

Advancing the  
wireless revolution™



[www.awrcorp.com](http://www.awrcorp.com)

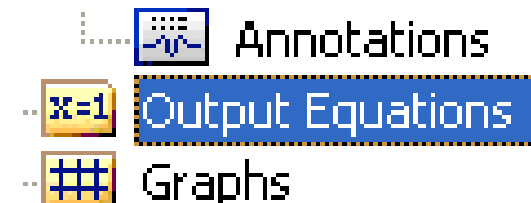
# Scripting



# User Written “Code” in MWO

## Equations

On Schematic or Global Definitions Page  
- Also typically on Output Equations Page



To set a parameter

```
inductor_value = 0.3 + size*4.2
```

To pass measured data to a  
parameter (an output equation)

```
x = Device IV:IDC(IVCURVE@VSweep)
```

Built in functions available

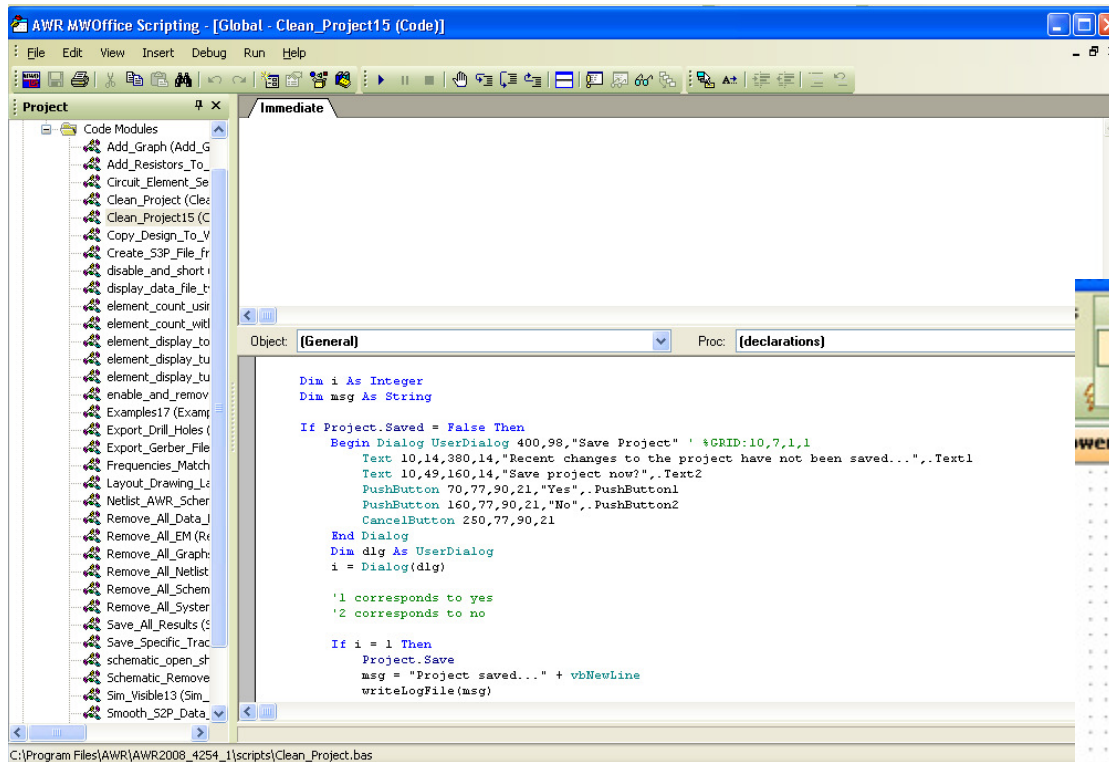
```
new_x = stepped(0,4,0.25)
```

**Note: Can also create a function by using scripting.**

# User Written “Code” in MWO - 2

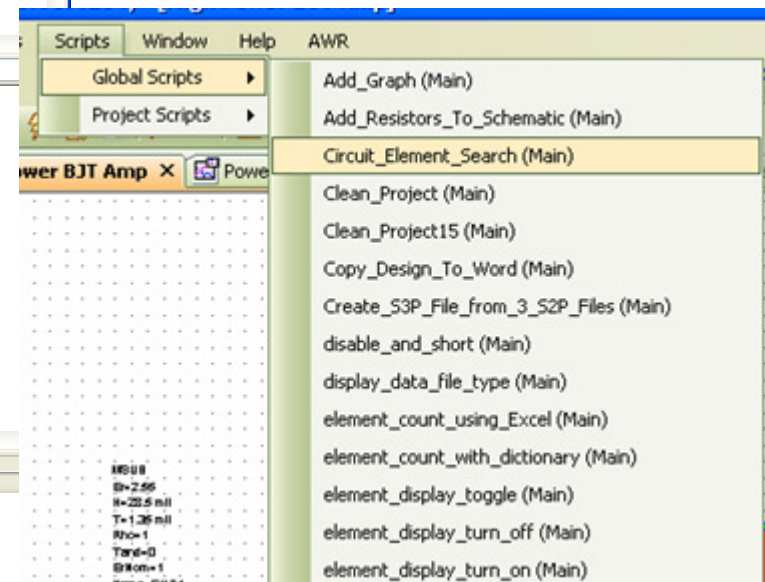
## Scripting

Used for general operations in MWO and the file system.



Scripts can be run from menu

- Global > Use in all projects.
- Project > Goes with project.



Scripts are written in Visual Basic.

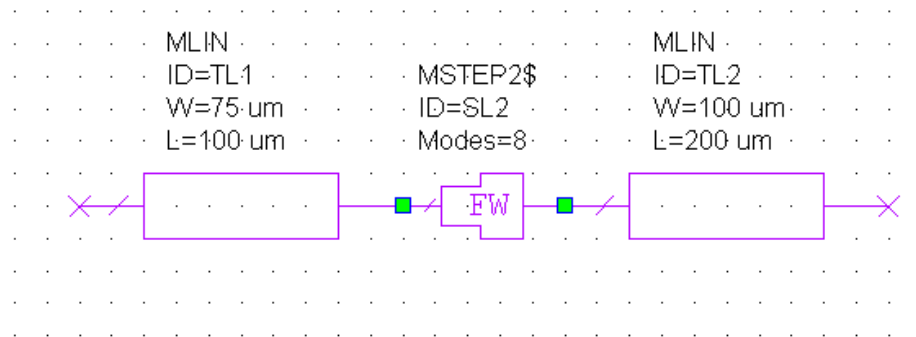
- Built in editor with debugger.

# User Written “Code” in MWO - 3

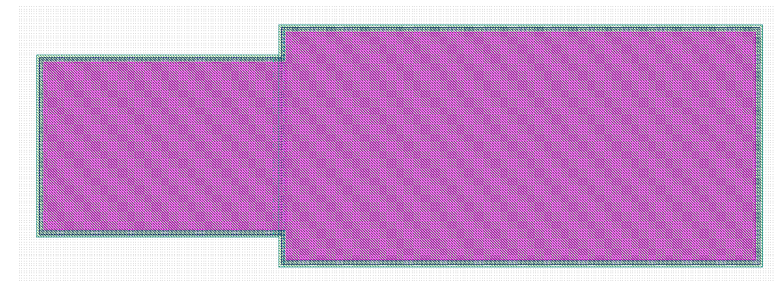
## Custom Library (PDK) Code

Used for writing compiled models, PCells, and Bridge Code.

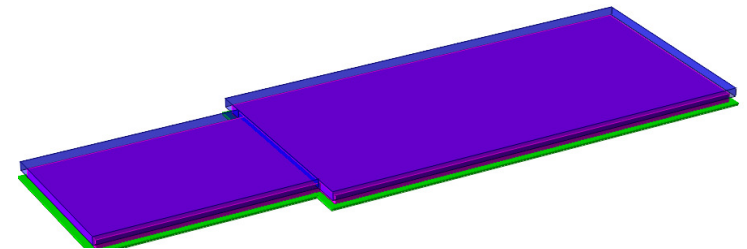
**Note: This code is written in C++.**



The parameters (W,L) change layout and model.



Bridge code allows “smart” connections.



# Scripts

## Accessing Scripts

**Global Scripts** – are saved on your computer.

**Project Scripts** – are saved with the project.

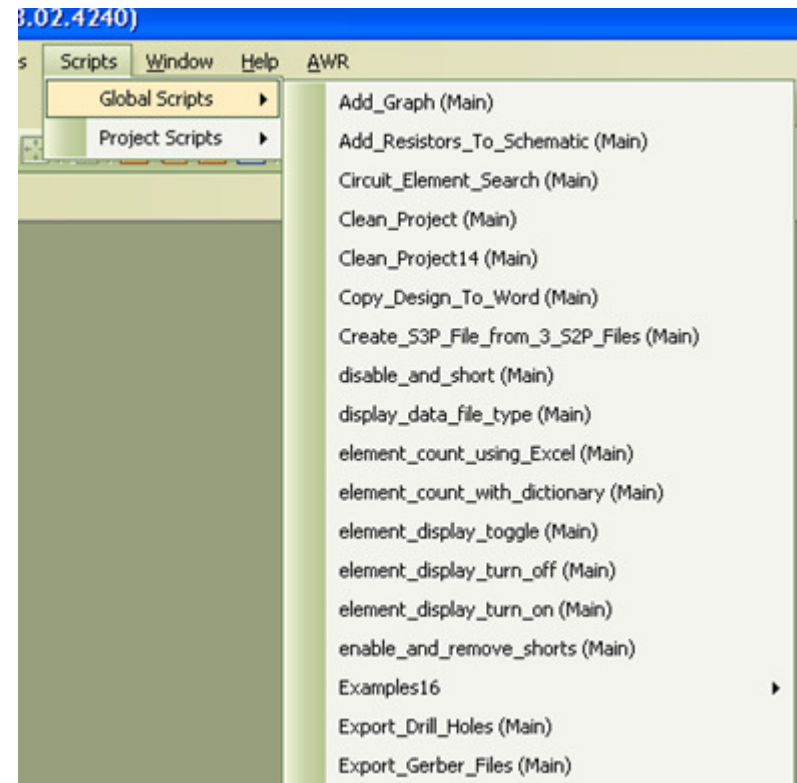
**Note: Global Scripts** – are saved in:

- **Scripts come with software. In install directory C:\Program Files\AWR\AWR2008\scripts**
- **The Examples.bas file – has many scripts in it.**  
**In any directory specified in your user.ini file: Help > Show Files/Directories > user.ini.**

### [File Locations]

**Projects=C:\john\awr\_projects\tmp**

**Scripts=\$DEFAULT;C:\john\awr\_projects\scripts;  
C:\john\awr\_projects\training\subversion\Training**



**Prior to version 8 – global scripts were saved in a global.mws file.**

- MWO/AO Users Guide - Chapter 13 and Appendix A
  - Brief discussion on equations, the scripting environment and functions. Section 13.5 has built-in functions and syntax.
- API Programming Guides –
  - Scripting/API Guide: Details of VBScript as applied to MWOOffice. (Not an intro to VBScript and programming.) [On the CD.](#)
  - Reference Guide: The schema and listing of the various object classes and methods. [On the CD.](#)
  - Sax Basic Reference Guide. [On the CD.](#)
  - Scripting Functions Guide. [On the CD.](#)
  - Adding Functions to the AWRDE through VB
- Info on VBScript on the Web
  - Microsoft has a lot of training.
  - <http://msdn.microsoft.com/en-us/default.aspx>
  - Go to the learn Tab ... and Visual Basic.

# Information and Help - 2

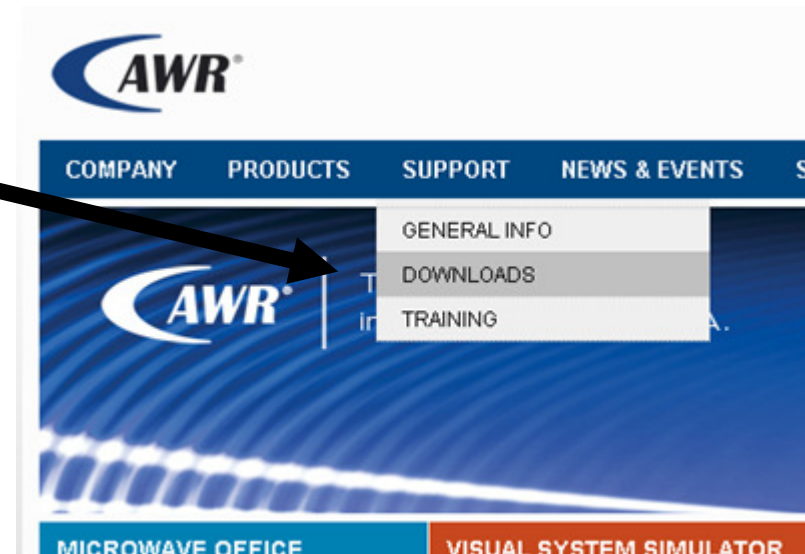
## API Documentation

Go to: [www.awrcorp.com](http://www.awrcorp.com)

- Go to downloads

It wants your email and password.

- You can get a password by filling out the “I’m and Existing Customer...” form.



### Downloads

Downloads		
Products	Wizards	Documentation
API Documentation	Vendor Libraries	Foundry Libraries
Description	Version	Last Updated
AWR Design Environment Scripting/API Guide	7.5	8/22/2008
API Programming from C++	7.5	11/17/2005
API Reference Guide	7.5	5/30/2007
Adding Functions to the AWR Design Environment through Visual Basic	7.5	11/17/2005
API Reference Guide	7.0	12/7/2006

Go to API  
Documentation  
Tab

# Example Scripts

There are over 160 scripts already written.

In KnowledgeBase ...



## Scripts

[2 Port Citi to Touchstone](#)  
[3 Port Citi to Touchstone](#)  
[4 Port Citi to Touchstone](#)  
[ACE\\_Component\\_Count](#)  
[Add\\_Graph](#)  
[Add\\_Name\\_and\\_Print\\_Graph](#)  
[Add\\_Resistors\\_to\\_Schematic](#)  
[Add\\_Time\\_Date\\_Project\\_Name](#)  
[Atan2](#)  
[BOM\\_Example](#)

## Article Listing

To browse for specific items, click on one of the following article types:

[Application Notes](#) , [Examples](#) , [Licensing](#) , [Questions](#) , [Scripts](#) , [Videos](#)

Scripting



# List of things done in Scripts

- Circuit Synthesis
- Graph formatting
- Test XML Library Files – script places every element in the XML Library onto a schematic for testing purposes.
- GUI Wizard
- Data Processing
- Project Documentation
- Layout Processing
- Clear Optimization/Yield/Tuning Variables

# Acronyms and Glossary

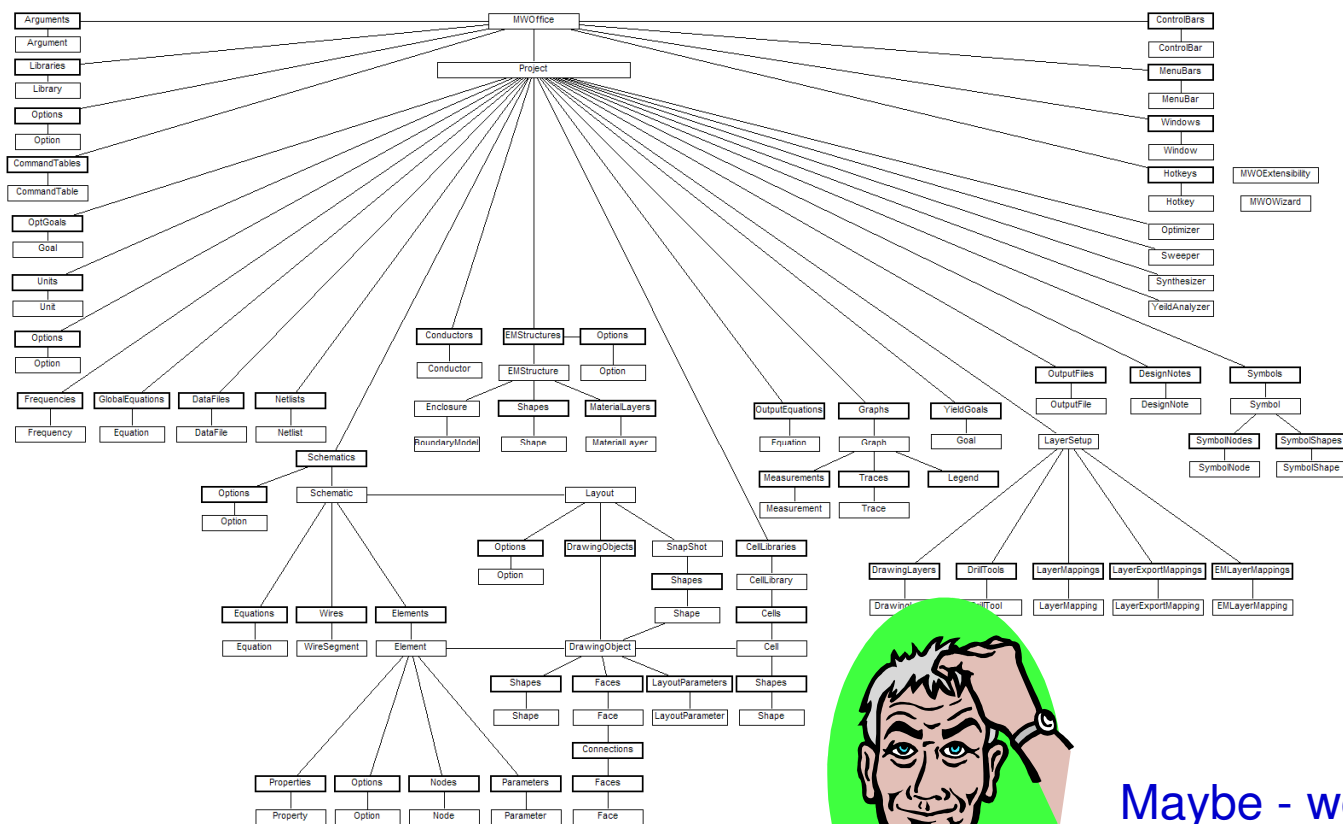
- **API** - Application Programming Interface
  - Environment to program MWO easily.
- **COM** - Component Object Model
  - Allows programs to “talk” to each other...for example, MWO and Excel.
- **Sax Basic**
  - The language we (normally) use for scripting in MWO.
- **Schema**
  - Organization of the various objects that can be used in programming - for example: graph, schematic...
- **AWRDE** - The AWR Design Environment
  - i.e. MWO!
- **OOP** - Object Oriented Programming

# Objects Oriented Programming (OOP)

- **Classes of Objects** are things (nouns!) we work with.
  - Examples in MWO of Classes: Schematic, Graph, Shape in a Layout, ...
  - Examples in Excel: Workbook, Sheet
- An **Object** is an instance created from a Class
  - Example: Schematic1 is an Object of Class Schematic.
- Objects have **Properties**:
  - Example for object Shape2: Vertices, Drawing Layer, ...
- **Methods** - What we do to objects (verb!)
  - Example for object Schematic1: Create, Delete, Export, ...
- **Collection** - is a group of the same class of objects
  - Example: Collection Schematics - The group of schematics in the current project.

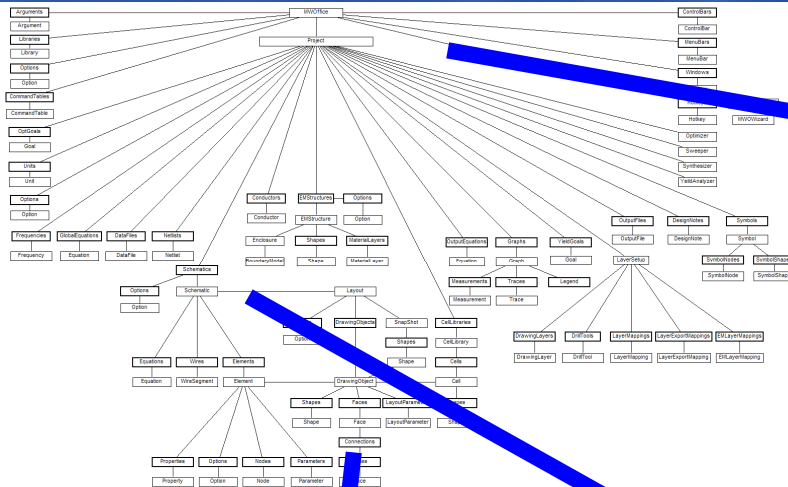
# The AWR Schema

## In the Reference Guide Tells us all the Classes of Objects in MWO



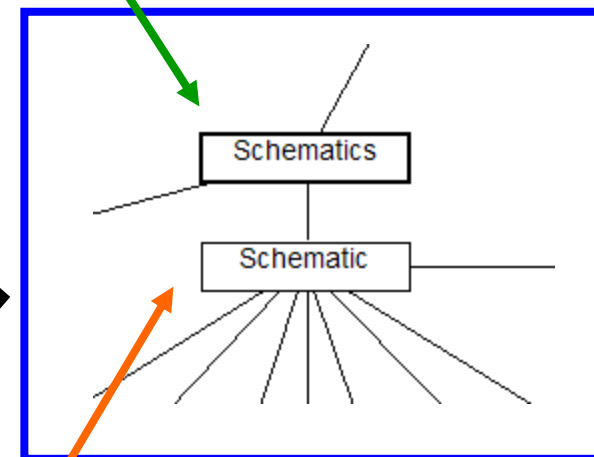
Maybe - we'd  
better zoom in!

Advancing the  
wireless revolution™  
[www.awrcorp.com](http://www.awrcorp.com)



The diagram illustrates a hierarchical structure. At the top is a box labeled "MWOOffice". Below it is a box labeled "Project". A vertical line connects "MWOOffice" to "Project". An orange arrow points from the top left towards the "MWOOffice" box. Below "Project", many lines radiate outwards to a large number of small, unlabeled boxes, representing individual projects or tasks.

```
graph TD; Equations --> Equation; Wires --> WireSegment; Equation --> Ports; WireSegment --> Ports; Ports --> Port;
```



Which contain one or more objects of the **Class** Schematic

# The AWR Schema - 3

## In the Reference Guide

### Lists all Collections and Classes of Objects

#### Objects

<a href="#">Application</a>	The MWOOffice application object.
<a href="#">Argument</a>	An MWOOffice Argument object.
<a href="#">Arguments</a>	A collection of MWOOffice Argument objects.
<a href="#">Attribute</a>	An MWOOffice Attribute object.
<a href="#">Attributes</a>	A collection of MWOOffice Attribute objects.
<a href="#">Axes</a>	A collection of MWOOffice Axis objects.
<a href="#">Axis</a>	An MWOOffice Axis object.
<a href="#">Boundary</a>	An MWOOffice Boundary object.
<a href="#">BoundaryModel</a>	An MWOOffice BoundaryModel object.
<a href="#">Cell</a>	An MWOOffice Cell object.
<a href="#">CellInstance</a>	An MWOOffice CellInstance object.
<a href="#">CellInstances</a>	A collection of MWOOffice CellInstance objects.

**Tip: Collections end in an “s”.**

#### Example:

- Schematic - Class
- Schematics - Collection of Objects of Class Schematic

Scripting

#### Cell

An MWOOffice Cell object.

#### Properties:

Name	DataType
Name	String
Shapes	IShapes
Boundary	IBoundary
Windows	IWindows
Instances	ICellInstances

#### Methods:

##### Activate

Description:	Activates the objects.
Returns:	Void
Syntax:	Sub Activate()

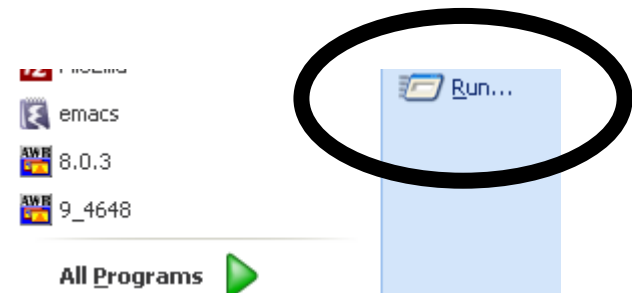
#### NewWindow

# Scripting Install – Possible Issue

We are running **version 8**.

• **If you never had version 7** on your computer ... you have to carry out the following procedure to fix the scripting editor ( ... so Intellisense is enabled).

1. **Open up the command prompt**  
- Click Run ... and type: `cmd`



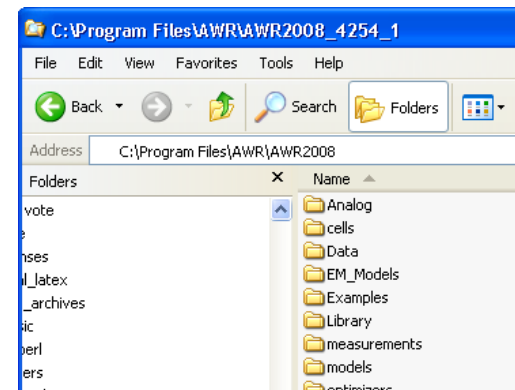
2. **Go to the 8.0 install directory – usually it's:**

**C:\Program Files\AWR\AWR2008**

**So type: `cd C:\Program Files\AWR\AWR2008`**

**Tip – To type this easily ... go to the Windows Browser – and copy / paste the path in the command window.**

3. **Type “`regsvr32 sb6ent.ocx`”**



# Scripting Help – Possible Vista Problem

## Problem:

If you are running Vista ....

- Typing F1 in the scripting editor – might not bring up the Help file.

## Solution:

- You can get a download of WinHlp32.exe from Microsoft at:  
<http://support.microsoft.com/kb/917607>

# Exercise: List the Schematics

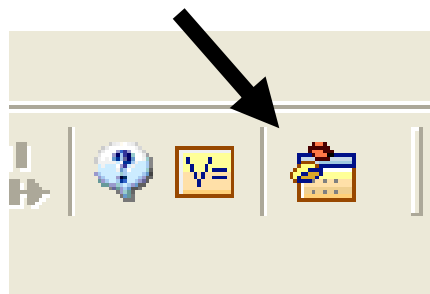
- Write a Program to List the Schematics in a Project

**Open up the project: “First\_Program”.**

## Open up the Scripting Editor

To open up the Editor:

- Tools > Scripting Editor
- Alt-F11
- Use the icon:



Standard Toolbar Scripting

Tip: If you can't see it - look in Tools > Manage Addins.

Make sure AWR Scripting IDE is checked.

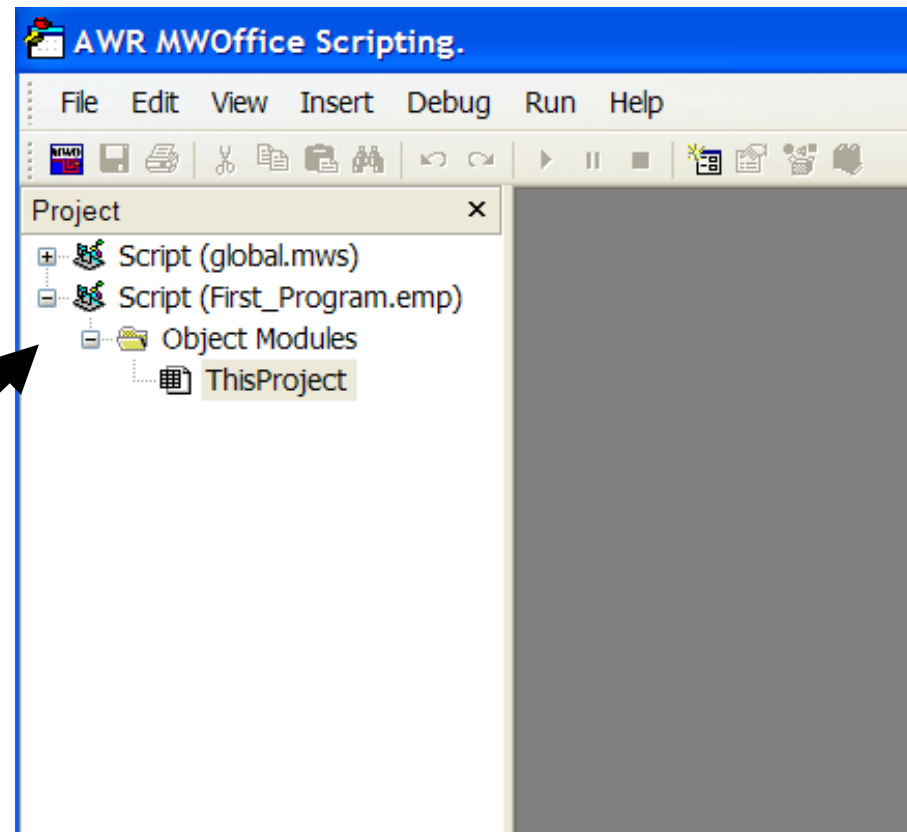
# The Scripting Editor

Global Scripts go here.

Note: All Users See these  
for every Project. They are  
stored in global.mws

Project Scripts go here.

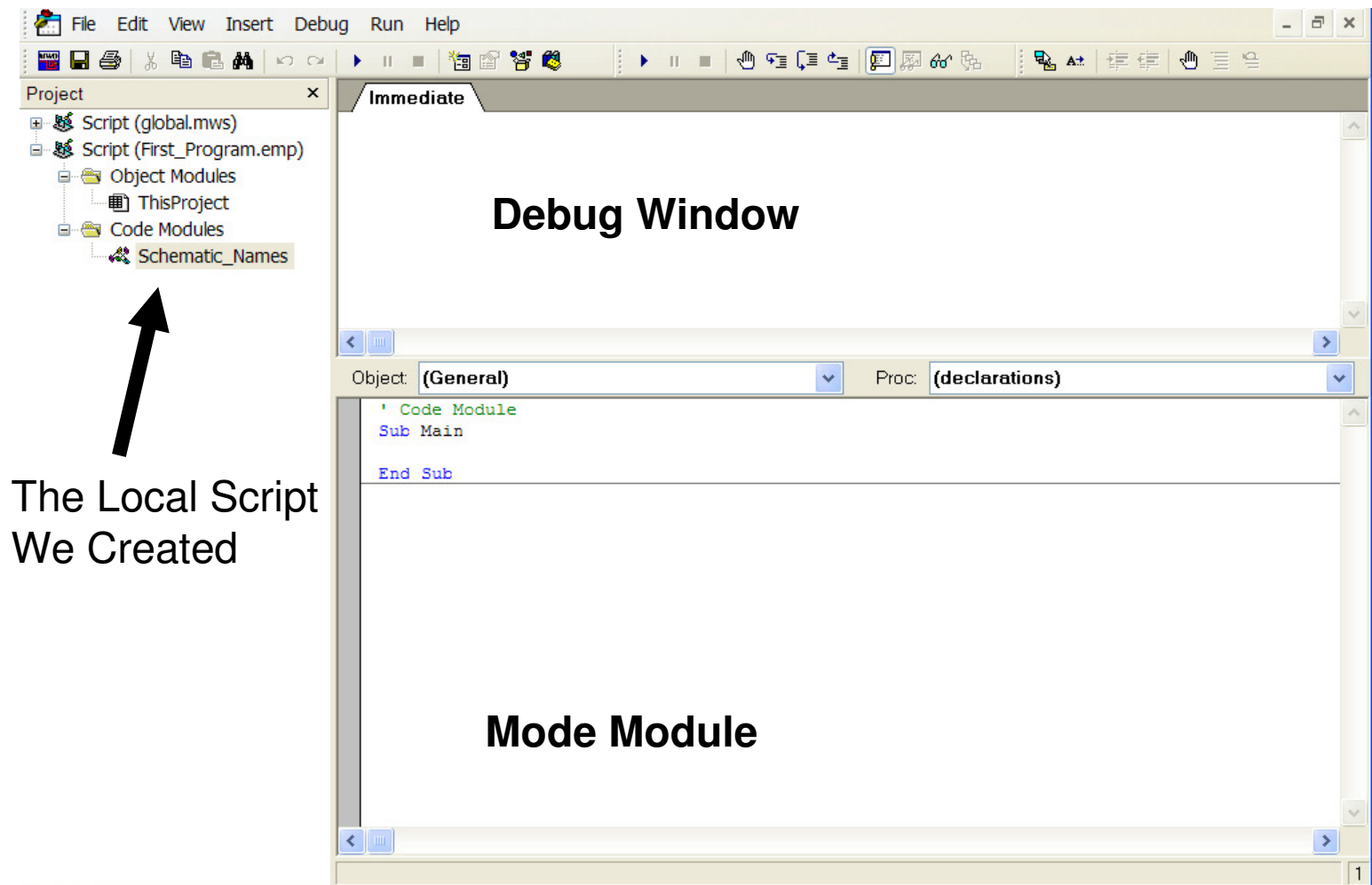
Note: These are stored in  
this project only.



**Create a new project script.**

- Right Click (RC) on Object Modules > Insert Modules**
- Rename it to Schematic\_Names**

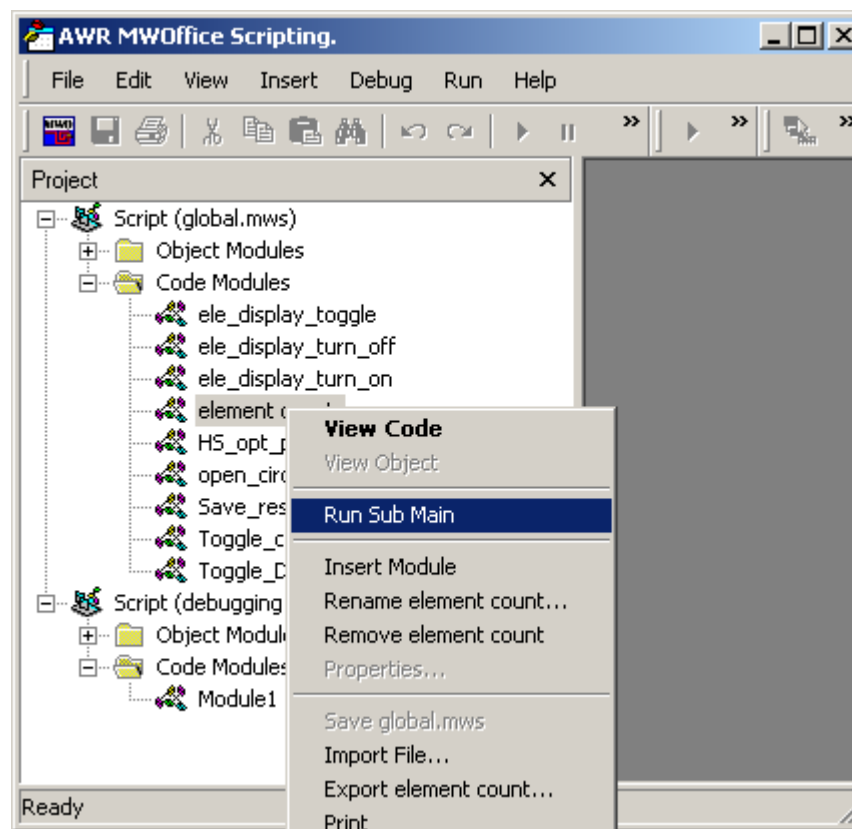
# The Scripting Editor - 2



The Local Script  
We Created

# How to Run Scripts

## Method 1: Right click on the name of the script and select Run Sub Main



# How to Run Scripts - 2

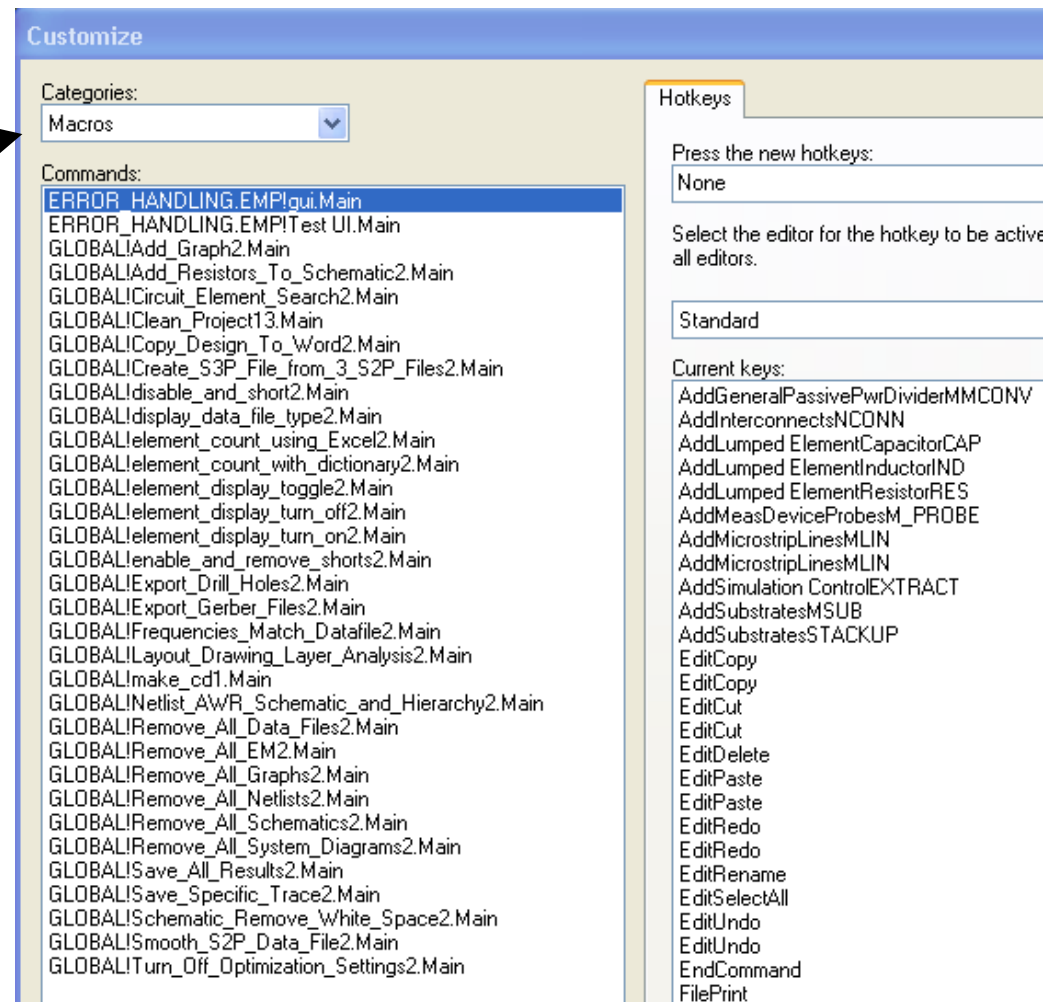
**Method 2: Have the code module open and put the blue Run button (looks like a VCR play button)**



# How to Run Scripts - 3

**Method 3: Assign script (or called macro) to a Hot Key, Toolbar, or Hot Key**  
– **Hotkey: Go to Tools > Hotkeys.**

**Category –  
Macros has the  
scripts.**



**Assign  
the  
Hotkey  
here.**

# Exercise: List the Schematics - 2

## The Strategy for Our Code

1. Look at the **Collection Schematics**
2. For Each **Object of Class Schematic** in the **Collection Schematics**
  - Find the **Schematic's** Name (A **Property**)
  - Print it to the Debug Window

Next Schematic

So ...

- How do we make an object from a class?
- How do we work with a collection of objects?
- How do we get the object's properties?
- How do we print?

# Miscellaneous SaX Basic Tips

- Comments Begin with ‘
- Use **Option Explicit**
  - Reduces errors as forces StrongTyping.
- Pick names that are easy to understand.

**Tip: Strong typing means the variable must be declared in a Dim statement....**

- **Dim foo As Long**
- **Dim bar As Schematic**

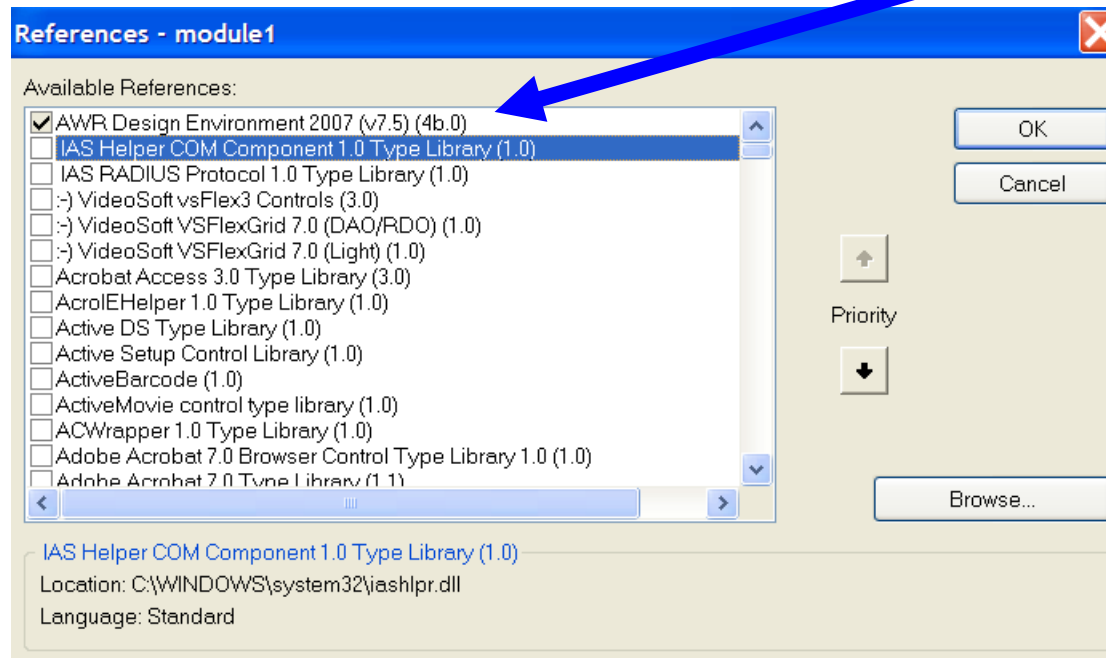
**This is done to prevent typos.**

- Typing fo0 will give “Error: fo0 not declared”.
- If you don’t have Option Explicit – it will create a variable fo0.

# Finding the Classes of Objects

- Each program (including MWO) using the COM interface has Classes of Objects.
- The available Programs can be found in the References Browser.

We only need AWRDE.



Open it by:

- Edit > References
- or Icon



**Note: You must have a module open to see the reference browser.**

# Organization of Code

```
' Code Module  
Option Explicit
```



**Add - Option Explicit**

```
Sub Main  
    Dim sch As Schematic  
  
End Sub
```

**Note:** Explicit is used to throw an error if we haven't properly defined something. It's not required ... but - it's a good idea!

This is a procedure - called Main.

**Note:** You always need Main - it's where the script (macro) starts. You can add other subroutines and functions if you wish.

**Add the: Dim sch As Schematic**

Created **object** sch of **Class** Schematic.

**Note:** The script starts by defining your objects **As classes**. You also **define variables here**.

**Tip:** sch is the name we gave our object ... it could have been named whatever you like.

# Getting Help in the Editor

## Object Browser

or F2 or View > Object Browser



**ActiveX Automation Members**

Back    `_Name = _String`    The Class or Data Type    Paste

Library: (All Libraries) ▼

Data Type: Schematic ▼

Methods/Properties:

- Export
- ExportNetlist
- Frequencies
- GridVisible
- Layout
- LockDiagram
- LockLevel
- LockUpdates
- Name**
- NamedConnectors
- NewWindow

Property: Name  
Value: String  
Dispatch ID: 0xFFFFF0

Close

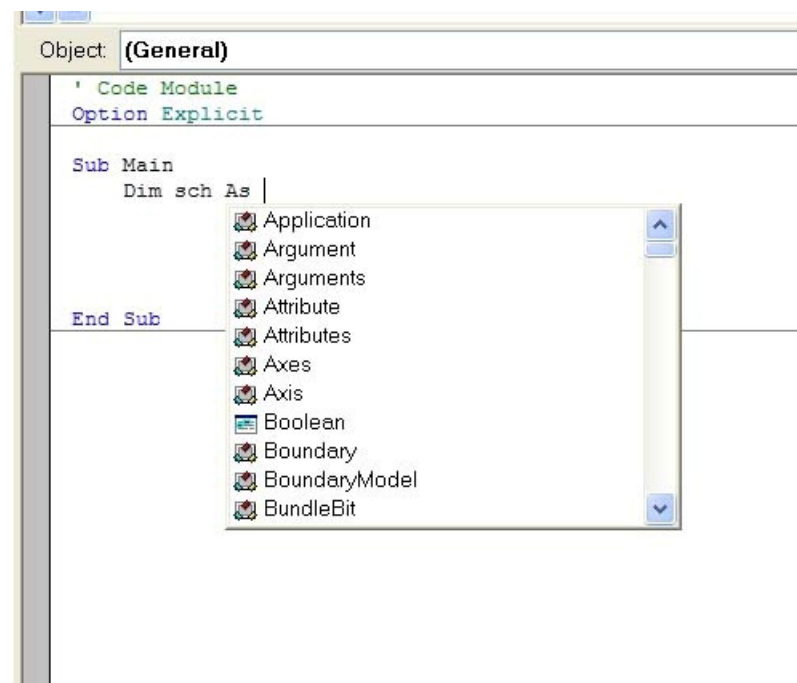
? [Help String](#)  
Returns/Sets the name used in code to identify an object.

The Methods and Properties  
- Returns a variable of Type String

# Getting Help in the Editor - 2

## Intellisense

It will help you complete/pick the only available choices.

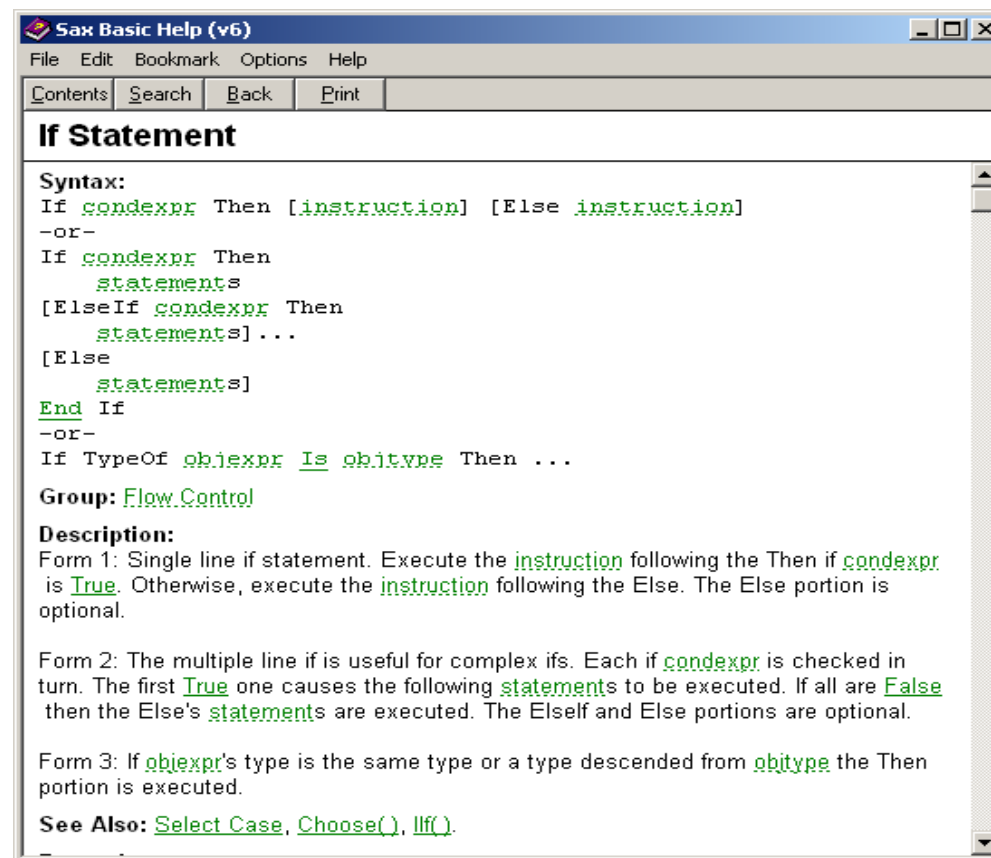


It automatically brings up the legal choices - as you start typing Schematic... it will complete it for you.

**Tip: Ctrl - Space will bring up Intellisense too.**

# Getting Help in the Editor - 3

- F1 for help.
- If cursor is over a recognized VB function name, help will go directly to that function.



# The Schema and Objects

Start at the top and  
work our way down...

For Our Schematic:

Class

MWOffice



Class

Project



Collection

Schematics



Class

Schematic

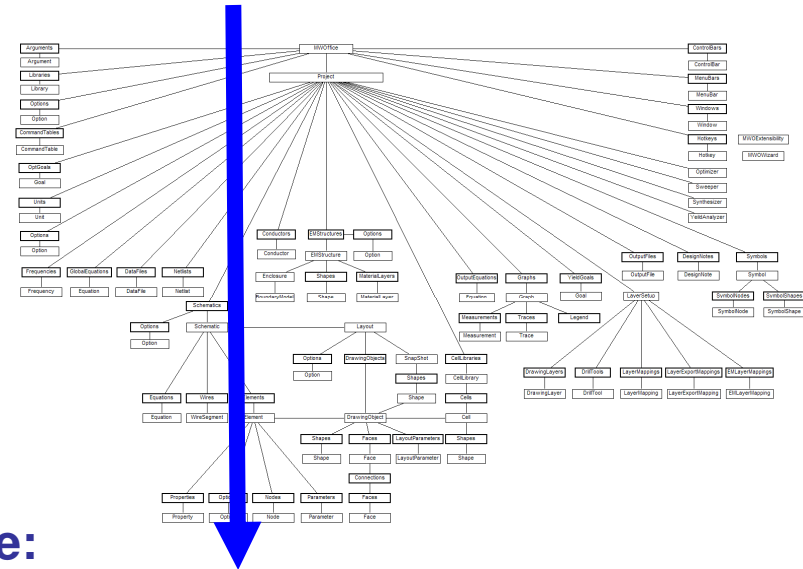
We Type:

MWOffice.Project

MWOffice.Project.Schematics

so ... Project.Schematics will work.

Scripting



**Tip: Normally we omit MWOffice** - in our definitions of objects and collections. It's assumed. But if you had another NameSpace - like Excel - you have to start explicitly with Excel.

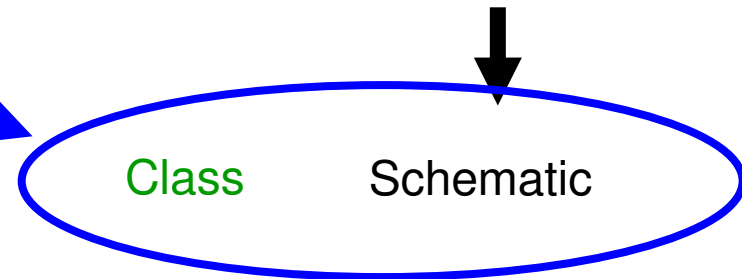


How do we get at the individual  
schematics in the collection?

Collection

Schematics

## Method 1:



### Schematics

A collection of MWOffice Schematic objects.

#### Properties:

Name	DataType	Description
Item	ISchematic	Returns a specific item of a Collection object either by position or by key.
_NewEnum	IUnknown	Used for internal support of For Each syntax.
Count	Long	Returns the number of objects in the collection.
ActiveSchematic	ISchematic	Returns the active schematic of a Project object.
Options	IOptions	Returns a reference to a collection of Option objects.
Exists	Boolean	Returns whether an item matching the specified criteria exists in the collection.

Count is a Property ... Number of objects.

Scripting

# Collections - 2

```
' Code Module
Option Explicit

Sub Main
    Dim sch As Schematic
    Dim numsch As Long

    numsch = Project.Schematics.Count
    Debug.Print numsch
End Sub
```

We have to define a variable of type Long.

**Note: Long is an integer.**

**Note: Notice the path - Project.Schematics ... and Count is a property of the Collection Schematics.**

The Debug.Print will print the value numsch in the debug window.

# Collections - 3

You Can get at an item of a collection - if you know which item it is...

```
numsch = Project.Schematics.Count  
Debug.Print numsch  
Debug.Print Project.Schematics.Item(2).Name
```

Here we are getting the second item in the Collection of Schematics - which is a Schematic Object.... then, Schematic has a property Name - which we can print.

**Tip: It is a pain to loop through all the Collection elements using something like –**

- **For I = 1 to N ...**

**Because you have to know number of elements in Collection.**

**Instead ... use For ... Each...**

## Method 2:

A nice control statement for Collections is - **For Each ... In ...**

- It will loop through all the objects in the Collection ... and you don't have to know the number ahead of time!

```
' Code Module  
Option Explicit
```

```
Sub Main  
    Dim sch As Schematic  
  
    Debug.Clear  
  
    For Each sch In Project.Schematics  
        Debug.Print sch.Name  
    Next sch  
  
End Sub
```

This clears the debugger window.

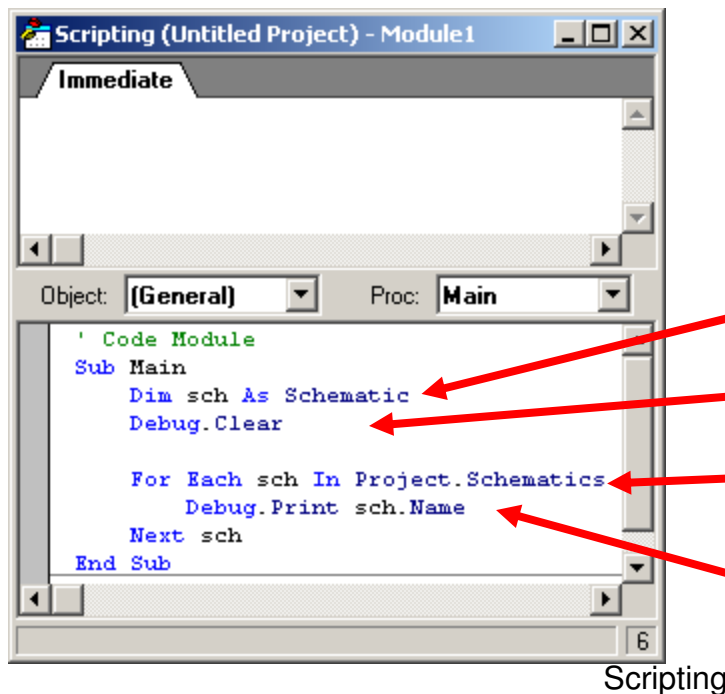
Immediate

```
Attenuator  
Distributed Filter  
Lumped Line
```

The loop assigns Object sch to each schematc in turn ... and prints it.

# Debugging

- Sax Basic has its own debugging tools.
  - Debug.print statement will print data to the debug window
  - Can set break points and then continue to next break points
  - Can step into our around subroutines
- We used it already in our example.



Dimension variable as class schematic

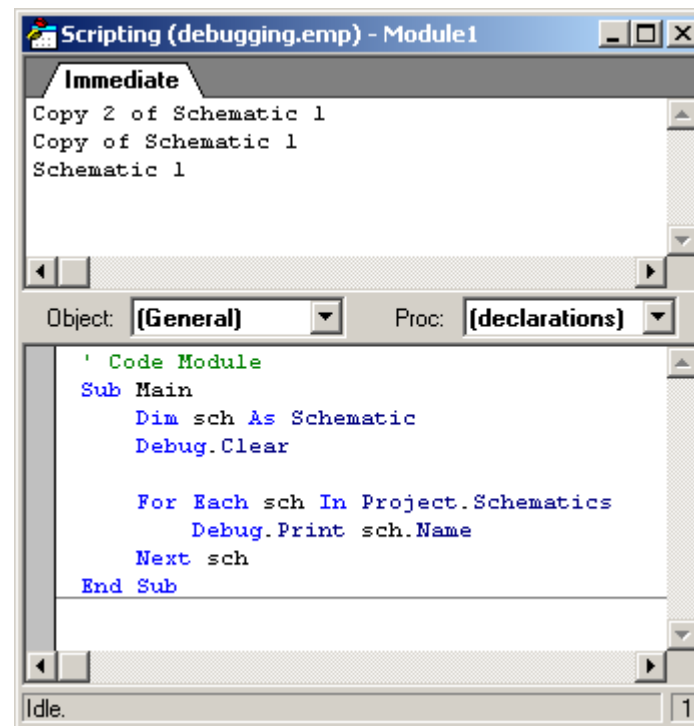
Clear debug window

Loop through all schematics in project

Print schematic name in debug window

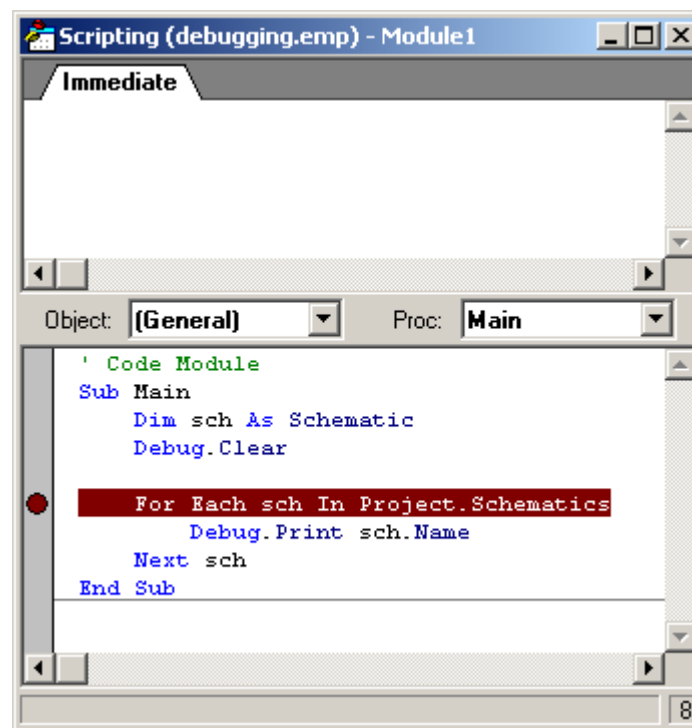
# Debugging Continued

- When this script is run, all of the schematic names will show up in the debug window.



# Debugging – Break Point

- You can easily add break points, select the location and hit F9 or use the **Debug** menu.
- Below shows a break point before the script is run set to stop at the beginning of the FOR loop.

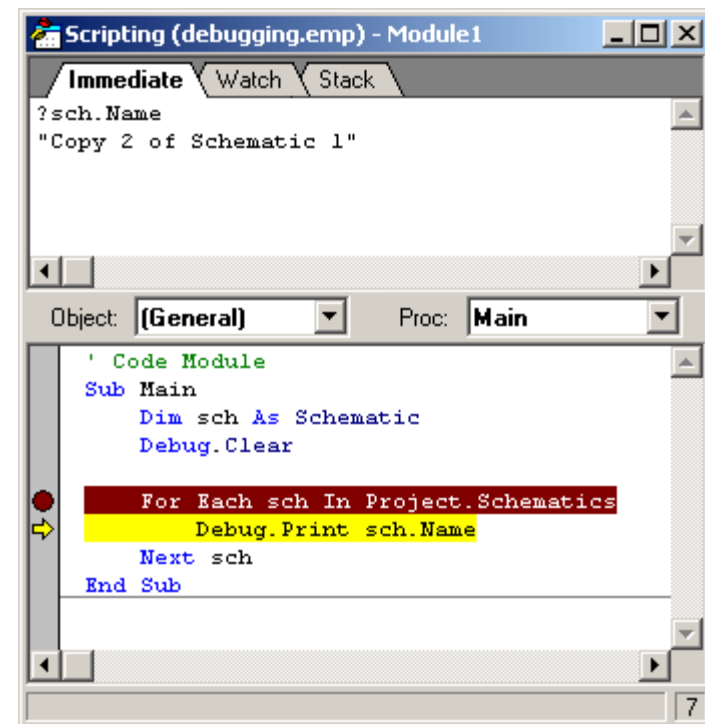


# Debugging – Query Variables

- When a script is holding at break point you can query the value for variables.
- In this example, F8 is used to step into the FOR loop. Then type “?sch.Name” in the debug window on the **Immediate** tab to query the value for sch.Name. This value is then displayed.

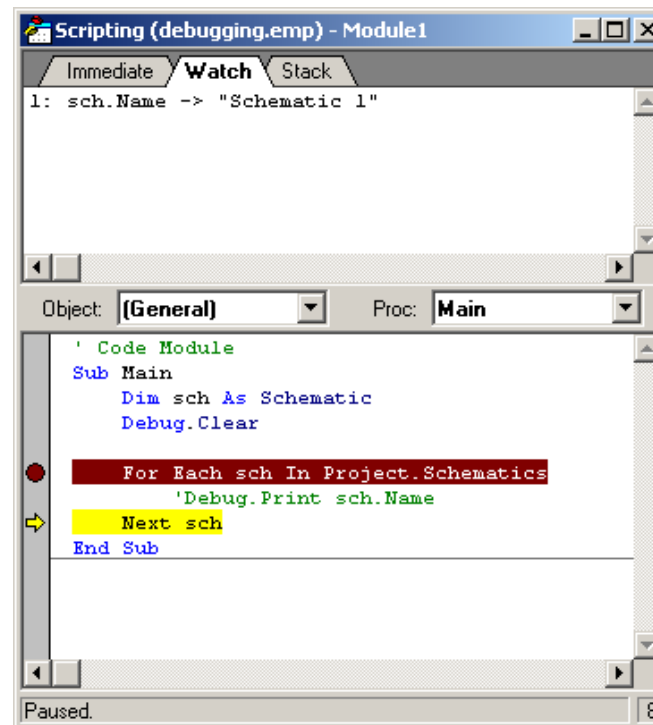
Use the () to get values of a vector where you would type the index of the vector between ().

You can also put cursor over the variable to see it's current value.



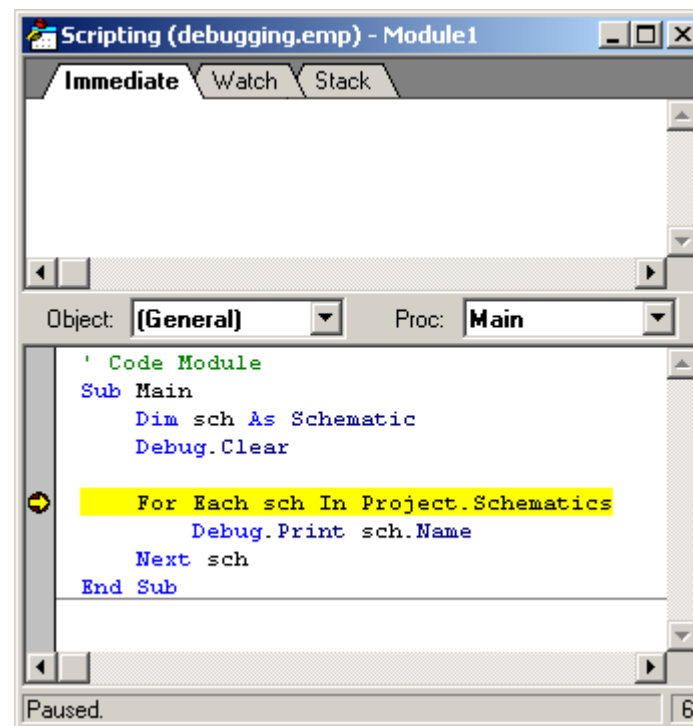
# Debugging – Query Variables

- Alternately, you can enter variables in the debug window on the **Watch** tab. This will constantly display a variable value as code is executed.
- Just type in the variable name in this area at it will be displayed as you step through code.



# Debugging – Break Point

- When the code runs it will stop at this point.
- You will see the break point turn yellow.
- Use the F8 key to then step through the code



- **Debug.Print**
  - Prints values to the debug window.
  - Smart enough to figure out data type - so long as scalar or string.
- Normally, Debug throws in a line feed... ; suppresses
  - Example: `Debug.Print 3 ; 4; 5`
- **Debug.Clear**
  - Clears the Debug Window.
  - Usually put near top of script.

# Variables and Data Types

- We still often have need for traditional variables - esp. when we work with functions.
- Must be defined with Dim statement (assuming you have Option Explicit turned on).
- Most Commonly Used Types of Variables:
  - Integer (A 16 bit integer)
  - Long (A 32 bit integer)
  - Single and Double (16 bit and 32 bit floating)
  - String (Type in “ “ to make a string. Can use & to concatenate.)
  - Boolean
  - Complex (This is defined in MWO's Data Types.)
  - and of course ... Object

Actually - We're not telling the whole story here ...  
(Variants in a few slides)

- We can define arrays:
  - Example: `Dim mywire(5) As Trace` ... An array of 5 elements... of class Trace.
  - Example: `Dim the_values(3,4) As String` ... A two index array of dimensions 3 X 4 of type String.
- Later - we can access them as normal:
  - Example: `the_values(0,2) = "big value"`
- Note - Arrays start with value 0 not 1.
  - `Dim myarray(1,1)` is a 2 X 2 array, with each index 0 and 1.

# Arrays - Redimensioning

- A common problem -
  - I don't know the size of the array until later in the script after I get some user input.
- First - Define a dynamic array
  - Example: `ival()` As **Integer** Number of elements not defined
- Later - Use `ReDim`
  - `ReDim ival(4)` As **Integer**
- You can keep old data with the Preserve Statement
  - `ReDim Preserve ival(4)` As **Integer**
  - This keeps the `ival(0) ... ival(4)` values if you later make the array bigger.
  - Otherwise you lose all your data.
- **Note: You will lose data if you downsize an array!**
- **Note: With Multidimensional arrays you can only resize the last index. (Can add a column, not a row.)**
- Example Script - `ReDim_Array`

- **For ...** Simple Loop (Note - Automatically steps)

For Num = First To Last  
statements  
Next Num

- **For...Each** Useful for Groups of Items

For Each **var** In **Items**  
- statements  
Next **var**

- **Do ... Until** Keep looping until condition met.

Do  
- statements  
Loop Until conditional expression

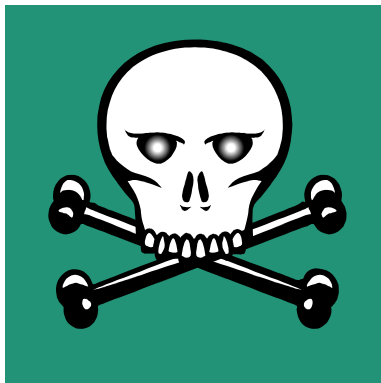
**Note: Can also use While instead of Until.**

# Control Statements - 2

## Other Statements

- For ... Next Details Omitted
- If Then Else Details Omitted
- Select Case Picks choice according to value of expression.
- And of course when all else fails ...

## GoTo



**But ... it's dangerous to use it – as you make your code unreadable and difficult to understand the logical flow.**

**And other scripters will scorn you!**



# Exercise - Number of Resistors in Schematics

- Want number of resistors in each schematic, and total number of resistors in project.
  - Don't need to include subcircuits, or netlists.
- User to be able to pick schematic in selection box.
- Number of resistors in schematic and project to appear in new box.

## We Need to Figure out:

- How to get at elements in schematics.
- How to find out which are resistors.
- Create a selection box, and output text box.

**Answer the following questions by looking at the Schema and Object descriptions.**

- How do you get the number of Schematics?
  - Hint: Look at the properties of the collection schematics.
- What is the path/hierarchy to get to a Class of type Schematic?
- How do you go from Schematic to Element?
- What Properties of an Element might help us find out if it is a resistor?
  - Hint: We are looking for something to decide if it is a resistor, so we can sort on it.

# Elements in Schematics - 2

## Conceptual Flow of the Code

1. Define All the variables and objects we need.

**Dim sch As Schematic**

...

2. Initialize the Counting Variables.

**totalres = 0**

...

3. Get the number of schematics and re-dimension the arrays.

**numsch = The number of schematics**

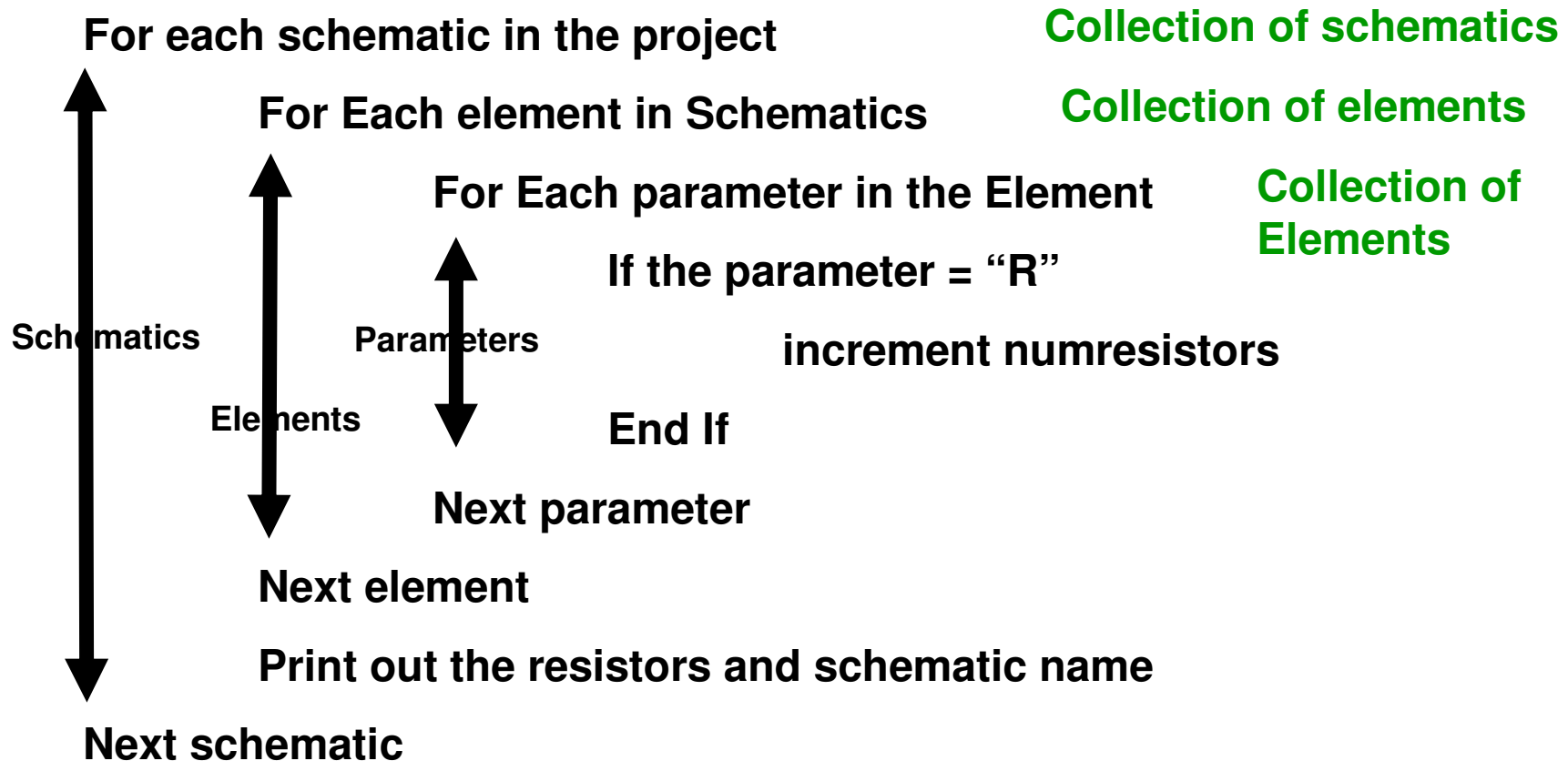
**ReDim schnames(numsch-1) As String**

...

**Tip: You could use a dictionary here for names and number of resistors ... more on this later.**

# Elements in Schematics - 3

4. Loop through the schematics.... Loop through each element in a schematic ... Look at Parameters in Each element ... See if it's a resistor ...



# The Actual Code - 1

**This is the code PartA\_Res.bas**

## **1. Define All the variables and objects we need.**

**Option Explicit**

**Sub Main**

```
Dim schnames() As String      ' The arrays and objects
Dim resvalues() As Integer
Dim sch As Schematic
Dim anelement As Element
Dim aparameter As Parameter
```

```
Dim totalres As Integer      ' Variables
Dim totalelements As Integer
Dim elementname As String
Dim numsch As Integer
Dim schindex As Integer
Dim numres As Integer
```

# The Actual Code - 2

## 2. Initialize the Counting Variables.

```
totalres = 0  
totalelements = 0  
schindex = 0  
elementname = "R"
```

## 3. Get the number of schematics and re-dimension the arrays.

```
numsch = Project.Schematics.Count  
ReDim schnames(numsch-1) As String  
ReDim resvalues(numsch-1) As Integer
```

# The Actual Code - 3

**4. Loop through the schematics.... Loop through each element in a schematic ... Look at Parameters in Each element ... See if it's a resistor ...**

```
For Each sch In Project.Schematics
    numres = 0
    For Each anelement In sch.Elements
        For Each aparameter In anelement.Parameters
            If elementname = aparameter.Name Then
                numres = numres + 1
            End If
        Next aparameter
    Next anelement

    schnames(schindex) = sch.Name
    resvalues(schindex) = numres

    Debug.Print schnames(schindex);" Number of Resistors is: "; numres
    schindex = schindex + 1
Next sch
```

# Elements in Schematics - 6

The finished code is module PartA\_Res.

```
Immediate Watch Stack
Number of Schematics is: 3
Schematic 1 Number of Resistors is: 2
bar Number of Resistors is: 2
go Number of Resistors is: 2
```

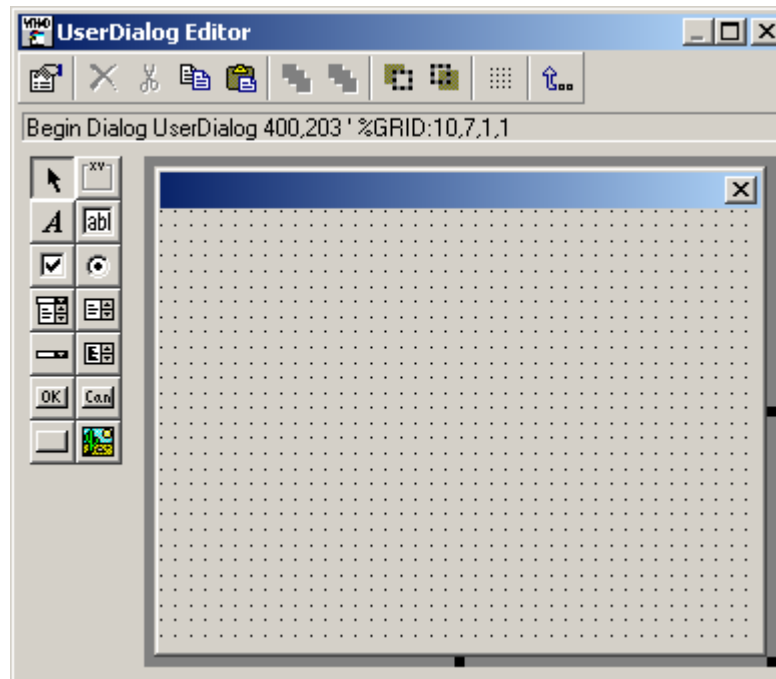
The results in the Debug Window.

Part B: Now let's add a user interface (UI).

We are going to use the UI builder to help us.

# UI Builder

We can graphically build the UI, and the code will be written for us.



**Insert > User Form** to bring up a blank UI builder.

After making your UI, and exiting the builder, the code will be inserted.

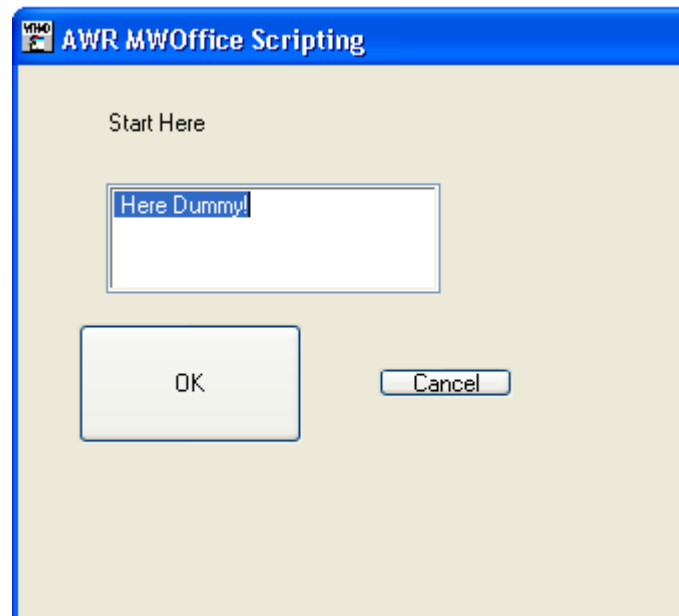
# UI Builder - 2

Let's start with a simple example first ...

Create a new script: **Code Modules > Insert Module.**

Rename it to: **“Test UI”.**

Our script will ask the user to input text into a dialog box, and print it out in debug window. There also is a cancel button and an OK button.



# UI Builder - 3

Start out by creating a string variable: usertext.

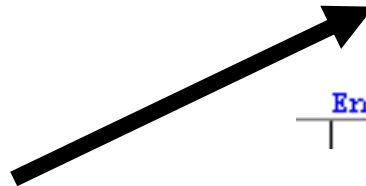
```
' Code Module
Option Explicit

Sub Main

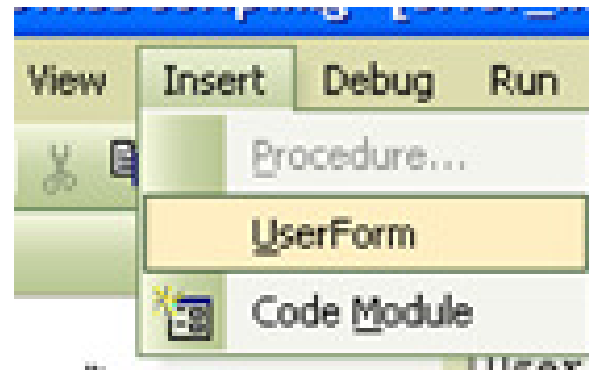
    Dim usertext As String

End Sub
```

Put the cursor here ...  
the user form will add  
the graphics code  
where the cursor is.



... then open up the  
UserForm dialog.

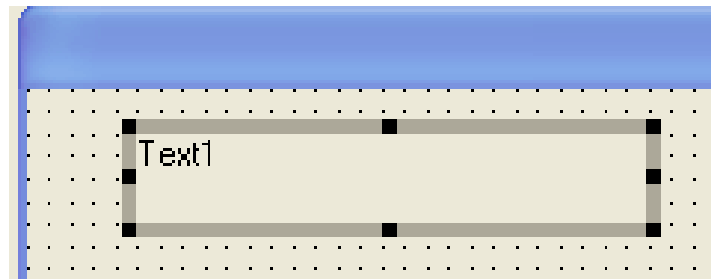
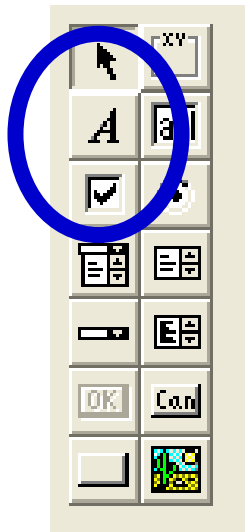


# UI Builder - 4

**We draw what we want the user to see.**

**The text is drawn in a box. Use the “A” to draw the box.**

**Note: This is used to put text on the screen. It has nothing to do with any input from the user.**

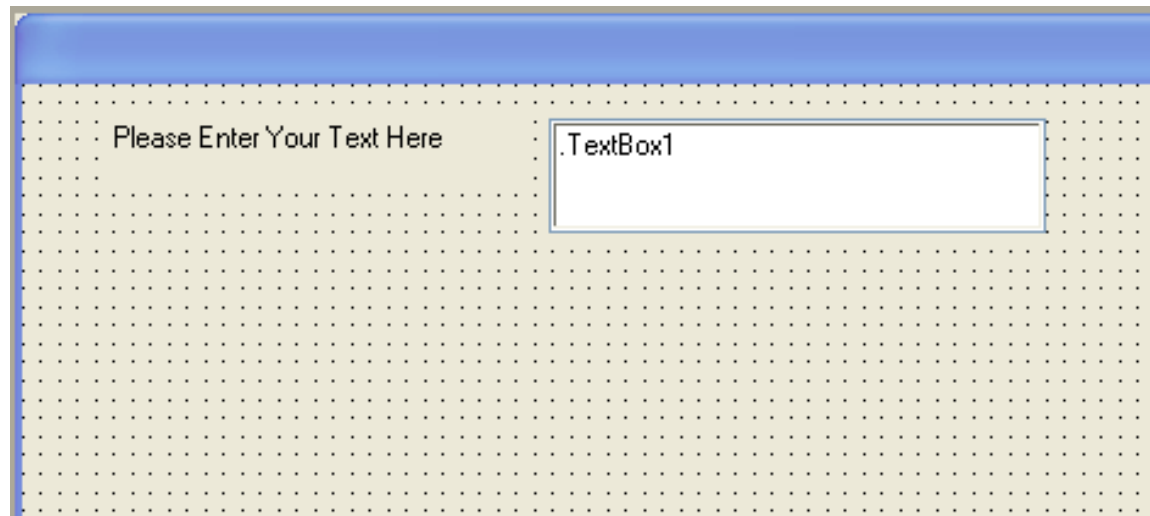
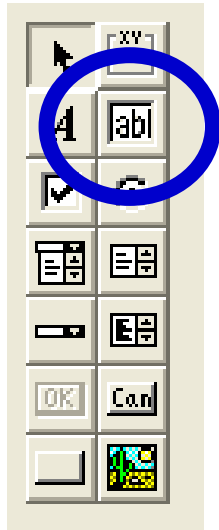


**Edit Text Properties**

Left	50	≤<
Top	14	≥>
Width	230	Close
Height	28	
Caption	Please Enter Your Text Here <input checked="" type="checkbox"/> Quoted	
Field	Text1	Left ▼
Comment	<input type="text"/>	

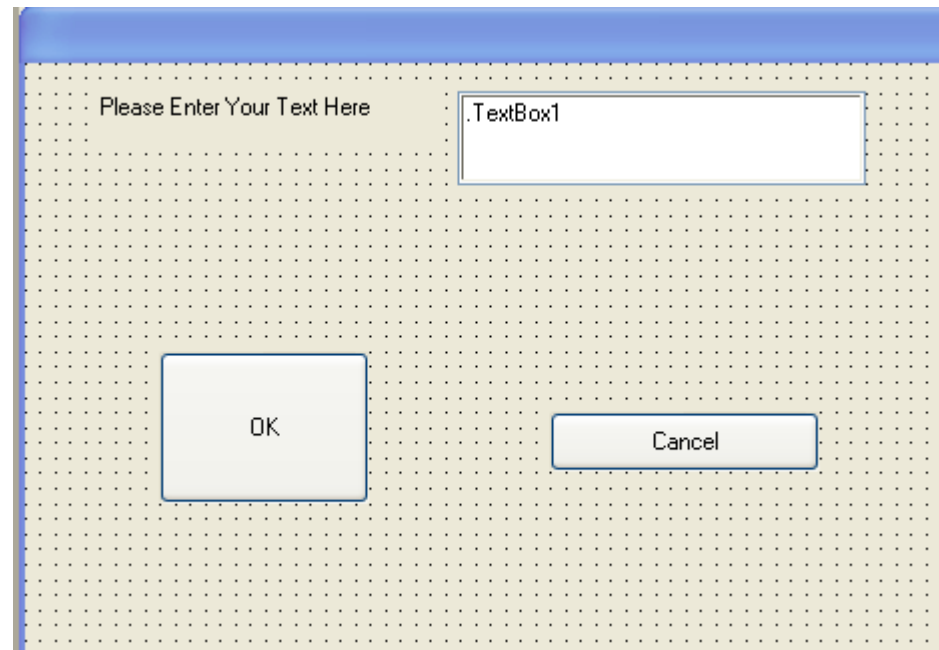
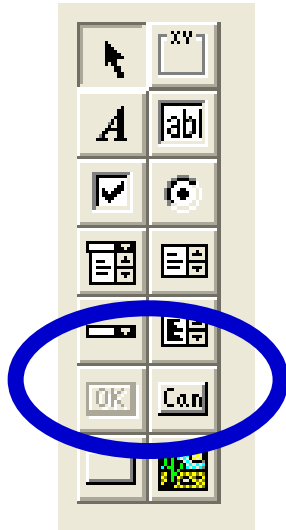
**RC the box and  
enter text in  
Caption.**

# UI Builder - 5



**A TextBox is placed on the form using the “AB” button. This is where text will be input by the user.**

**Add OK and Cancel Buttons.**



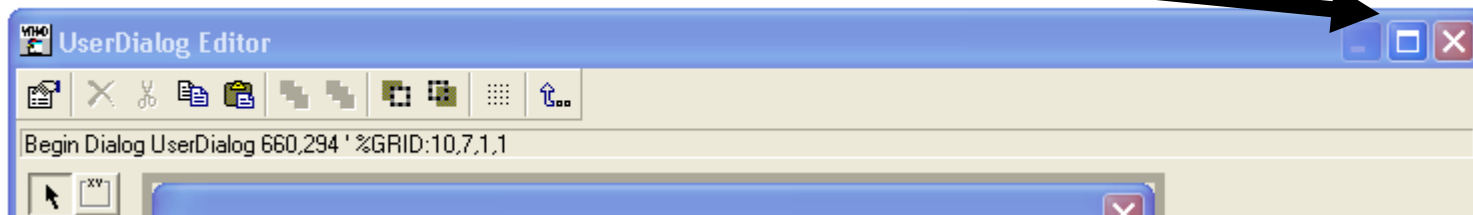
**OK – will close the dialog box and continue code execution.**

**Cancel – will close the dialog box and exit with an error.**

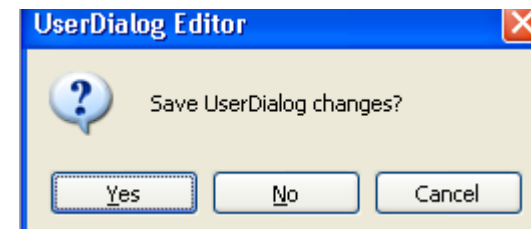
**Tip: We will be able to capture the error and use it.**

# UI Builder - 7

Close the UserDialog Editor...



Say Yes to the Save Changes.



And it inserts the  
code where the  
cursor was.

**Tip:** If you have the cursor  
in this part of the code ...  
opening Insert > User Form  
... allows you edit it.

```
Dim usertext As String

Begin Dialog UserDialog 660,294 ' %GRID:10,7,1,1
    Text 50,14,230,28,"Please Enter Your Text Here",.Text1
    TextBox 290,14,270,42,.TextBox1
    OKButton 90,133,140,70
    CancelButton 350,161,180,28
End Dialog
Dim dlg As UserDialog
Dialog dlg

End Sub
```

Can be thought of as a special class – UserDialog.

```
Dim usertext As String
```

Dialog Form  
class defined.

```
Begin Dialog UserDialog 660,294 ' %GRID:10,7,1,1  
Text 50,14,230,28,"Please Enter Your Text Here",.Text1  
TextBox 290,14,270,42,.TextBox1  
OKButton 90,133,140,70  
CancelButton 350,161,180,28
```

dlg is an object of  
this class.

```
End Dialog  
Dim dlg As UserDialog  
Dialog dlg
```

This creates (a method) the  
form.

```
End Sub
```

**Dim dlg As UserDialog** ... creates an object (here dlg) of class  
UserDialog.

**Dialog dlg** ... the method Dialog working on object dlg ... makes  
the userform.

Here ... we change the message.

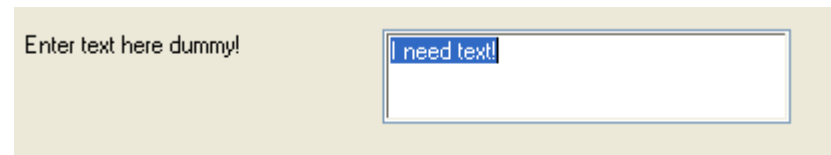
```
usertext = "Enter text here dummy!"
```

**Tip:** `.Text1`, ... etc.  
are properties that  
can be assigned to  
the object.

```
Begin Dialog UserDialog 660,294 ' %GRID:10,7,1,1
  Text 50,14,230,28,usertext,.Text1
  TextBox 290,14,270,42,.TextBox1
  OKButton 90,133,140,70
  CancelButton 350,161,180,28
End Dialog
```

Here ... an initial value is set for the TextBox.

```
Dim dlg As UserDialog
dlg.TextBox1 = "I need text!"
Dialog dlg
```



**Note:** `dlg` has been created with `Dim`  
before giving it the property  
`.TextBox1`.

**Note:** You can change the property  
name `.TextBox1` to ... whatever you  
want (with a `.` in front of it).

This prints the user input to the debug window.

```
Dim dlg As UserDialog  
dlg.TextBox1 = "I need text!"  
  
Dialog dlg  
Debug.Clear  
Debug.Print dlg.TextBox1
```

**Note:** The method Dialog has been used to create the form.

The user clicks the OK button, which closes the user form and continues code execution.

A screenshot of a user dialog form. It has a light beige background. On the left, there is a label "Enter text here dummy!". To the right of the label is a text box containing the text "User Entered this.".

Text typed in and OK button clicked.



Shows up in Debug window.

## Error Handling

If the user clicks the cancel button, an error is thrown.

**Note:** Many methods return a value ...besides carrying out their action (making the dialog form).

**Note:** Make sure you define errornumber as a Long.

**Dialog(dlg)** returns an integer ... 0 is an error – i.e. they hit the Clear button.

```
errornumber = Dialog(dlg)

If (errornumber = 0) Then
    Debug.Print "They hit the cancel button"
End
End If

Debug.Clear
Debug.Print dlg.TextBox1
```

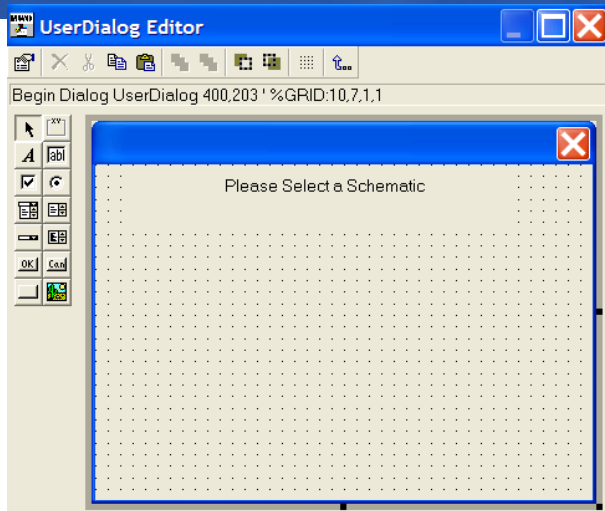
**Tip:** End ... ends the program. Otherwise it will keep executing.

**Go back to our Resistor project.**

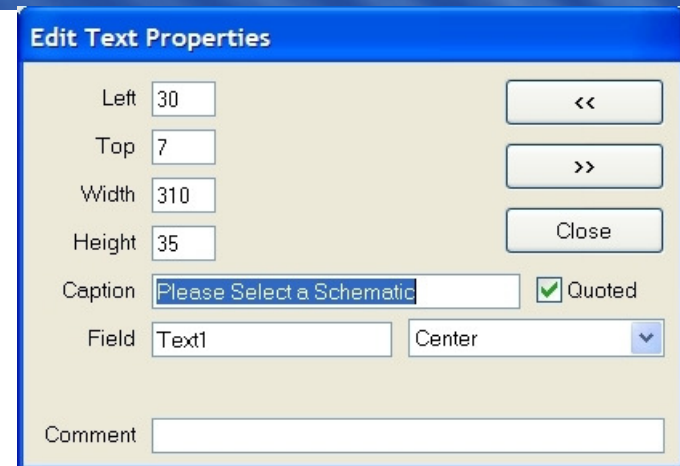
**Let's use a ListBox ... for the schematic names.**

**User will select a schematic ... and we will give them the number of resistors.**

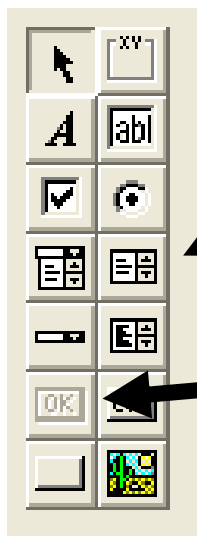
# UI Builder - 2



Add Text

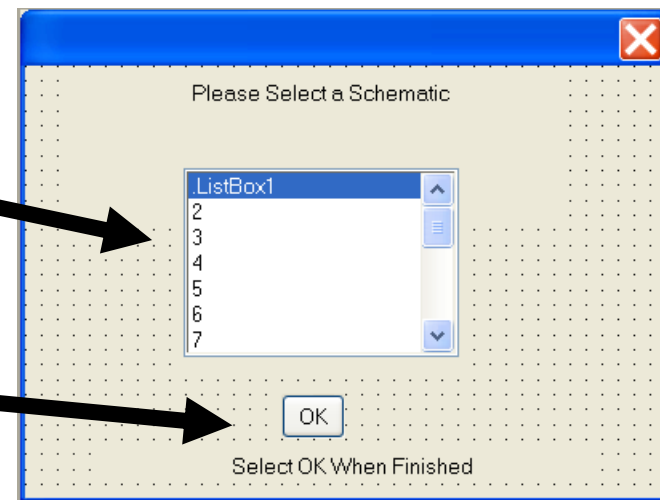


Double Click on Text to Input



List Box

OK Button



Finished Dialog Box

# UI Builder - 3

Upon Closing - Will Place Code where cursor was in Module.

```
Begin Dialog UserDialog 400,203 ' %GRID:10,7,1,1
    Text 30,7,310,70,"Please Select a Schematic",.Text1,2
    ListBox 100,49,170,91,ListArray(),.ListBox1
    OKButton 160,154,40,21
    Text 50,182,310,14,"Select OK When Finished",.Text2,2
End Dialog
Dim dlg As UserDialog
Dialog dlg
```

**Note: F1 gets you help on any keyword.**

- ListArray() is a placeholder for a 1D String array
  - Replace ListArray() with schname()
- .ListBox1 is a property of dlg which will give the element number of the schname() selected.

## Using the Code

```
Dim dlg As UserDialog  
Dialog dlg  
Debug.Print dlg.ListBox1
```

Define an Object - dlg  
of Type UserDialog

Run Dialog

Note: The ListBox1 function is defined for dlg. For example we can print it.

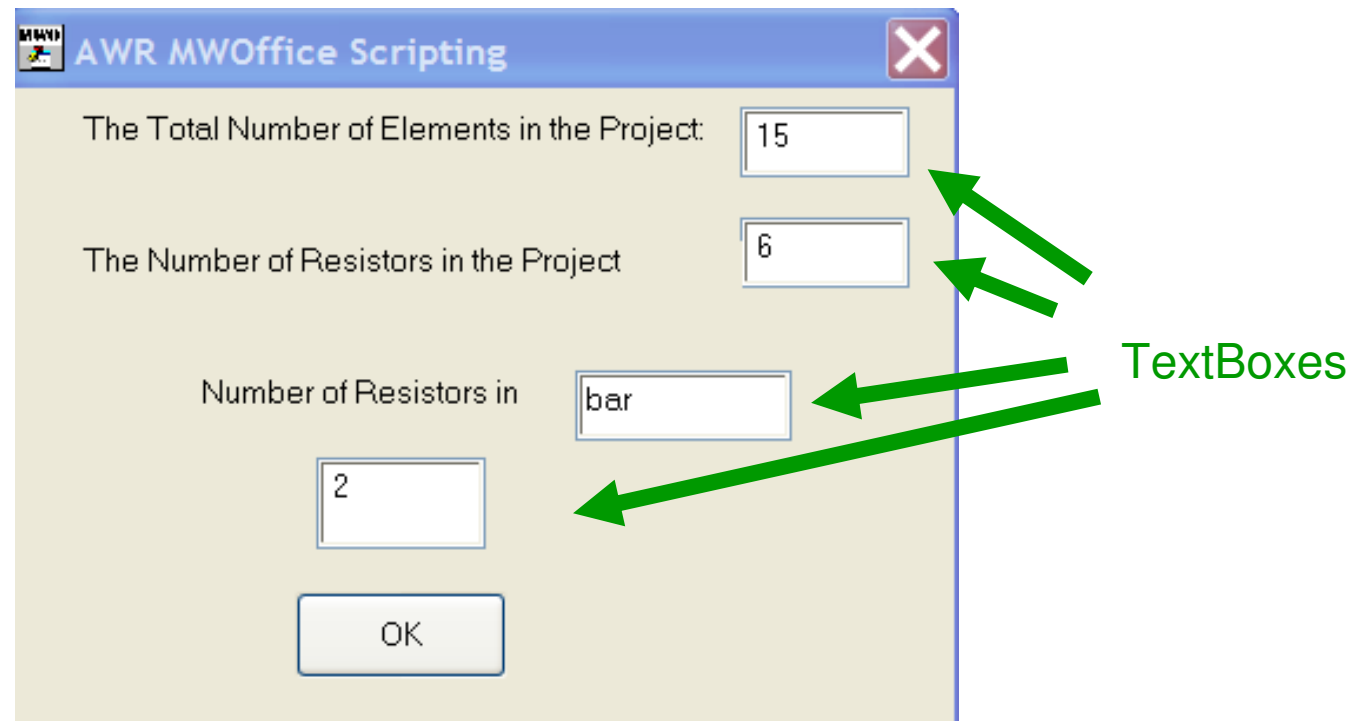
Remember - arrays start with element 0!

- dlg.ListBox1 = 0 ... The first element in the ListBox
- dlg.ListBox1 = 1 ... The second element in ListBox
- ...

# UI Builder - 5

## Create a second User Box...

- Lists the total number of elements and resistors in the project.
- Lists the number of resistor in the chosen schematic.




## One Way to Do It ...

```
Begin Dialog UserDialog 400,203 ' %GRID:10,7,1,1
    Text 30,7,330,28,"The Total Number of Elements in
the Project: ",.Text1
    TextBox 310,7,70,21,.TextBox1
    TextBox 310,42,70,21,.TextBox2
    Text 30,49,280,21,"The Number of Resistors in the
    Project",.Text2
    Text 80,91,210,28,"Number of Resistors in ",.Text3
    TextBox 240,91,90,21,.TextBox3
    TextBox 130,119,70,28,.TextBox4
    OKButton 120,161,90,28

End Dialog
```

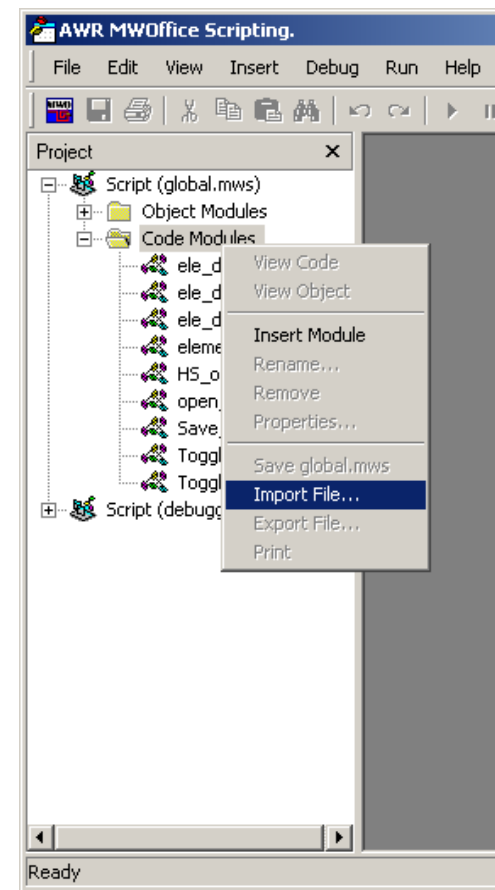
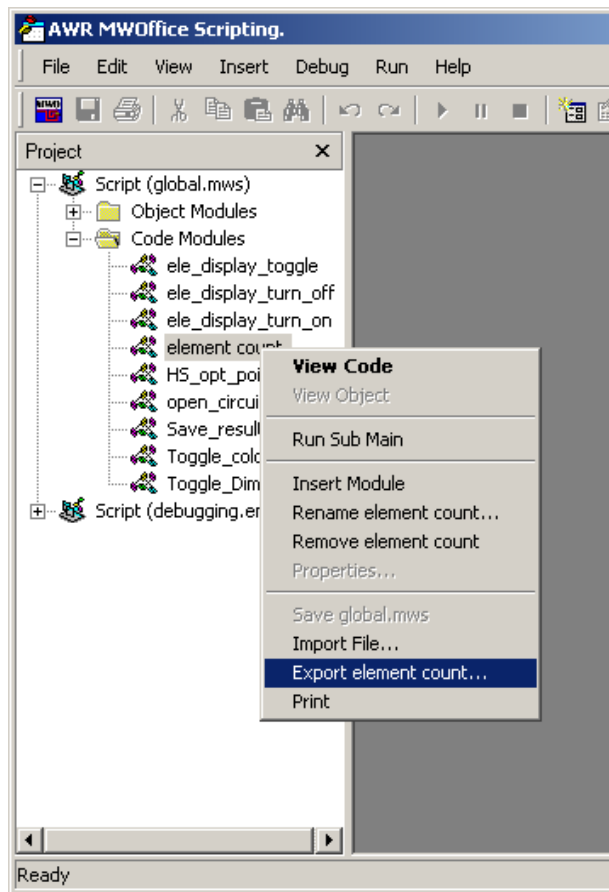
```
Dim answerbox As UserDialog
answerbox.TextBox1 = Str(totalelements)
answerbox.TextBox2 = Str(totalres)
answerbox.TextBox3 = schnames(dlg.ListBox1)
answerbox.TextBox4 = Str( resvalues(dlg.ListBox1))
```

 Note: Use of Str()  
function to convert  
to String.

# How to Share Scripts

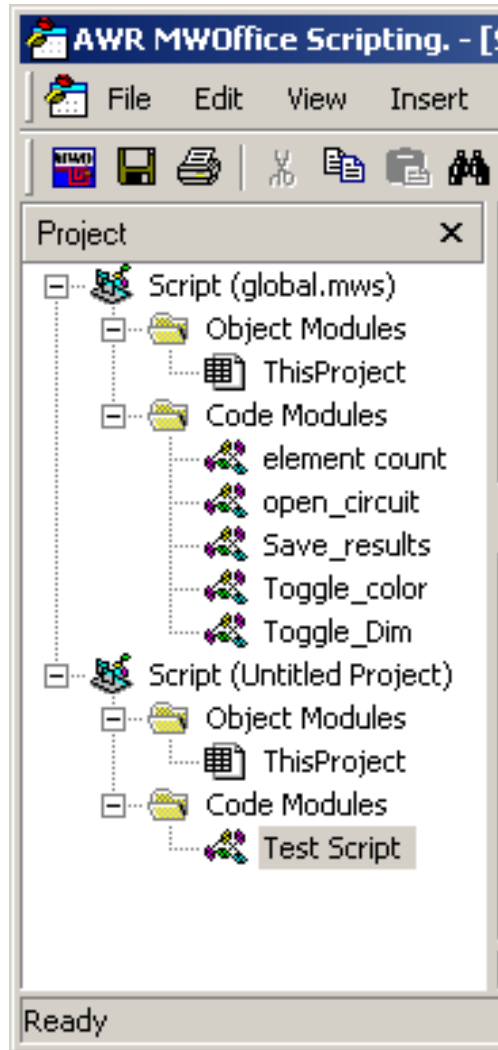
## Exporting Our Script

- Export a Visual Basic module into a .bas file that can then be imported into another project.



Scripting

# Global vs Local Scripts

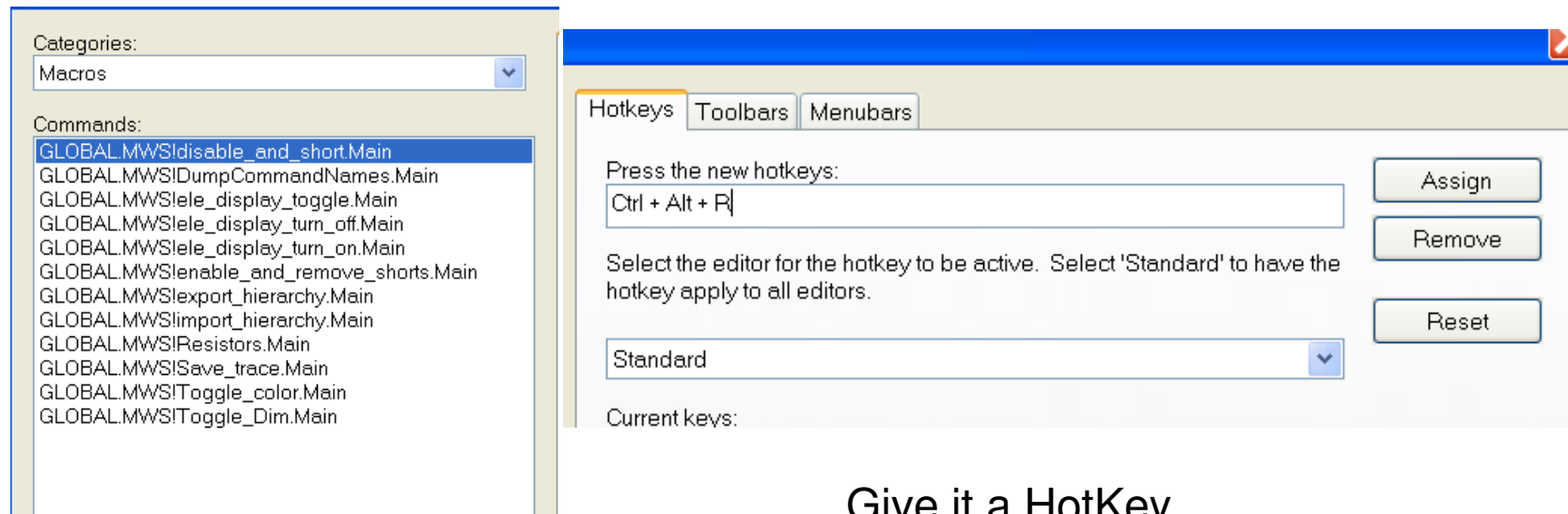


## Import Our Script in As a Global Script

- Global scripts will be available in every AWR project you open... they live on your computer.
  - They are saved in directories described in Slide 5.
- Project scripts will only be available in the project they are loaded.
- Project scripts are saved per project.

# Assigning HotKeys to Global Scripts

## Tools > Customize



Give it a HotKey

Pick Script in Category Macros

Make sure you hit Apply!

Tip: Can use Ctrl, Shift, and Alt keys  
in defining hotkey.

# Calling Other Com Objects

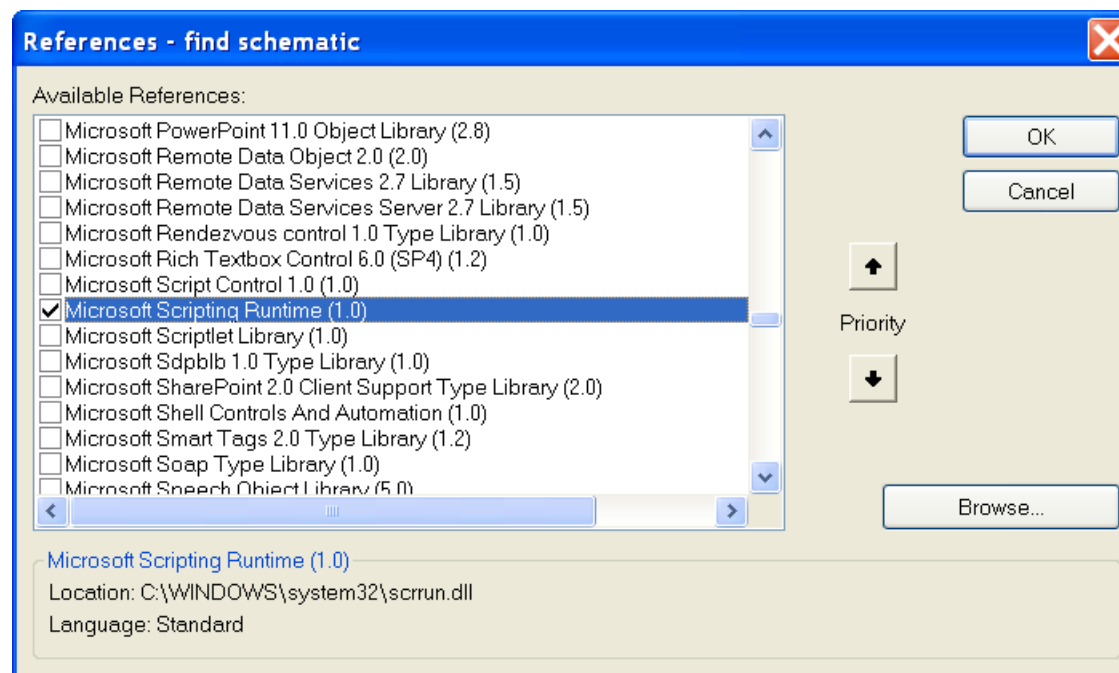
- Through the Microsoft Com API, you can easily call other COM compliant applications.
- You should add the object reference to the project.
- Then when you dimension a vector as the other object, you can call those objects, view the Intellisense for them etc.
- We will look at two cases
  - Microsoft Scripting Runtime
  - Excel

**Note: Don't confuse this with  
Microsoft Scripting Library**

# Dictionaries – A Handy Storage Container

- Dictionaries are a great way to store data
  - No need to dynamically resize anything
  - Array is keyword and value pairs

Requires Microsoft Script Runtime Reference be added as an object reference

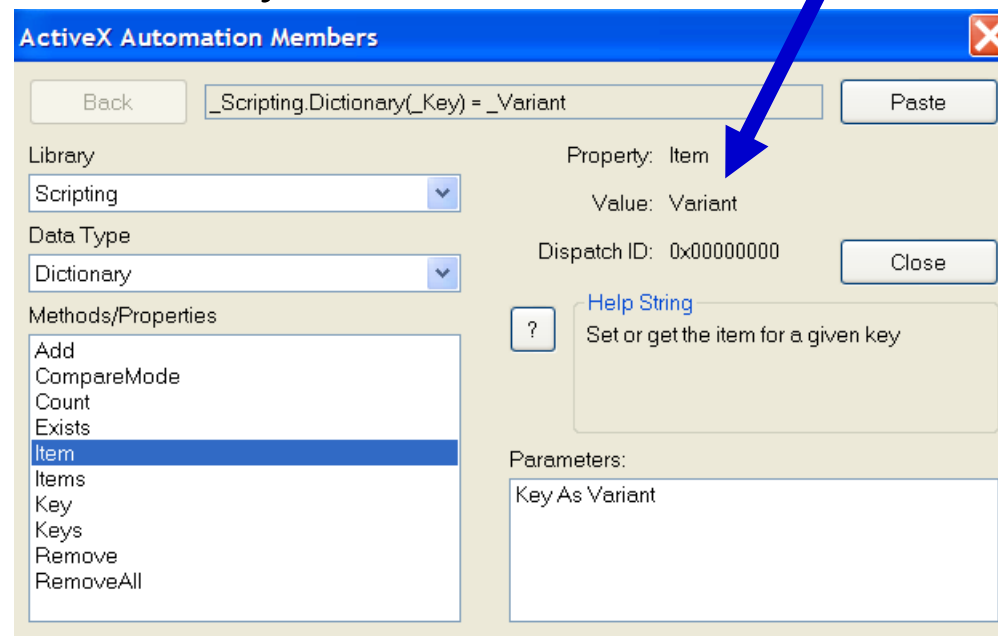


**Note: This library gives access to Files, Folders, Drives, etc.**

**- It's the critical library for working with anything in the file system!**

# The Variant Class

- All variables are of class Variant.
  - Integer, Long, ... are sub-classes.
- Variant can be anything! - string, integer, float,...
  - So - a Variant array can be a combination of different things.
  - Of course ... there's a danger you will get confused...
  - So use Option Explicit to override this.
- Dictionary items and keys work with the Variant datatypes - for maximum flexibility.



# Dictionaries - 2

This statement **defines** Test as a Dictionary object.

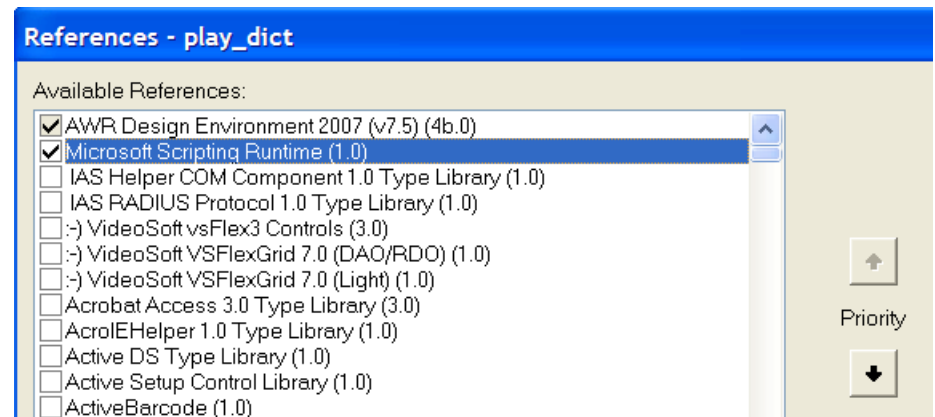
```
Dim Test As Scripting.Dictionary  
Set Test = New Dictionary
```

Set is used to **assign**  
Test to class Dictionary.

Assign and define are different - you need them both. when you create an object and want to work with it.

Note: Scripting is here to make sure we are looking in the MWOOffice Scripting library. If you don't put it in - it will look in AWR Design first ...

(But AWR doesn't have a Dictionary...)



Decreasing Priority

# Dictionaries - 3

- Provides ability to look up an 'item' based on a 'key'
- Add and remove entries based on either the item or the key

## Adding items

`_Object.Add(_Key,_Value)`

```
Test.Add("first","Resistor1")
```

```
Test.Add("second","My_project")
```

```
Test.Add("third",3.14)
```

```
Test.Add(4,7)
```

```
Test.Add("4","Pi")
```

Notice the Key and Value pairs  
can be any scalar variable type:  
Integer, String, Float,...

Note: These keys are different!  
One is an integer 4, the other is a  
string.

# Dictionaries - 4

## Other Examples

```
Debug.Print Test("third")
```

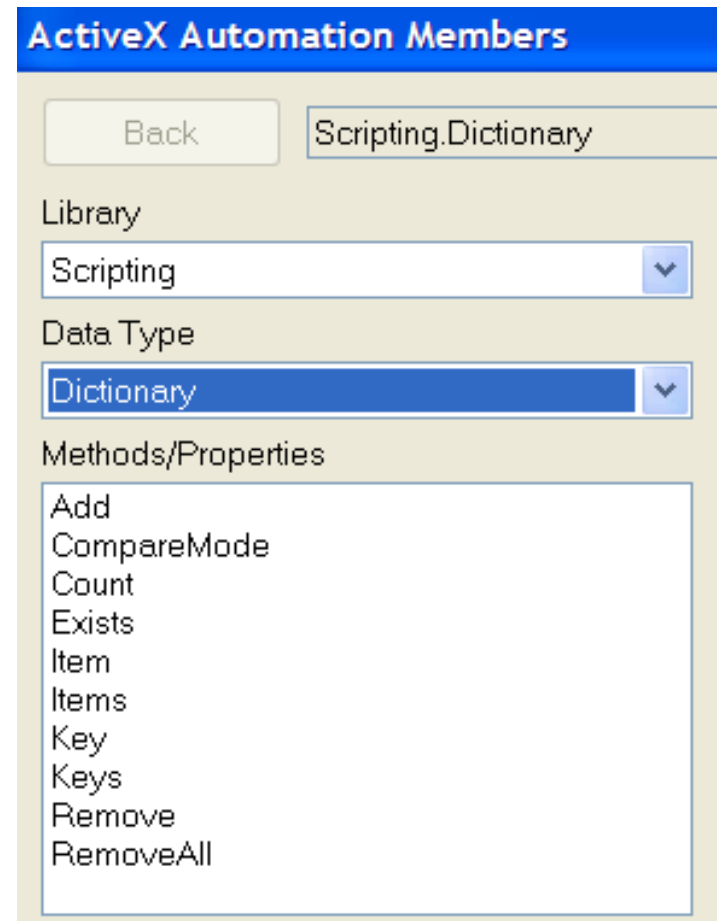
Prints item of Test with key "third".

```
Test("third") = 6.7
```

Sets array element with key "third" to 6.7.

```
myitems = Test.Items
```

myitems is a 1D Variant array with elements of the Test items.



Methods and Properties for  
Dictionaries

# Dictionary Example

Work with the PartA\_Res script.

We add a dictionary – where the keywords are the names of the schematics, and the values are the number of resistors.

```
Dim dictRes As Scripting.Dictionary ' A dictionary from Microsoft  
Scripting Namespace.
```

```
Set dictRes = New Dictionary
```

The **Dim** statement defines the dictRes as an object of type Dictionary ... but ....

The **Set ... New** statement is needed to actually create the dictionary!

# Dictionary Example - 2

```
schnames(schindex) = sch.Name  
resvalues(schindex) = numres
```

```
dictRes(sch.Name) = numres
```

```
.  
.  
.
```

```
Debug.Print dictRes("Moe")
```

The dictionary is placed in the loop with the schematic names as keywords ... and numres as values.

So later on we can find the number of resistors for a schematic.

The advantage of this ... a user could just type in a string - to be used as the keyword in the dictionary ...

# Microsoft Scripting Runtime ... and the file system

**You can interact with the file system.**

**Here's some snippets of a script written for recursively copying all the files from one directory to another.**

**Function** RecCopy(strFromLoc,strToLoc) ' **Recursively copies directories and files from strFromLoc to strToLoc.**

**Note: A function can be used in the main program. For example – we can use this one in the main program as:**

**RecCopy("C:\foo","C:\bar")**

**Functions have the form:**

**Function** funcname(variables)

...

**End Function**

# The File System - 2

**Function** RecCopy(strFromLoc, strToLoc) ' Recursively copies directories  
and files from strFromLoc to strToLoc.

```
Dim objFSys As Scripting.FileSystemObject
Dim objFolder As Object
Dim objSubFolder As Object
Dim objFile As Object
```

Defining the objects

```
Set objFSys = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSys.GetFolder(strFromLoc)
```

Creating the objects... 2 different ways.

We got a folder!



**Set ... CreateObject** – same as **Set ... New**

**GetFolder** – since we are doing something  
(method) ... the objFolder is created in the  
process.

# The File System - 3

`objFSystem.CreateFolder(strToLoc)`

**Create a folder.**

`objFSystem.CopyFile(strFilePath,strToLoc&strFileRel,True)`

**Copy Files.**

`strFolderName = objSubFOlder.Name`

**Find its name.**

**You can also delete ... so – be careful!**

It's usually a good idea to check if a folder or directory exists before you do something...

If `objFolder.SubFolders.Count > 0` Then ...

The entire function is named ... `RecCopy.bas`

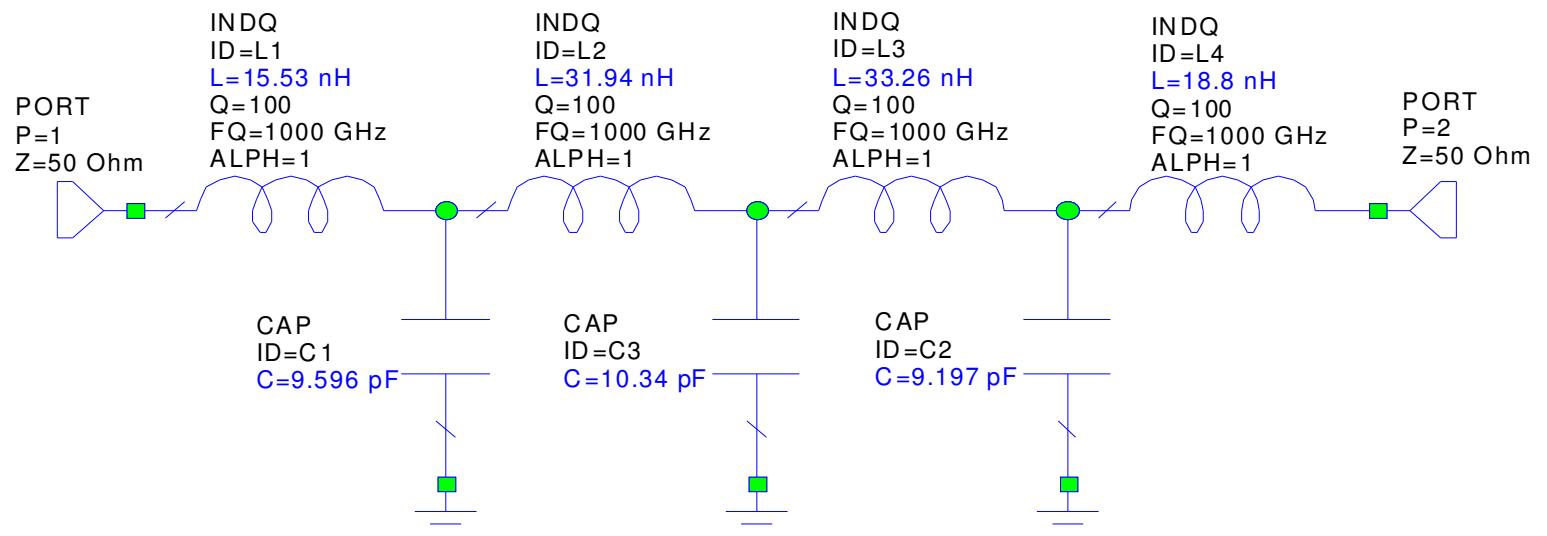
**Tip:** [http://msdn.microsoft.com/en-us/library/bstcxhf7\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/bstcxhf7(VS.85).aspx)

**for a lot of info on Script Runtime Library.**

Scripting

# Example - An Excel Spreadsheet

- Example of putting all the parameters for elements in a schematic in an Excel spreadsheet.
- Below is the schematic being used for this example - Excel.emp.

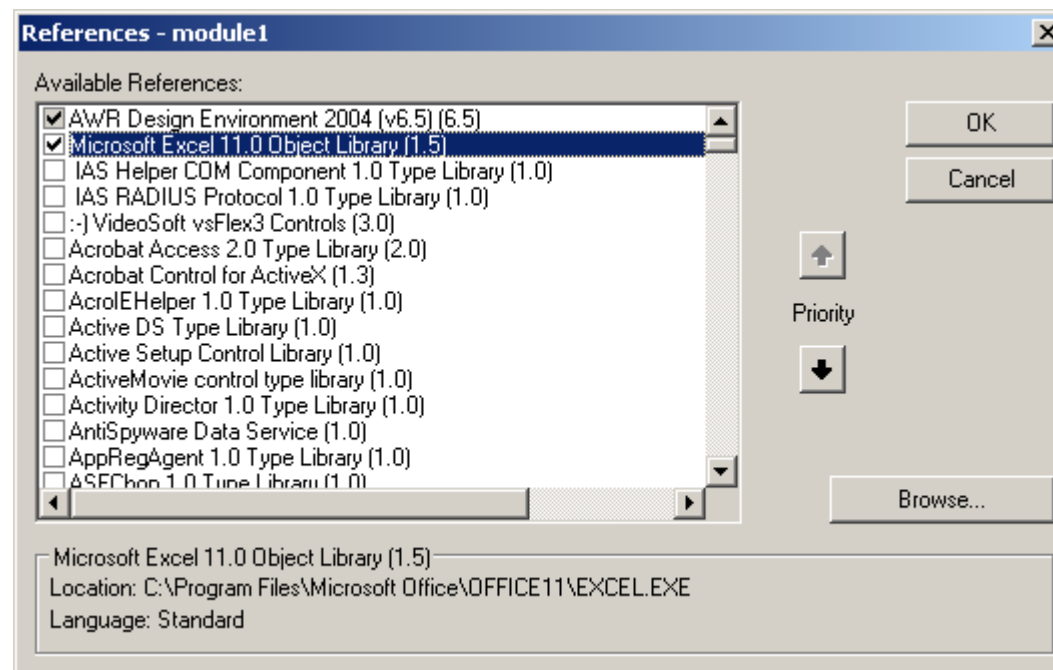


# Excel Reference Added

- Add Excel Object




- Add Microsoft Excel Reference



# Excel Code - 1



```
Dim sch As Schematic  
Dim p As MWOOffice.Parameter  
Dim elem As Element  
Dim Excel As Object  
Dim Workbook As Object  
Dim sheet As Object
```

Defining these as generic  
Objects right now... Can get  
specific later.



```
Dim Ex As Excel.Application
```

Need to specifically refer to  
Excel namespace as MWOOffice  
also has Application.



```
Sub Main
```

```
' Create an instance of Excel
```

```
Set Ex = CreateObject("Excel.Application")
```

Ex is now assigned to an Excel  
Application.

# Excel Code - 2

```
If Ex = "" Then
    MsgBox("Excel not found on this machine, program terminated")
    Exit Sub ← Forces you out of the If Then
End If
Ex.Visible = True
Ex.Interactive = True
shts = Ex.SheetsInNewWorkbook 'stores users original default sheets per
workbook
Ex.SheetsInNewWorkbook = 1 'sets new workbooks to only have one
sheet, however, changes user default
Set Workbook = Ex.Workbooks.Add() 'adds new workbook
    Ex.SheetsInNewWorkbook = shts 'reset sheets per workbook
default
    shtcnt = 1 'stores number of sheets
```

This code adds a new workbook with one sheet - after making sure not to trash the user's default number of sheets for new workbooks.

## Excel Code - 3

'set schematic (could be done with UI).

Set sch = Project.Schematics("filter")

'Add column headers for each new sheet

Set sheet = Workbook.Sheets(1)

sheet.Name = sch.Name

sheet.Range("A1").FormulaR1C1 = "Element"

sheet.Range("B1").FormulaR1C1 = "Parameters"

row = 2 'starting row

This code names the sheet to the schematic name, and puts the fields Element and Parameters at the top of the first 2 columns.

# Excel Code - 4

'loop through all elements

For Each elem In sch.Elements

    If elem.Enabled = True Then

        sheet.Range("A"+row).FormulaR1C1 =

        elem.Name

        chrval = 66 'numerical value for character A

        For Each p In elem.Parameters

            End If

        Next elem

        sheet.Range(Chr(chrval)&row).FormulaR1C1 = p.Name + "=" +  
        p.ValueAsString

        chrval = chrval+1

        Next p

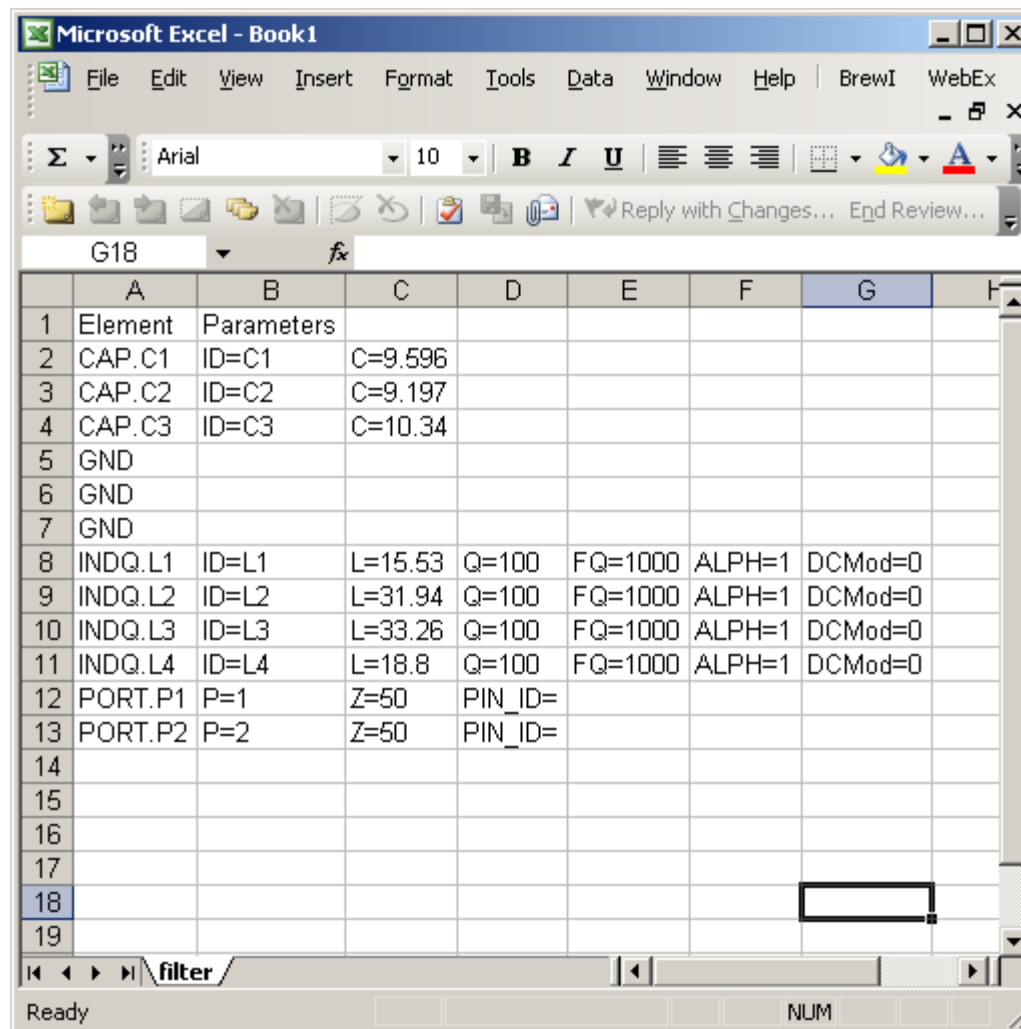
        row = row+1

    ' Fit the columns to the data we've entered.

    sheet.Range("A1:J1").EntireColumn.AutoFit

End Sub

# Calling Other Applications – Results

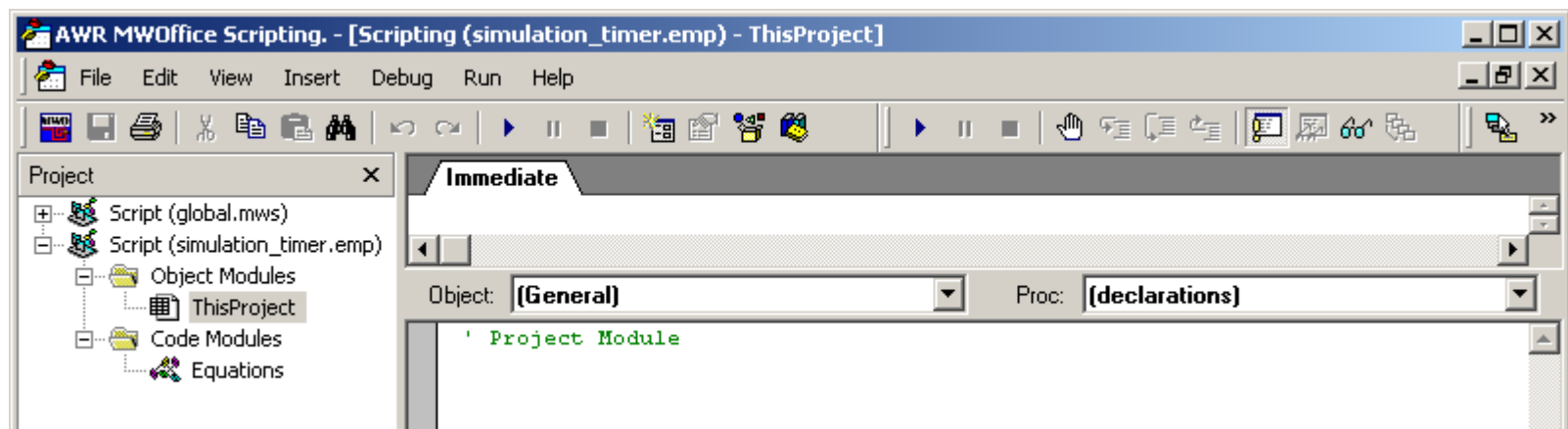


The screenshot shows a Microsoft Excel window titled "Microsoft Excel - Book1". The spreadsheet contains a table with 19 rows and 8 columns (A-H). The first row (row 1) has headers "Element" in column A and "Parameters" in column B. The subsequent rows contain circuit components and their associated parameters. The table is as follows:

	A	B	C	D	E	F	G	H
1	Element	Parameters						
2	CAP.C1	ID=C1	C=9.596					
3	CAP.C2	ID=C2	C=9.197					
4	CAP.C3	ID=C3	C=10.34					
5	GND							
6	GND							
7	GND							
8	INDQ.L1	ID=L1	L=15.53	Q=100	FQ=1000	ALPH=1	DCMod=0	
9	INDQ.L2	ID=L2	L=31.94	Q=100	FQ=1000	ALPH=1	DCMod=0	
10	INDQ.L3	ID=L3	L=33.26	Q=100	FQ=1000	ALPH=1	DCMod=0	
11	INDQ.L4	ID=L4	L=18.8	Q=100	FQ=1000	ALPH=1	DCMod=0	
12	PORT.P1	P=1	Z=50	PIN_ID=				
13	PORT.P2	P=2	Z=50	PIN_ID=				
14								
15								
16								
17								
18								
19								

# Event Handlers

- Scripting Code that runs when certain events happen with the software (simulation starts, simulation stops, place an element, etc).
- Can stop the simulation in each optimization iteration.
- Event handles are coded by opening up the **ThisProject** node under the **Object Modules** node.
- Select **Project** in the **Object** selection to have a list of all AWR event handles in the **Proc:** section.
- When you select an event handler a section of code is added.



# Scope and Lifetime of Variables

- **Scope** is what part's of the program can “see” a variable.
  - **Procedure Level Scope (or Local Scope)**: Within a procedure or variable - everything can see the variable.
  - **Note**: So far, all are variables have been Local to Procedure Main.
  - **Script Level Scope (or Global Scope)**: When the whole script sees a variable.
  - **Note**: For Classes - also something called Private and Public... We won't discuss it here.
- **LifeTime** of a variable is how long it “exists” in memory.
  - Normally ... variables exist while the procedure / function is running.

# Scope Example

```
' Code Module
```

```
Option Explicit  
Dim foo As Long
```

---

foo is a global variable

```
Sub Main  
    Debug.Clear  
    foo = 0  
    Debug.Print foo  
    foo=4  
    Testexample  
    Debug.Print foo  
    Debug.Print longTry  
End Sub
```

---

```
Sub Testexample  
    Dim longTry As Long  
    longTry = 3  
    Debug.Print longTry + foo  
    foo=10  
End Sub
```

---

longTry is local to the  
Procedure Testexample.

# Simulation Timer Example

Make Sure on Project not (General)



```
Object: Project

' Project Module
Option Explicit

Dim tstart As Double
Dim tstop As Double

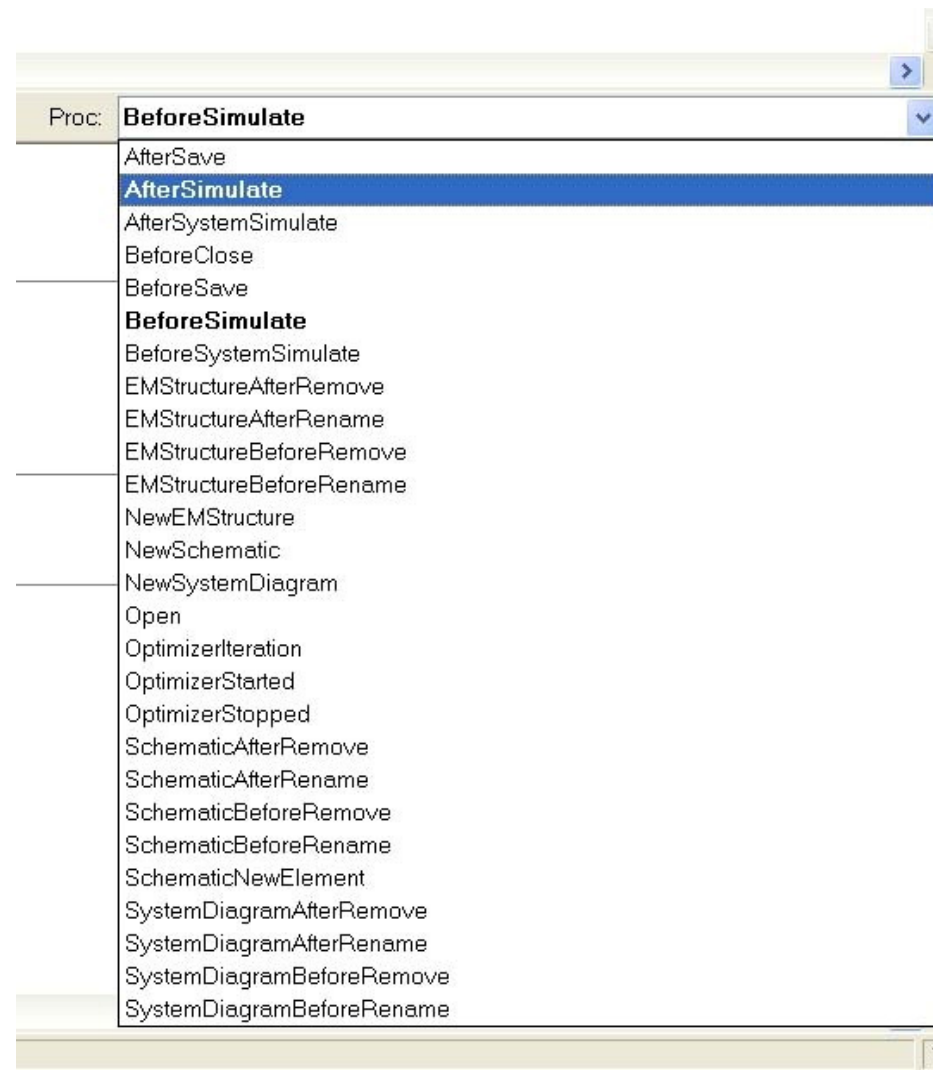
Private Sub Project_AfterSimulate()
    Dim tlength As Double
    tstop = Timer
    tlength = tstop - tstart
    MsgBox("Simulation Time" + CStr(tlength) + " Seconds")
End Sub

Private Sub Project_BeforeSimulate(Cancel As Boolean)
    tstart = Timer
End Sub
```

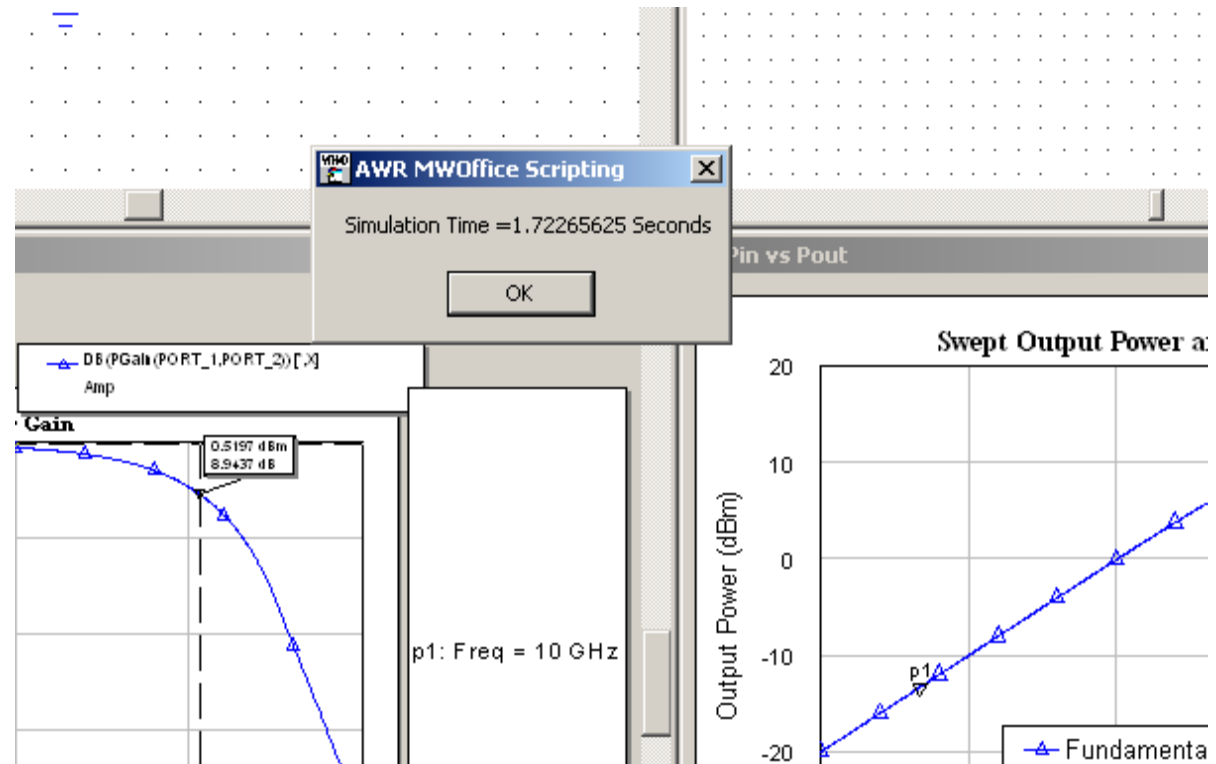
Globally Defined Variables

# Simulation Timer - 2

These are the available  
Procedures.

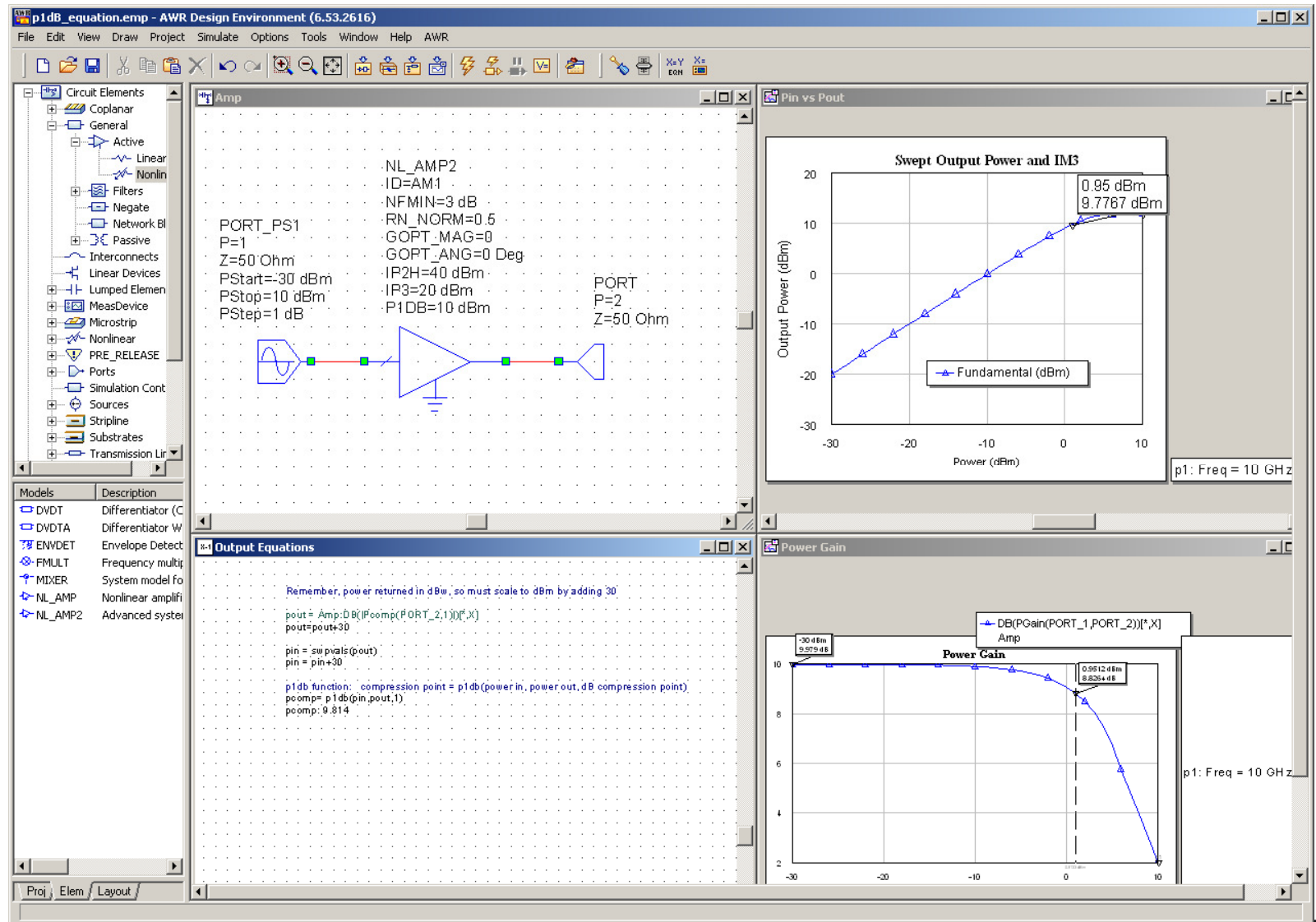


# Simulation Timer Example - Results



- Methodology to extend Output Equations to do anything you want.
- Allows users to use a power programming environment that will proven and bug-free
- Done by writing “Function” blocks that live in a VB Module called “Equations”.
- P1dB Example
  - P1dB is not a built in measurement, so can be done with a custom function.

# P1dB Example - Project



# P1dB Example - Equations

- First 4 equations are getting input and output power in dBm.
- Fifth equation has a custom function called p1db, that takes vector of input power, vector of output power, and the compression point (1dB, 2dB, etc) and returns the compression point as a single number

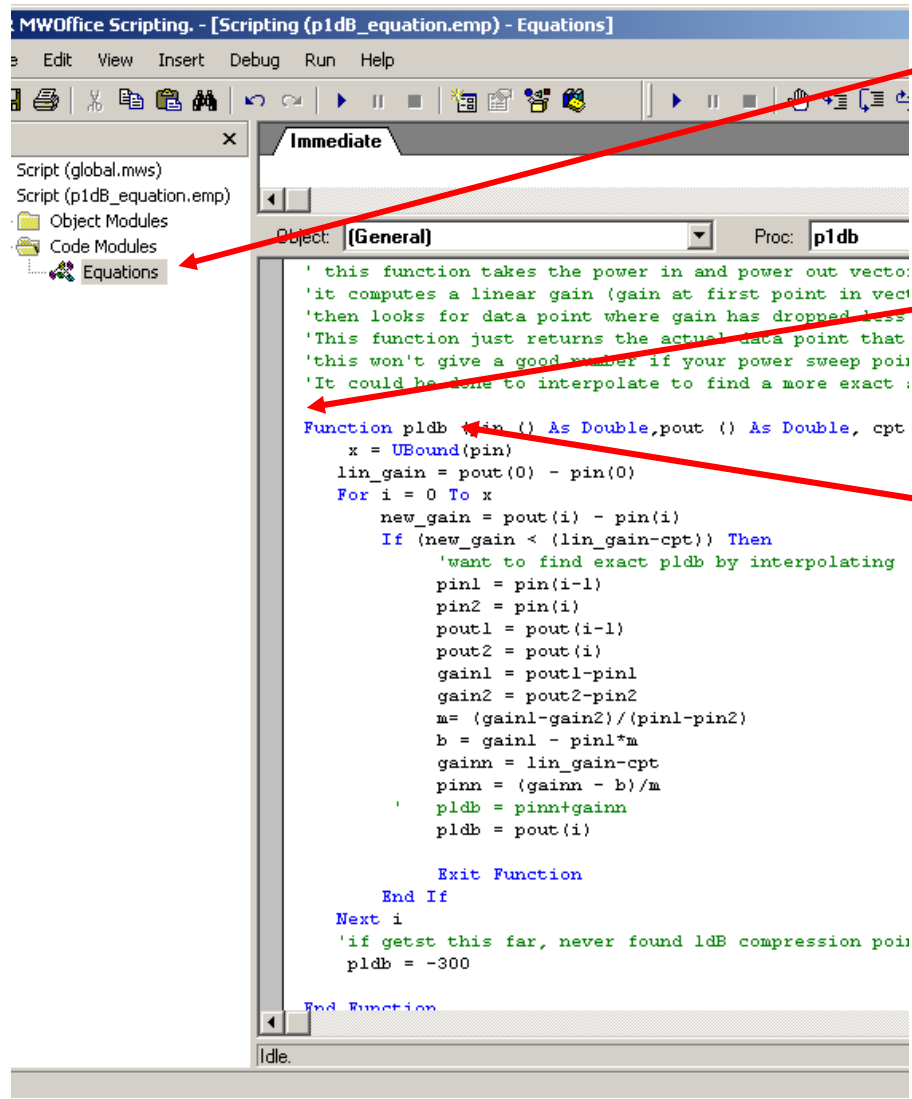
Remember, power returned in dBw, so must scale to dBm by adding 30

```
pout = Amp:DB(|Pcomp(PORT_2,1)|)[*,X]  
pout=pout+30
```

```
pin = swpvals(pout)  
pin = pin+30
```

```
p1db function: compression point = p1db(power in, power out, dB compression point)  
pcomp= p1db(pin,pout,1)  
pcomp: 9.814
```

# P1dB Example - Equations



```
Script (global.mws)
Script (p1dB_equation.emp)
Object Modules
Code Modules
Equations

' this function takes the power in and power out vector
' it computes a linear gain (gain at first point in vector)
' then looks for data point where gain has dropped less than 1dB
' This function just returns the actual data point that
' this won't give a good number if your power sweep point is not
' It could be done to interpolate to find a more exact point

Function p1dB (pin () As Double, pout () As Double, cpt As Double) As Double
    x = UBound(pin)
    lin_gain = pout(0) - pin(0)
    For i = 0 To x
        new_gain = pout(i) - pin(i)
        If (new_gain < (lin_gain-cpt)) Then
            'want to find exact p1dB by interpolating
            pin1 = pin(i-1)
            pin2 = pin(i)
            pout1 = pout(i-1)
            pout2 = pout(i)
            gain1 = pout1-pin1
            gain2 = pout2-pin2
            m = (gain1-gain2)/(pin1-pin2)
            b = gain1 - pin1*m
            gainn = lin_gain-cpt
            pinn = (gainn - b)/m
            p1dB = pinn+gainn
        End If
    Next i
    'if getst this far, never found 1dB compression point
    p1dB = -300
End Function
```

Name of code module must be **Equations**

Code written as a **Function**

Name of function is name used in output equations to call the function

Debugging custom functions is the same as stand alone scripts. The only difference is that the break points will catch when you simulate (causes script to execute)