Morningstar Add-In

VBA Guide

MORNINGSTAR®

Table of Contents

Overview

Visual Basic for Applications (VBA) allows developers to build user defined functions and automate processes. The Morningstar Add-In application allows the use of VBA to help enhance the user experience.

VBA commands

Disclaimer

Visual Basic for Applications (VBA) scripts should be used with caution and all tests should be performed in test environments and in accordance with your company's policy. This VBA guide is provided by Morningstar to exemplify additional functionality – Morningstar does not take any responsibility for damages to work sheets, programs, or other systems resulting from codes herein or other VBA scripts developed with these codes.

Refresh all Morningstar Add-In calls with assigned button

This command will refresh all Add-In calls upon running the subroutine. Create a button and assign the code to refresh when clicking the button.

```
Sub RefreshAddin
Set cmd = Application.CommandBars("Cell").Controls("Refresh All")
cmd.Execute
End Sub
```

Refresh all Morningstar Add-In calls upon opening workbook

This command will automatically refresh all Add-In calls upon opening the Excel file.

```
Sub Auto_open()
Set cmd = Application.CommandBars("Cell").Controls("Refresh All")
cmd.Execute
End Sub
```

Refresh a specific cell within the workbook

To refresh a cell, reference a specific cell or array and assign the code to a button.

```
Sub RefreshAddin()
Sheet1.Cells(1, 1).Activate 'change cell reference here (Row #, Column #)
Set cmd = Application.CommandBars("Cell").Controls("Refresh")
cmd.Execute
End Sub
```

Refresh all Morningstar Add-In calls at a specific time

Certain users would like to refresh worksheets at certain times. This is the code for refreshing on a specific time as well as the command to stop the code. In the worksheet, reference a cell and type in the time the sheet should be refreshed. For example: 10:01:00 am.

```
Public dTime As Date
Dim lNum As Long

Sub RunOnTime()
dTime = Sheet1.Cells(1, 1) 'change the cell reference here (Row #, Column #)
Application.OnTime dTime, "RunOnTime"
Set cmd = Application.CommandBars("Cell").Controls("Refresh All")
cmd.Execute
End Sub
```

To cancel the "RunOnTime" subroutine, use the following subroutine:

```
Sub CancelOnTime()
Application.OnTime dTime, "RunOnTime", , False
End Sub
```

Refresh all Morningstar Add-In calls for recurring intervals

In order to refresh at regular time intervals, substitute the refresh "RunOnTime" subroutine with the subroutine below:

This is the code to run the call every 60 minutes.

```
Public dTime As Date
Dim lNum As Long

Sub RunOnTime()
dTime = Now + TimeSerial(0, 60, 0)    ' add the amount of delay here
Application.OnTime dTime, "RunOnTime"
Set cmd = Application.CommandBars("Cell").Controls("Refresh All")
cmd.Execute
End Sub
```

Caution: Frequent calls (1 minute or less) to the server might cause a degradation of the server speed and may temporarily prevent the application from running properly. Please use caution when setting recurring intervals.

To cancel the run on time subroutine, use the following subroutine.

```
Sub CancelOnTime()
Application.OnTime dTime, "RunOnTime", , False
End Sub
```

Process time

Morningstar Add-In libraries could affect process times for other VBA programs running simultaneously within Excel. In some circumstances, the following codes can decrease latency by disabling any commands in the workbook that are running in the Morningstar Add-In:

Disable/Enable Ribbons and Buttons

The Morningstar Add-In checks each active cell in the Excel worksheet. This interaction allows the program to highlight the appropriate ribbon button if a cell contains an Add-In function.

During calculations that involve large numbers of cells or arrays, this process can slow response time. If the Morningstar Add In is causing a delay, then disabling this interaction could improve response time. It is important to keep or return this option to ON, in order to utilize full functionality of the program. The directions to turn the ribbon interactions on and off are listed below:

By default, the option is turned on. Every time a cell is active, the Morningstar Add-In will run code that will enable or disable the appropriate buttons within the ribbon. The code below returns the Add In to the default/on state.

```
Application.Run("MORNIconIsOn")
```

The code below will turn off the interaction. Every time a select cell is active, the Morningstar Add-In will NOT run a code that will enable or disable the appropriate buttons within the ribbon.

```
Application.Run("MORNIconOff")
```

Disable/Enable Events

Events are a powerful aspect of Excel programming. They enable you to make your application respond to user actions such as entering data into cells or clicking the print button. If your application uses events, you will probably also need to be able to control whether or not an event executes its code or not (e.g. to avoid event looping or to enable your code to do things you are preventing your user to do through the user interface).

Place before VBA code to disable events:

```
Application.EnableEvents = False
```

Place after VBA code to re-enable events:

```
Application.EnableEvents = True
```

It's important to ensure that Application.EnableEvents is set back to True again before the procedure ends.

Example:

```
Application.EnableEvents = False

‘your code here

Application.EnableEvents = True
```

Disable/Enable ScreenUpdating

The ScreenUpdating property controls most display changes on the monitor while a procedure is running. When screen updating is turned off, toolbars remain visible and Excel still allows the procedure to display or retrieve information using status bar prompts, input boxes, dialog boxes, and message boxes. You can increase the speed of some procedures by keeping screen updating turned off. You must set the ScreenUpdating property to True when the procedure finishes or when it stops after an error.

Place before VBA code to disable events:

```
Application.ScreenUpdating = False
```

Place after VBA code to re-enable events:

```
Application.ScreenUpdating = True
```

When the macro ends, don't forget to set the ScreenUpdating property back to True.

Disable/Enable Morningstar Add-In Application

This code completely removes and reinstalls the application from the Excel environment. Please note that the application is not removed from your computer, but only from the Excel environment. Also note that the process of reinstalling the ribbon may take several seconds. Please try other solutions first before disabling and re-enabling the application.

Disable the Add-In

Place before VBA code to disable events:

```
AddIns("Morningstar Add-In").Installed = False
```

Enable the Add-In

Place after VBA code to disable events:

```
AddIns("Morningstar Add-In").Installed = True
```

Upload Multiple Worksheets

This code allows a user to upload data on multiple worksheets to upload simultaneously to the Marketplace.

- `BatchUpload = function name`
- `CME = provider name`
- `CME_Futures = feed name`
- `Book2.xlsx = workbook name`
- `Sheet3 = sheet name`
- `A1:I3 = cell range`

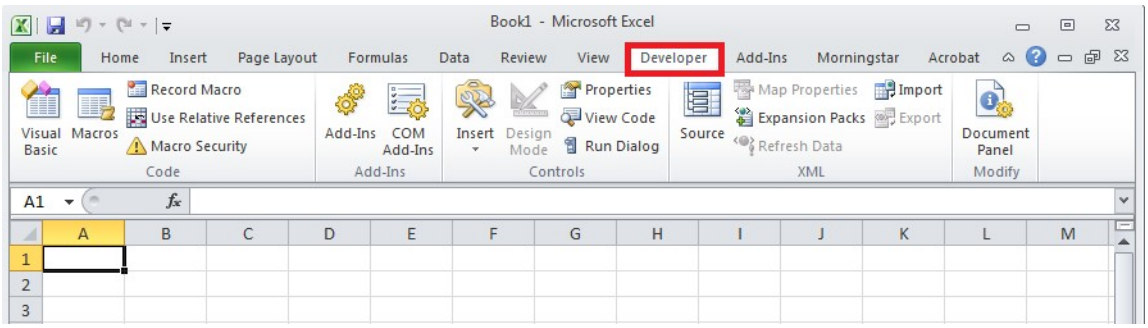
Note: the parameter False,False,True allow users to see the window status upload without going to the user interface.

```
Sub Button1_Click()

ret = Application.Run("BatchUpload", "CME", "CME_Futures",
"[Book2.xlsx]Sheet3!$A$1:$I$3",False,False,True)
MsgBox ret

End Sub
```

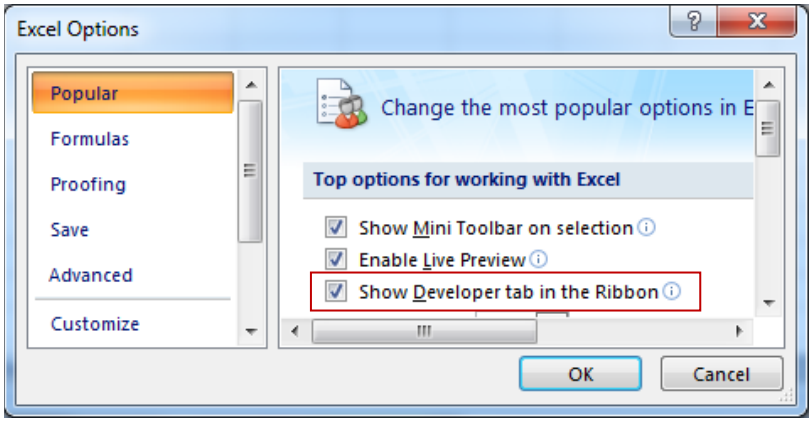
Enter VBA Editor

To enter the VBA editor, click on the Developer tab in the ribbon.

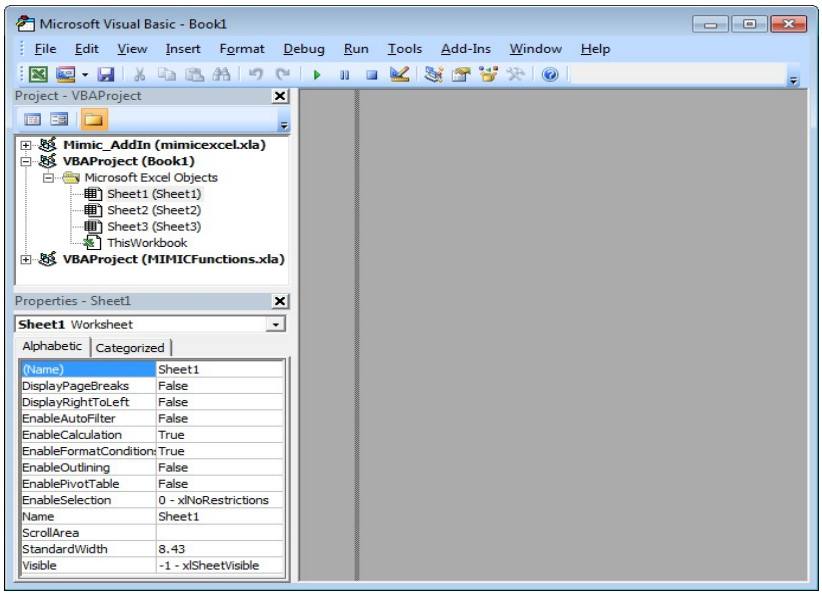
Please note that the example screenshots being displayed are those of Office 2007. Your screen may look different depending on the version of Office that you have.



If the developer tab is not available, it must be selected from the Excel options. Click on the Microsoft Office icon on the top left and select "Excel Options". In Office 2007, under the popular tab, select "Show Developer tab in the Ribbon". For other versions of Office, please go to help in Excel to determine how to enable the developer tab.



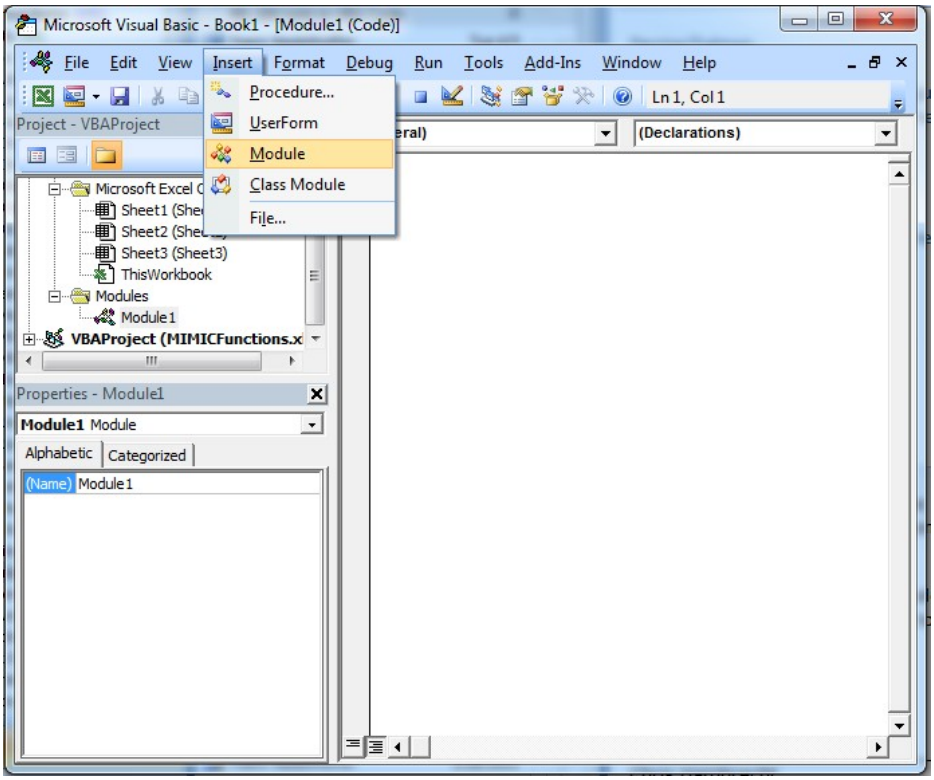
Once in the developer tab, click on the first icon on the left titled "Visual Basic". This will open the Visual Basic editor.



Inside the Visual Basic editor a user can create macros and functions specific to certain sheets or to entire workbooks.

Creating Modules

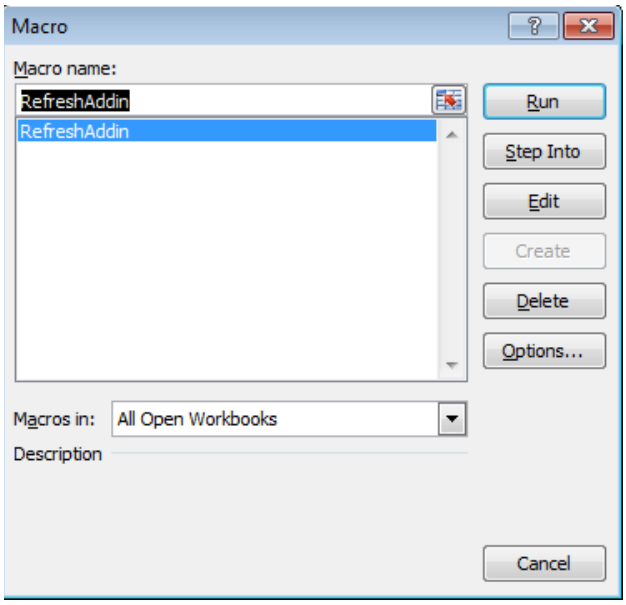
Modules contain VBA commands that are run to control Microsoft Excel. To create a module, simply go to insert/module in the VBA editor.



Once inside the module, a user can simply copy and paste the code examples from the VBA Commands section. Once the module is created, exit out of the VBA editor.

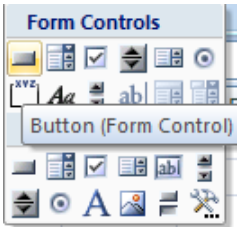
Run Macros

To run a macro that was just created, go to the developer tab and click on Macros. Once inside the macro window, highlight the macro to run and click "run".



Create a Button

To create a button for a macro, go to the developer tab and click the insert icon.
From the drop down menu under "Form Controls", click on the very first icon.



Next, select a cell where the macro is to be placed. Once the cell is selected the "Assign Macro" pane will appear. Select the macro name and click "OK". Right click on the button and select, "edit text" to rename the button.