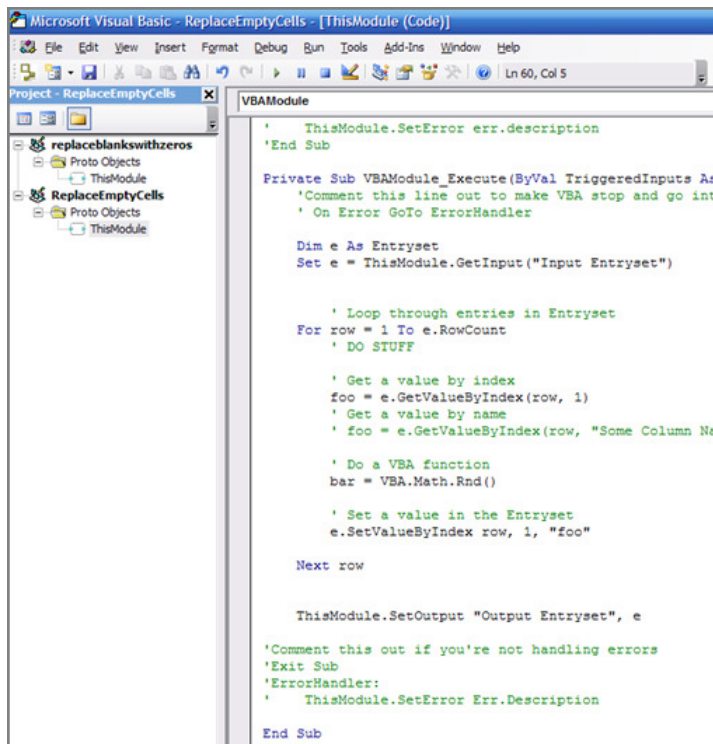


Introduction to Using VBA

MiniTutorial



```
Microsoft Visual Basic - ReplaceEmptyCells - [ThisModule (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - ReplaceEmptyCells
  replaceblankswithzeros
  Proto Objects
  ReplaceEmptyCells
  Proto Objects
  ThisModule
VBAModule
  ' ThisModule.SetError err.description
  'End Sub
Private Sub VBAModule_Execute(ByVal TriggeredInputs As
  'Comment this line out to make VBA stop and go int
  ' On Error GoTo ErrorHandler

  Dim e As Entryset
  Set e = ThisModule.GetInput("Input Entryset")

  ' Loop through entries in Entryset
  For row = 1 To e.RowCount
  ' DO STUFF

  ' Get a value by index
  foo = e.GetValueByIndex(row, 1)
  ' Get a value by name
  ' foo = e.GetValueByIndex(row, "Some Column Na

  ' Do a VBA function
  bar = VBA.Math.Rnd()

  ' Set a value in the Entryset
  e.SetValueByIndex row, 1, "foo"

  Next row

  ThisModule.SetOutput "Output Entryset", e

  'Comment this out if you're not handling errors
  'Exit Sub
  'ErrorHandler:
  ' ThisModule.SetError Err.Description
End Sub
```

In this MiniTutorial, you will learn the basic concepts necessary to use Visual Basic for Applications (VBA) to extend Proto's functionality.

This tutorial should take 15-30 minutes, and then you can explore examples on your own.

MiniTutorial Overview

In this MiniTutorial, you will learn how to use Visual Basic for Applications (VBA) within Proto to process data and interface with other applications through Automation. The VBA Integrated Development Environment (IDE) in Proto is the same VBA IDE that is found in MS Excel, Access and other Office applications, and should be familiar to people with programming experience. You will:

- Learn about the most common uses of VBA to extend the functionality of Proto.
- Walk through some VBA scripts and learn what they do in Proto.
- Insert a pre-configured VBA module in Proto.
- Learn the commands to manipulate Entrysets and communicate between VBA and Proto.
- Explore VBA scripts on your own to process Entrysets, connect to databases, interface with MS Outlook and parse the XML from a Criagslist search.

Before you start this tutorial, you will need to have installed Proto Financial Version 1.8 and downloaded the tutorial files from the Proto website. Tutorial files can be downloaded in a zipped format from Proto's website at <http://www.protosw.com/devcenter>

This MiniTutorial uses a Proto file with sample data and some pre-configured applications of VBA.

- 1 Locate the folder named "MiniTutorial - VBA" in the folder titled "Proto Getting Started Lessons."
- 2 Open the file named "Using VBA.proto" by double-clicking on the file.
- 3 Save this Proto dashboard with a new name in the "MiniTutorial - VBA" folder. To do so, select File > Save As from the application menu bar. Then browse to the appropriate location and save the file as "Using VBA - Copy.proto."

The Most Common Uses of VBA in Proto

Visual Basic for Applications is a critical component of the Proto environment for advanced users and people with some programming experience. While VBA is typically licensed by software vendors and included as an environment to *automate* their applications, in Proto VBA is used strictly as a scripting environment to process data and connect to other applications through automation APIs.

VBA allows users to extend Proto's capabilities and fill in functional gaps that are not covered by the current set of primitive modules in Proto. For the millions of expert Excel, Access and users, VBA also provides comfort that they will not get "stuck" trying to work with data in Proto.

#1 – Processing Entrysets

The most important use of VBA in Proto is the ability to process data in a procedural fashion. Proto provides many modules to handle common data processing needs like filtering, manipulating columns, creating simple calculated columns, iteratively applying a Component to a set of data, etc. However, sometimes a 10-line script is a more natural, concise and effective way to process an Entryset.

#2 – Interfacing Programmatically with Other Windows Applications

Through VBA, Proto can interface with hundreds of applications and useful libraries. Whether you need to create an email in MS Outlook, execute a shell script or update records in a database through ADO, the VBA IDE provides users with the same environment they use in MS Office to script applications.

Understanding a Simple VBA Script

Let's start by exploring a simple VBA script used to add a Row Index column (from 1 to N) to an Entryset. The VBA module is embedded inside a Component, so it can be used in Proto just like any other Component via its inputs and outputs, without the user knowing anything about the VBA code inside.

Here is the entire script:

```

Private Sub VBAModule_Execute (ByVal TriggeredInputs As Variant)
    Dim eset As Entryset
    Dim indexColumnName As String

    indexColumnName = ThisModule.GetInput("What to Call It")
    Set eset = ThisModule.GetInput("Entryset")

    eset.InsertColumn 1, indexColumnName

    For i = 1 To eset.RowCount
        eset.SetValueByIndex i, 1, i
    Next

    ThisModule.SetOutput "Entryset", eset

End Sub

```

Now let's step through it line by line, describing what each statement is doing.

- 1 Proto calls a method in the VBA module to pass the execution context to VBA. The subroutine in VBA accepts a list of the inputs to the VBA module, which were triggered in the current execution cycle. Usually you won't care which input(s) were triggered, but occasionally you will want to conditionally execute the code only when an input such as "Insert Rows" is triggered, and not "Set Rows to Insert into DB."

```

Private Sub VBAModule_Execute (ByVal TriggeredInputs As Variant)

```

- 2 Next the script initializes two variables. The "eset" variable will allow you to reference the Entryset that is being passed into the VBA module to process. The Entryset Object exposes Entryset API functionality in VBA to set values, add rows / columns, create Entrysets, etc. The other variable is a VBA String, and will hold the name to give the index column. This value will also be passed into the VBA module via a Connection in the Builder.

```

    Dim eset As Entryset
    Dim indexColumnName As String

```

- 3 The next two lines set the variables by requesting the values from Proto. Note that because the "eset" variable is an Object, you must use the VBA syntax for setting Object references.

```

    indexColumnName = ThisModule.GetInput("What to Call It")
    Set eset = ThisModule.GetInput("Entryset to Process")

```

- The next line inserts a new column in the Entryset in the leftmost column position.

```
eset.InsertColumn 1, indexColumnName
```

- The next three lines set up a loop in VBA to iterate over each row in the Entryset. Each iteration of the loop sets the value of the cell in the first column, for the current row, with the value of the variable "i".

```
For i = 1 To eset.RowCount  
    eset.SetValueByIndex i, 1, i  
Next
```

- After processing the Entryset, the next line sets an output on the VBA module named "Processed Entryset".

```
ThisModule.SetOutput "Processed Entryset", eset
```

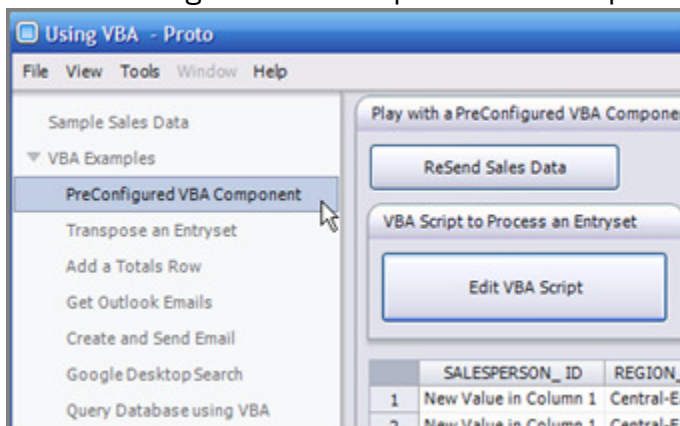
- Finally, the "VBAModule_Execute" subroutine ends. When this point is reached, execution context is returned to Proto from VBA. Any outputs set with values as shown in Step 6 will be 'triggered' by the VBA module. Data will be sent down all Connections made from triggered outputs on the VBA module.

```
End Sub
```

Quick Start to Processing an Entryset

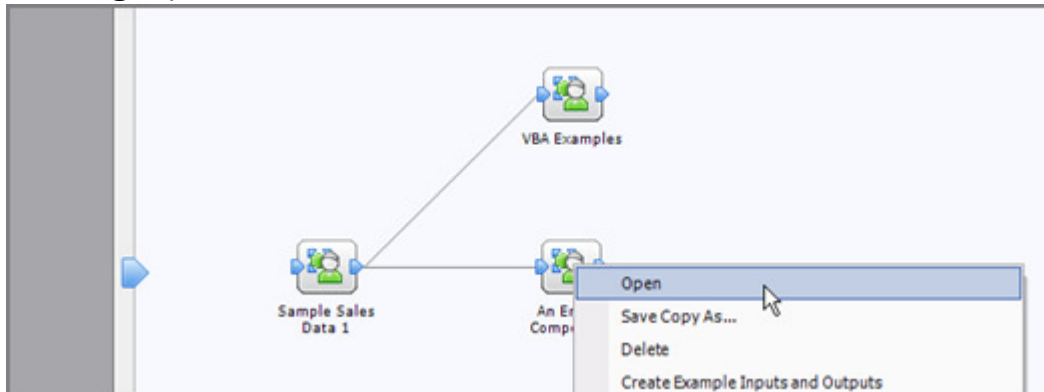
You can define variables and make connections directly to the VBA primitive module. However, frequently you will simply want to process an input Entryset with VBA. You can save a few minutes by using a Component that is prewired to process a single Entryset.

In a moment, you will insert the VBA Script to Process an Entryset Component on your own from the Getting Started Components. To see what you are about to build, click on the "PreConfigured VBA Component" sub-step in the "VBA Examples" workflow step.



Now try inserting and experimenting with the “VBA Script to Process an Entryset” Component on your own.

- 1 Go to the Builder
- 2 Open the Component called “An Empty Component” by right-clicking on it and selecting “Open” from the menu.



- 3 Inside the Builder window for the An Empty Component, insert the prebuilt VBA Component.

Location: ..\Getting Started Components\Do Stuff with Data

Component: VBA Script to Process an Entryset.proto

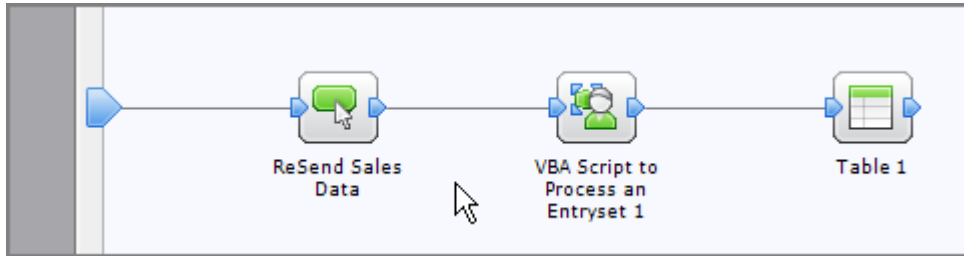
- 4 Insert a primitive Button module by right-clicking on the background of the Builder and selecting it from the “Insert Module...” menu. You will use this to gate the Entryset input, and retrigger the VBA module.
- 5 Name the Button “ReSend Sales Data” by double-clicking the Button module and changing the text in the Label field in the General tab of the Property Panel.
- 6 Insert a primitive Table module. You will use this to view the output from the VBA module.
- 7 Make the following Connections.

From: Builder Inputs
Output: Sample Sales Data
To: ReSend Sales Data
Input: Button Connectors > Set Value

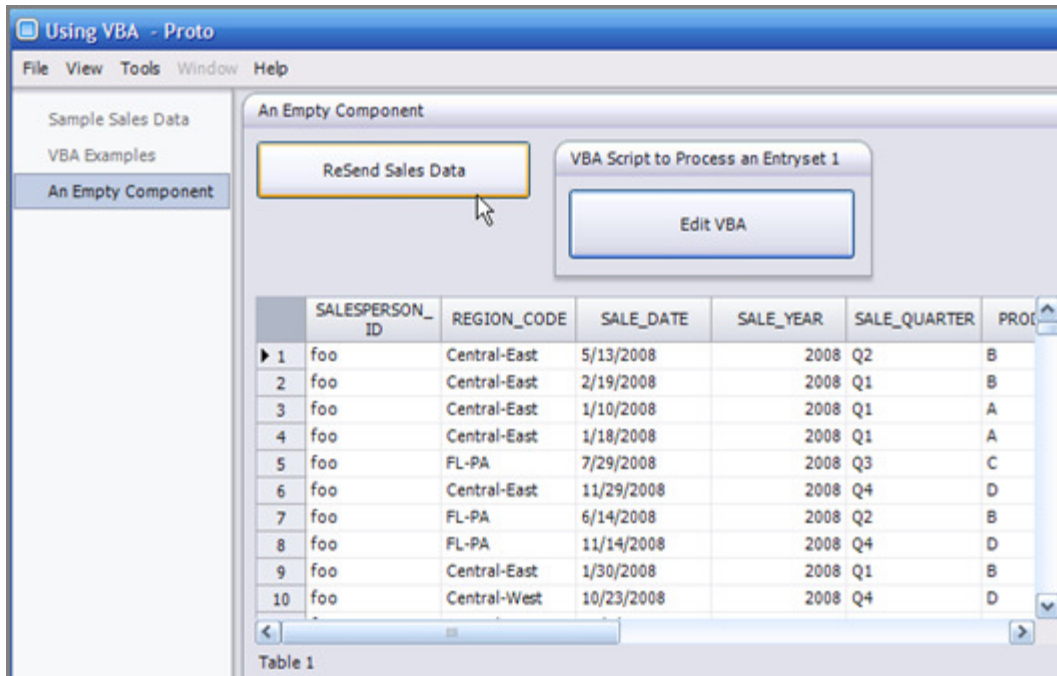
From: ReSend Sales Data
Output: Button Connectors > Get Value
To: VBA Script to Process an Entryset 1
Input: Input Entryset

From: VBA Script to Process an Entryset 1
Output: Processed Entryset
To: Table 1
Input: Table Connectors > Set Entryset

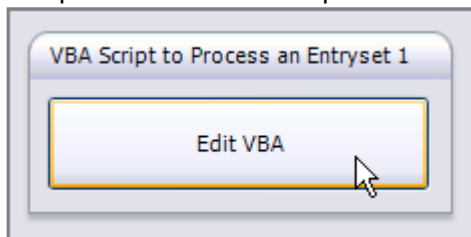
- 8 Your Builder window should look similar to the following screenshot.



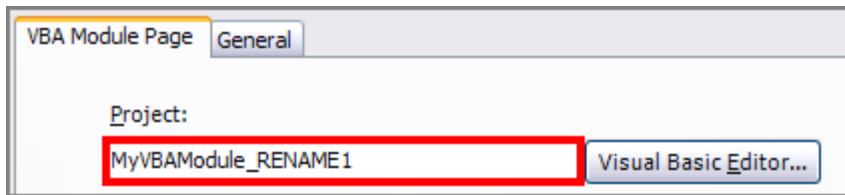
- 9 Go to the Viewer and arrange the Button, Table and the Component by entering Edit Mode as shown below.



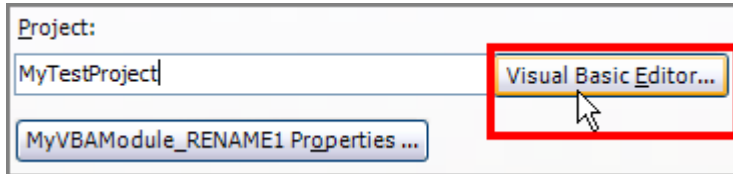
- 10 Click on the Button called "Edit VBA" in the Viewer to open up the Property Panel of the pre-connected VBA primitive module.



- 11 Give the VBA Project a new name (alphanumeric with no spaces) if you like in the "Project" label in the VBA Module Page tab of the Property Panel.



- 12 Then click on the button “Visual Basic Editor...” to go to the VBA IDE.



- 13 In the VBA IDE, find the “VBAModule_Execute” Subroutine, and experiment with the VBA code. To run the VBA code, go to the Viewer and click on the Button you created called “ReSend Sales Data.”

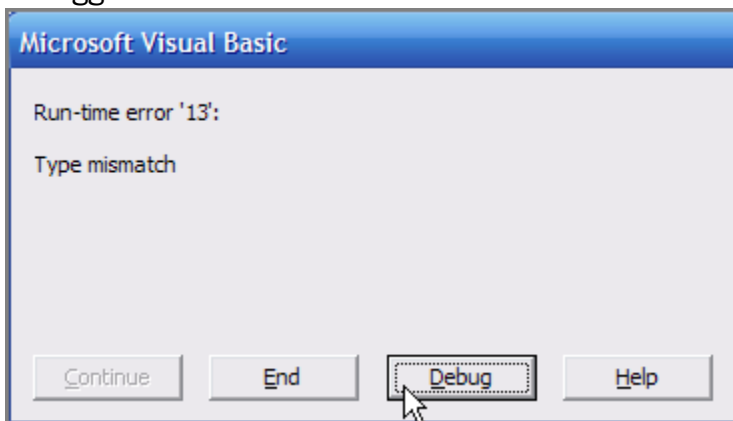
Note: If there is an error in the VBA code and the VBA IDE window is open (even in the background), then you may not be prompted with the VBA error dialog even though execution has stopped. In the Viewer, you will simply see in the status bar at the bottom left:



To get to the error dialog, bring the VBA IDE window to the front.

If the VBA IDE window is **closed** when the error occurs, the VBA error dialog will pop up in the front of the screen.

From the Microsoft Visual Basic error window, you can choose to End execution or Debug the script. If you choose to Debug the script, you will enter the familiar VBA debugger in the VBA IDE.



Configuring the VBA Primitive Module

When you insert a new VBA primitive module into the Builder, you'll need to configure the VBA module's input and output variables before you can make Connections to it. The general sequence of events for setting up a new VBA primitive module is as follows:

- Insert the VBA primitive module
- In its Property Panel, define input and output variables (which you can connect to in the Builder)
- Connect to the inputs and outputs on the Builder
- Go to the VBA IDE and write a script

Inserting modules and making Connections are covered extensively in Lesson 1 and Lesson 2. Below we will cover only the VBA primitive module Property Panel and a few other useful tricks to quickly set up the code.

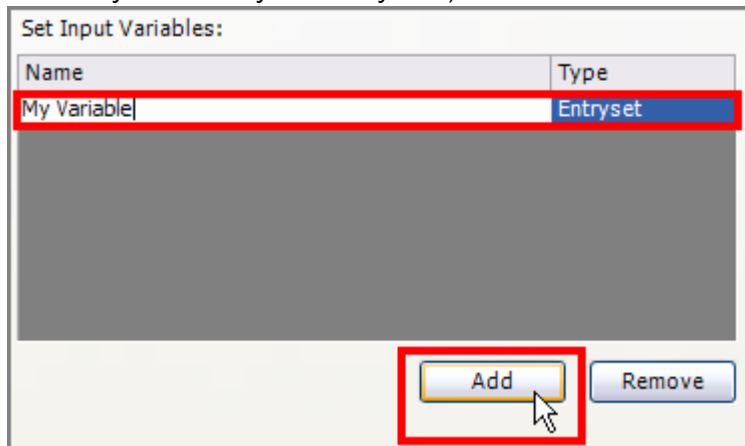
The VBA Primitive Module Property Panel

The Property Panel of the VBA module is relatively straightforward. The most important functions of the Property Panel are:

- Creating input variables
- Creating output variables
- Naming your VBA project

To create input variables, follow these steps.

- 1 Click the Add button in the "Set Input Variables:" Then double-click in the "Name" column to type in a name for the variable. And finally set the type of the variable (this is mainly necessary for Entrysets).

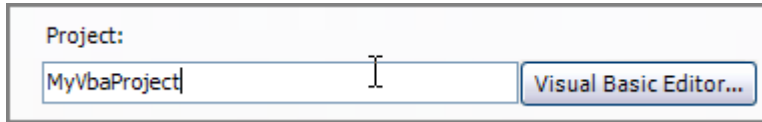


- 2 To delete a variable, click on the appropriate row in the grid, and then click the Remove button.

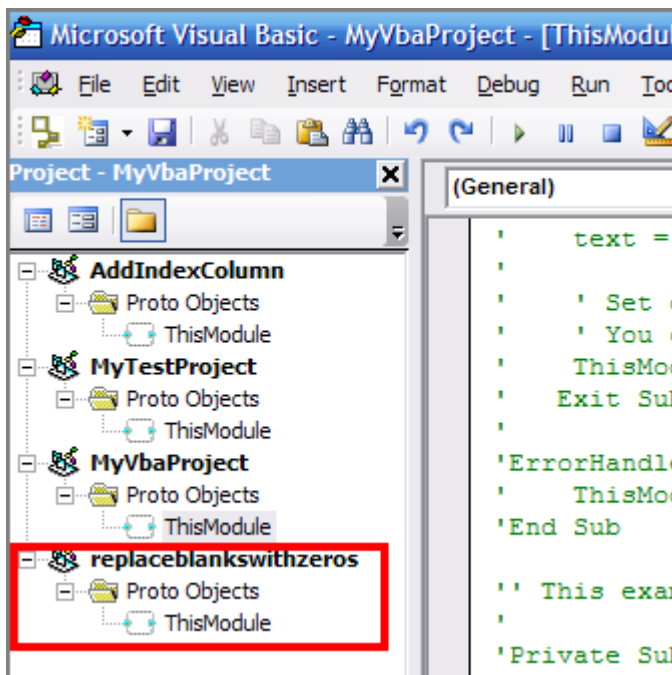
To create output variables, you follow virtually identical steps in the “Set Output Variables:” section of the Property Panel, making sure to set the type for Entryset outputs.

To name your VBA project, follow these steps.

- 1 Type in a new name (alphanumeric with no spaces) in the Project label in the Property Panel.

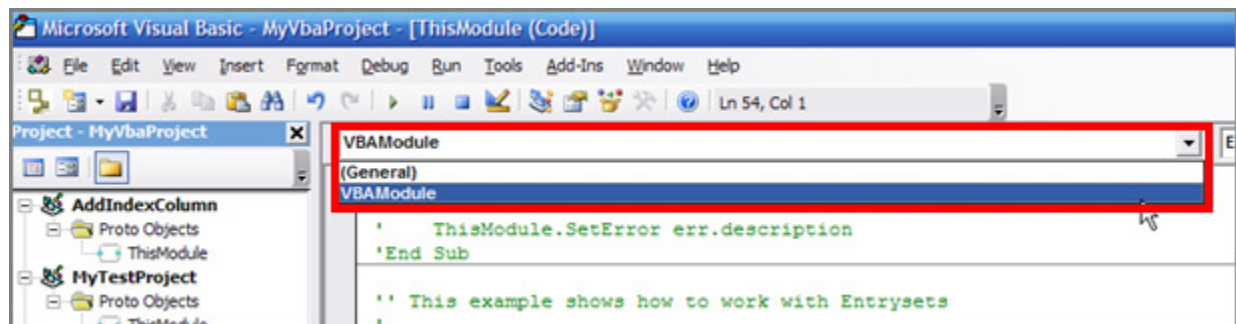


- 2 In the VBA IDE, the name of your project will be displayed in the project browser, and it is much easier to locate and edit the right code if the VBA projects have useful names.



Setting up the Code

When you go to the VBA IDE of a new project, there will not be a “VBAModule_Execute” Subroutine defined initially. There is a useful shortcut to add this method with the correct signature by selecting “VBAModule” from the method explorer.



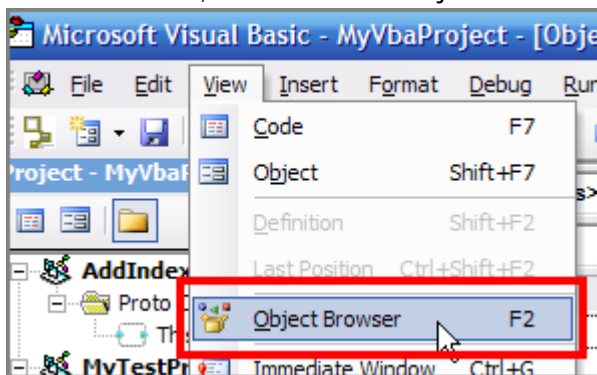
Alternatively, you can use one of the commented-out sections of code as a starting point. There are two: the first is very simple, the second includes iterating over an Entryset. To uncomment one of these blocks of code, delete the leading single quote from each line.

The Entryset and Proto APIs Accessible in VBA

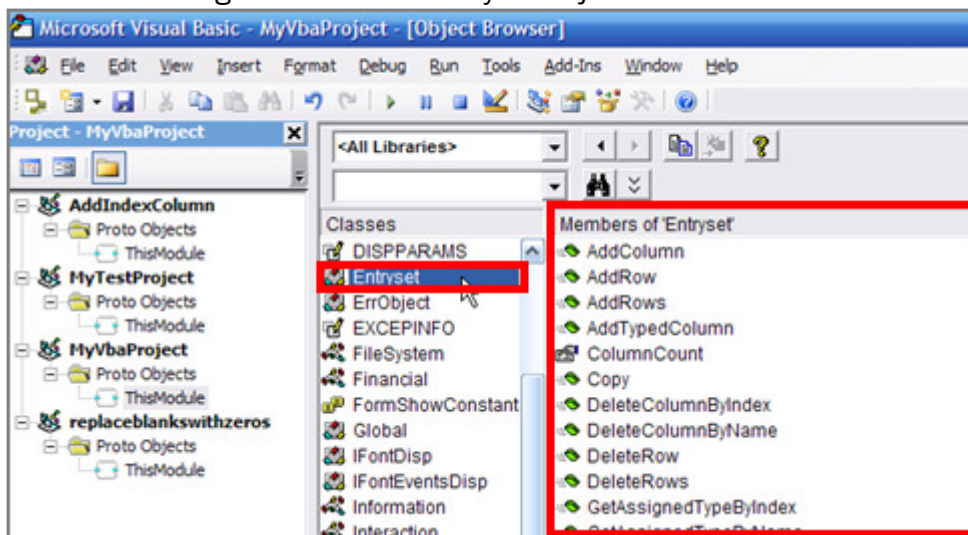
As you saw in the code reviewed earlier in this tutorial, the Entryset Object APIs are exposed in VBA so that you can interface with Entryset data to create columns, set values, etc. Additionally, there are methods exposed by the ThisModule Object, that allow VBA to communicate with Proto.

To explore the methods exposed by the Entryset Object and the ThisModule Object, go to the Object Browser in the VBA IDE, and then click on the Entryset class or the ThisModule class.

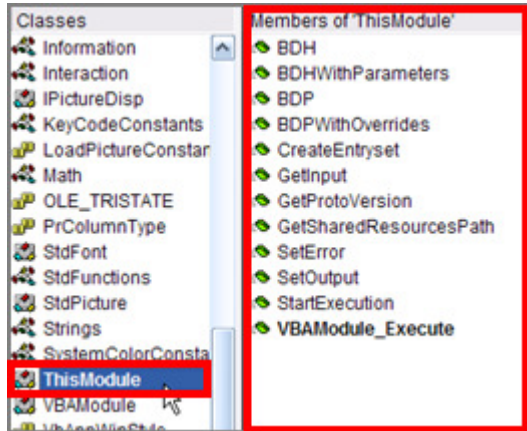
- 1 In the VBA IDE, select View > Object Browser from the application menu.



- 2 Then click on the Entryset class in the Object Browser window to explore the methods and signatures of the Entryset object.



- 3 Likewise you can select the ThisModule class to view the methods and signatures you can use to interface with Proto.



Explore the Sample Scripts on Your Own

For experienced VBA programmers, this introduction should set you on your way to using VBA within Proto.

In the “Using VBA.proto” file included with this MiniTutorial, there are a number of Components that rely on VBA scripts, which you can explore on your own. Some of the Components illustrate basic concepts such as transposing an Entryset or adding a row with totals for numeric columns. Other Components make extensive use of other libraries within VBA to interface with MS Outlook, send keystrokes to a window, perform a Google Desktop Search, connect to a database through ADODB, and parse the XML response from a Craigslist search.

Thank You!

Thank you for your interest in Proto. If you have feedback regarding this tutorial or would like to see more training material covering specific topics or questions, please email us with suggestions at: devcenter@protosw.com