 400", "Price > 20.00 AND Volume = 400", etc.

* Available operators include = (equal to), <> (not equal to), > (greater than), < (less than), >= (greater or equal to), <= (less than or equal to). You may also use brackets "("" to clarify order of operations.

Return Value: None

Sample Code 1:

Cancel all orders that have a price greater than \$20.00 and a volume equal to 400.

```
API.CancelOrderExpr ("Price > 20.00 AND Volume = 400")
```

Sample Code 2:

Cancel all orders associated with the stock CRZY.

```
API.CancelOrderExpr ("ticker = 'CRZY'")
```

Sample Code 3:

Cancel all orders that have a price greater than \$20.00 and a volume equal to 400, or all orders associated with the stock CRZY.

```
API.CancelOrderExpr ("(Price > 20.00 AND Volume = 400)  
OR ticker = 'CRZY'")
```

Accessing and Accepting/Declining Tender Offers¹

¹ If the API commands do not work, please make sure that you have the most recent version of the API installed on your PC. You can check this by following the instructions at http://rit.rotman.utoronto.ca/documents/API_Install_Instructions.pdf

In addition to the order submission and cancellation in RIT, the following API commands can be used to retrieve real-time data from RIT on institutional tender offers, in addition to accepting/declining those tender offers. Note that the “Cells” (or “Ranges”) VBA command is used in the examples below in order to display the result in a cell.

API Command Syntax: `API.GetActiveTenders()`

Description: Currently active tenders offered to the trader

Return Value: An array of integers, with each integer representing the ID number of a tender offer

Example

```
Sub test()  
    Dim API As RIT2.API  
    Set API = New RIT2.API  
    tenders = API.GetActiveTenders()  
    Cells(1, 1) = tenders(0)  
End Sub
```

Notes: “tenders(0)” is accessing the first value in the array returned by “API.GetActiveTenders()”. Arrays in VBA are zero-indexed. For the second value in the array, one would use “tenders(1)”.

API Command Syntax: `API.GetActiveTenderInfo(id)`

Description: Information on the requested active tender

Return Value: An array of information on the requested tender offer: [Tender ID, Ticker, Quantity, Price, Received Tick, Expiry Tick].

Example

```
Sub test()  
    Dim API As RIT2.API  
    Set API = New RIT2.API  
    tender = API.GetActiveTenderInfo(0)  
    Range("A1") = "Tender ID: " & tender(0)  
    Range("A2") = "Ticker: " & tender(1)  
    Range("A3") = "Quantity: " & tender(2)  
    Range("A4") = "Price: " & tender(3)  
    Range("A5") = "Received Tick: " & tender(4)  
    Range("A6") = "Expiry Tick: " & tender(5)  
End Sub
```

Notes: In the above code, information on the tender with an ID of 0 is being fetched. As with “API.GetActiveTenders()”, the result is returned as an array of information on that tender. Each value can be accessed using VBA array notation.

API Command Syntax: `API.AcceptActiveTender(id, bid)`

Description: Accepts the tender offer indicated by the ID

Return Value: True if successful, false otherwise

Example

```
Sub test()  
    Dim API As RIT2.API  
    Set API = New RIT2.API
```

```

    Dim status As Variant
    Status = API.AcceptActiveTender(0, 0)
    Cells(1, 1) = status
End Sub

```

Notes: A bid must be specified when sending a command to accept a tender offer. However, if the tender has a set bid, then this value will be ignored. In the above code, the tender with an ID of 0 is being accepted.

API Command Syntax: `API.DeclineActiveTender(id)`

Description: Declines the tender offer indicated by the ID

Return Value: True if successful, false otherwise

Example

```

Sub test()
    Dim API As RIT2.API
    Set API = New RIT2.API
    Dim status As Variant
    Status = API.DeclineActiveTender(0)
    Cells(1, 1) = status
End Sub

```

Notes: In the above code, the tender with an ID of 0 is being declined.

Additional API commands

In addition to the order submission and cancellation in RIT, the following API commands can be used to retrieve real-time data from RIT instead of using the RTD Link functions in Excel. Note that the “Cells” (or “Ranges”) VBA command is used in the examples below in order to display the result in a cell.

API Command Syntax: `API.GetCurrentPeriod`

Description: Current period

Equivalent Excel RTD Function: `=RTD("rit2.rtd",,"PERIOD")`

Example

```

Sub test()
    Dim API As RIT2.API
    Set API = New RIT2.API
    Dim status As Variant
    status = API.GetCurrentPeriod
    Cells(1, 1) = status
End Sub

```

API Command Syntax: `API.GetNLV`

Description: Current profit or loss

Equivalent Excel RTD Function: `=RTD("rit2.rtd",,"PL")`

Example

```

Sub test()
    Dim API As RIT2.API

```

```

Set API = New RIT2.API
Dim status As Variant
status = API.GetNLV
Cells(1, 1) = status
End Sub

```

API Command Syntax: API.GetOrderInfo(order_id)

Description: Detailed order information

Example

```

Sub test()
    Dim API As RIT2.API
    Set API = New RIT2.API
    Dim status As Variant
    status = API.GetOrderInfo(1027)
    Range("A2", "I2") = status
End Sub

```

Excel result

This API command requires 9 columns in Excel to retrieve the order information as shown below.

	A	B	C	D	E	F	G	H	I	J
1	OrderID	Ticker	period	Buy/Sell	Type	Volume	Price	Status	Remaining volume	
2	1027	CRZY_M	1	BUY	LIMIT	10000	9.97	PARTIAL	6000	
3										
4										

API Command Syntax: API.GetOrders

Description: Open order IDs

Example

```

Sub test()
    Dim API As RIT2.API
    Set API = New RIT2.API
    Dim status As Variant
    status = API.GetOrders
    Range("A2", "J2") = status
End Sub

```

Excel result

This API command displays the Order IDs of any live/partial orders submitted by the user that can be found from the Trade Blotter window from RIT Client.

	A	B	C	D	E	F	G	H	I	J
1										
2	709	708	707	706	215	214	139	136	#N/A	#N/A
3										
4										

Trade Blotter						
Live/Partial	Filed/Cancelled	Tenders	Endowments			
ID	Timestamp	Tick	Ticker	Price	Type	
709	12:44:29	P1 : 129	CRZY_M	8.46	BUY	
708	12:44:29	P1 : 129	CRZY_M	8.67	BUY	
707	12:44:28	P1 : 129	CRZY_M	8.87	BUY	
706	12:44:28	P1 : 128	CRZY_M	8.88	BUY	
215	12:26:56	P1 : 23	CRZY_M	9.96	SELL	
214	12:26:56	P1 : 23	CRZY_M	9.95	SELL	
139	12:25:11	P1 : 8	CRZY_M	10.13	SELL	
136	12:25:11	P1 : 8	CRZY_M	10.07	SELL	

API Command Syntax: `API.GetTickerInfo(ticker)`

Description: Detailed ticker and portfolio information

Example

```

Sub test()
    Dim API As RIT2.API
    Set API = New RIT2.API
    Dim status As Variant
    status = API.GetTickerInfo("CRZY_M")
    Range("A2", "N2") = status
End Sub

```

Excel result

This API command requires 14 columns in Excel to retrieve the ticker information as displayed below.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Ticker	Position	Last	Bid Size	Bid	Ask	Ask Size		Volume	Cost	Unrealized P&L	Realized P&L		Volume
2	CRZY_M	5000	9.11	310200	9.11	9.18	86900	0	9414500	9.21	-505.56	-1040.44		9414500

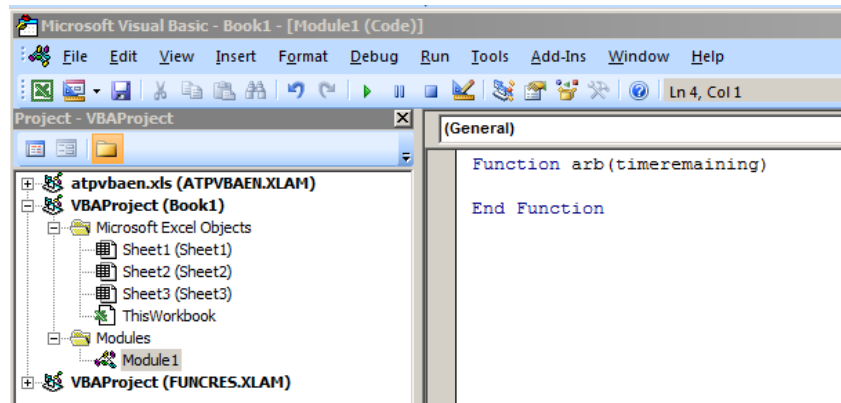
Additionally, there are the following additional API commands that can be used to retrieve the real-time data from RIT.

API Command syntax	Description	Equivalent Excel RTD Function
<code>API.GetCurrentPeriod</code>	Current period	<code>=RTD("rit2.rtd", "PERIOD")</code>
<code>API.GetNLV</code>	Current profit or loss	<code>=RTD("rit2.rtd", "PL")</code>
<code>API.GetTickers</code>	List of tickers	<code>=RTD("rit2.rtd", "ALLTICKERS")</code>
<code>API.GetTimeRemaining</code>	Time remaining	<code>=RTD("rit2.rtd", "TIMEREMAINING")</code>
<code>API.GetTotalTime</code>	Ticks per period	N/A
<code>API.GetYearTime</code>	Ticks per year	<code>=RTD("rit2.rtd", "YEARTIME")</code>

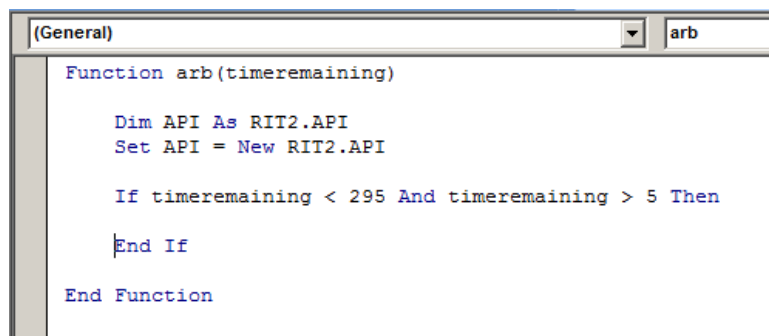
Algorithmic Trading Example - Arbitrage

This example assumes that students are building the arbitrage VBA codes while they are connected to the RIT Client with the ALGO1 case running. By default, the case runs for 300 seconds and there is one security that is traded on two different exchanges – CRZY_A and CRZY_M.

Before we start, please make sure that the Rotman Interactive Trader is enabled in Tools → References. (Please refer to the “Setting up RIT API configuration” section in page 11). Once you create a new module, you should type into the code-box on the right hand side of the window and define a function. In this example, the function will be called “arb” and it will have one parameter called “timerremaining”.



While there are many other ways to switch on/off the arbitrage algorithm, we will use the “timerremaining” to signal when the algorithm can start and stop. Once we initialize the RIT API, we can have the following ‘if statement’ to control the time that the algorithm is turned on and off.



Operationally, every time the “arb” function is run, Excel will initialize the API, and then check to see if the time remaining is between 5 and 295. As shown in the above example, the code currently initializes the API and allows for algorithmic trades to be submitted if the time remaining is between 5 and 295. (However, it will not submit anything because there are no commands written after the IF statements yet.)

The VBA code is now setup to run the arbitrage function whenever the case is running. The last step is to go into the code and program the logic to check for arbitrage opportunities, and execute the appropriate trades.

Before we setup the codes, it is suggested to have the market data from RIT and bring it to Excel using RTD links, so that we can analyze it with our algorithm.

	A	B	C	D	E	F	G
1		Bid	Ask				
2	CRZY_A	10.14	10.16	<-----	=RTD("rit2.rtd",,"CRZY_A","BID")	=RTD("rit2.rtd",,"CRZY_A","ASK")	
3	CRZY_M	10.06	10.08	<-----	=RTD("rit2.rtd",,"CRZY_M","BID")	=RTD("rit2.rtd",,"CRZY_M","ASK")	
4							

Now with this data linked in Excel, we can use an IF statement in our algorithm so that it only executes the buy/sell pair or orders when an arbitrage opportunity exists. Hence, the logic should be to check for two potential arbitrage opportunities:

If the ask price of CRZY_A is less than the bid price of CRZY_M, then the algorithm should submit a market order to buy CRZY_A and a market order to sell CRZY_M.

If the ask price of CRZY_M is less than the bid price of CRZY_A, then the algorithm should submit a market order to sell CRZY_A and a market order to buy CRZY_M.

The code is presented as follows:

```

(Function) arb
Function arb(timerremaining)
    Dim API As RIT2.API
    Set API = New RIT2.API

    If timerremaining < 295 And timerremaining > 5 Then

        If Range("CRZY_A_BID") > Range("CRZY_M_ASK") Then
            OrderID = API.AddOrder("CRZY_M", 1000, 0, API.BUY, API.MKT)
            OrderID = API.AddOrder("CRZY_A", 1000, 0, API.SELL, API.MKT)
        End If

        If Range("CRZY_M_BID") > Range("CRZY_A_ASK") Then
            OrderID = API.AddOrder("CRZY_A", 1000, 0, API.BUY, API.MKT)
            OrderID = API.AddOrder("CRZY_M", 1000, 0, API.SELL, API.MKT)
        End If

    End If

End Function

```

Here, each cell is named with the security name and bid/ask information. As you can see from the example below (highlighted in blue) Cell B2 has been named as "CRZY_A_BID", etc. This is not a required step, but naming each cell will help you understand the information it contains. You can use Range("B2") instead of Range("CRZY_A_BID")

	A	B	C	D	E	F	G
1		Bid	Ask				
2	CRZY_A	10.37	10.38	<-----	=RTD("rit2.rtd",,"CRZY_A","BID")	=RTD("rit2.rtd",,"CRZY_A","ASK")	
3	CRZY_M	10.34	10.35	<-----	=RTD("rit2.rtd",,"CRZY_M","BID")	=RTD("rit2.rtd",,"CRZY_M","ASK")	

The code sets OrderID = API.AddOrder because whenever an order is submitted to the API, it returns an "Order Identifier". In our situation, we will not use the OrderID (in the future, one could use the "Order Identifier" to check the status of the order, cancel it, etc.)

Alternatively, this can be replaced with the examples of the code we used in the “Submitting an Order” section above as shown below.

```

If timeremaining < 295 And timeremaining > 5 Then
Dim status As Variant

If Range("B2") > Range("C3") Then
    status = API.AddOrder("CRZY_M", 1000, 0, API.BUY, API.MKT)
    status = API.AddOrder("CRZY_A", 1000, 0, API.SELL, API.MKT)
End If

```

Finally, in order to run the “arb” function, you would need to return to the spreadsheet, find a cell and type in “=ARB(E2)”

The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G	H	I
1		Bid	Ask		Time Remaining				
2	CRZY_A	0	0		300	<--=RTD("rit2.rtd", "TIMEREMAINING")			
3	CRZY_M	0	0						
4									
5					0	<--=ARB(E2)			
6									

This will tell Excel to execute the function “Arb” and pass into the function the value from cell E2 (which happens to be the time remaining in the simulation). In this situation, the time remaining is 300 seconds, so the code in the “IF” statement will not execute. Once the case is started (and timeremaining is < 295), then the code in the “IF” statement will execute.

While the ALGO1 case is running, whenever the markets become crossed, the algorithm should automatically buy shares on one market and sell shares on the other and generate a profit.

Excel runs the function (and the code) on a continual basis. Therefore, when students try to edit the code in VBA, it will cause an error (because Excel is trying to run half-written code). In order to proceed, students should delete the function =ARB(E2) in the spreadsheet before editing their code, and then add it back later.

Note that this is a simple arbitrage algorithm. Please feel free to try to improve this by making it more dynamic (i.e. link the order size and price to Excel), include the gross/net limit restrictions in the case, etc.