Scripting Getting Started

Introduction

The AWR Design Environment comes with a complete COM Application Program Interface (API) that can be accessed programmatically through any language or system that can use COM APIs. The AWRDE also contains a Scripting Development Environment to allow you to develop your automation tasks. The application engineers at AWR frequently use the SDE to build custom solutions to customer problems. Many such solutions have wide-spread applications. The AWRDE software includes these scripts, and they are available from the **Scripts** menu in software. This guide will walk you through the basics of the API and the SDE. The guide includes:

1. Links to essential resources to help you work with the AWR API.
2. Create your first simple script.
3. Recreate the linear chapter of the AWR Getting Started Guide which is a simple lumped element filter simulation including optimization.
4. List of other useful topics when developing scripts.
5. The same finished code in Python.

Important Resources

- Sax Basic function help document. AWR_Sax_Basic.pdf
- Script Development Environment (SDE) Documentation. Scripting Development Environment
- AWR API UML Diagram. AWR_V13_API_Diagram.pdf
- Shipping AWR Scripts as Examples. Script Help Landing Page for AWR V15
- Scripting Topics. Scripting Landing Page
- The **Some Topics to Think About** section below contains useful information on building dialog boxes and other coding approaches that will be useful.

My First Script

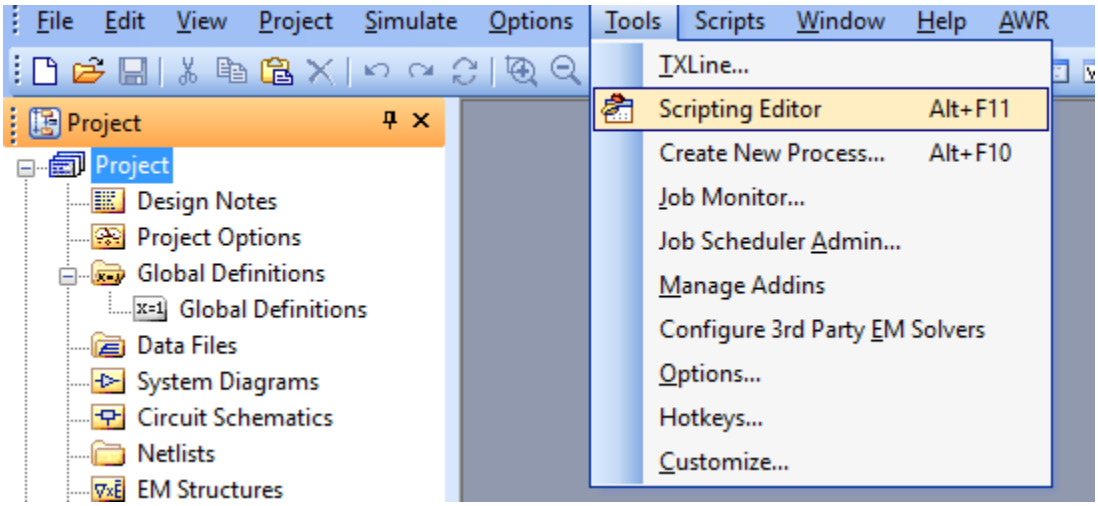
This first script will be a simple script to show you the Scripting Development Environment (SDE) built into the AWR Design Environment.

Start the AWR Design Environment

Launch the AWR SDE from within the AWR Design Environment. Your first step is to start the AWR Design Environment. Save the project to a file named **MyFirstScript** to help follow along with this guide.

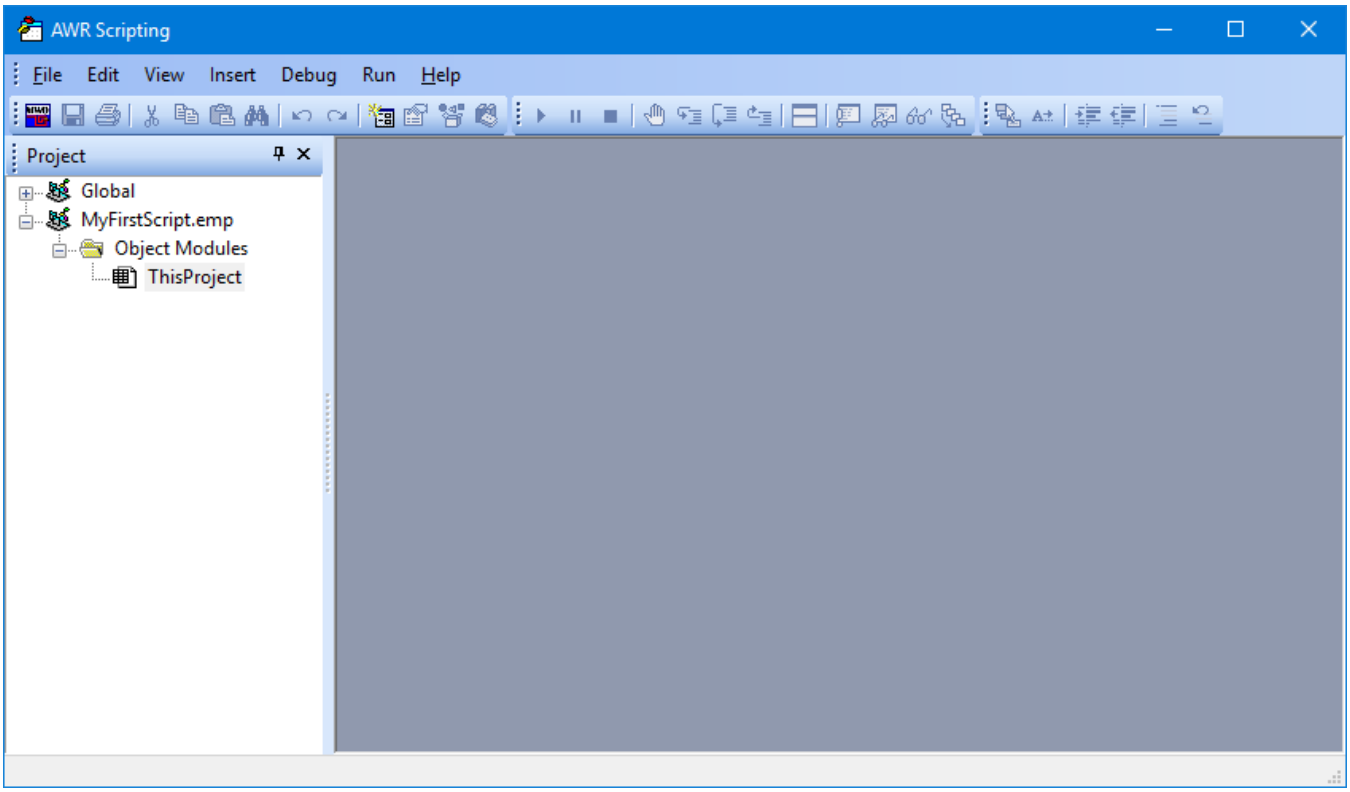
Opening the SDE

From the menus, select **Tools > Scripting Editor.**



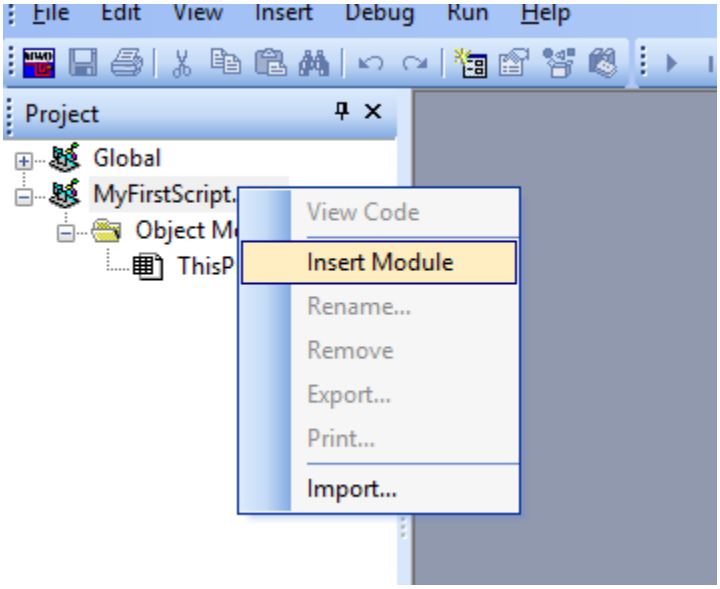
Alternately, you can use the hotkeys Alt+F11 or click on the **Scripting Editor** toolbar.

A new window titled **AWR Scripting** will open which is the AWR SDE.

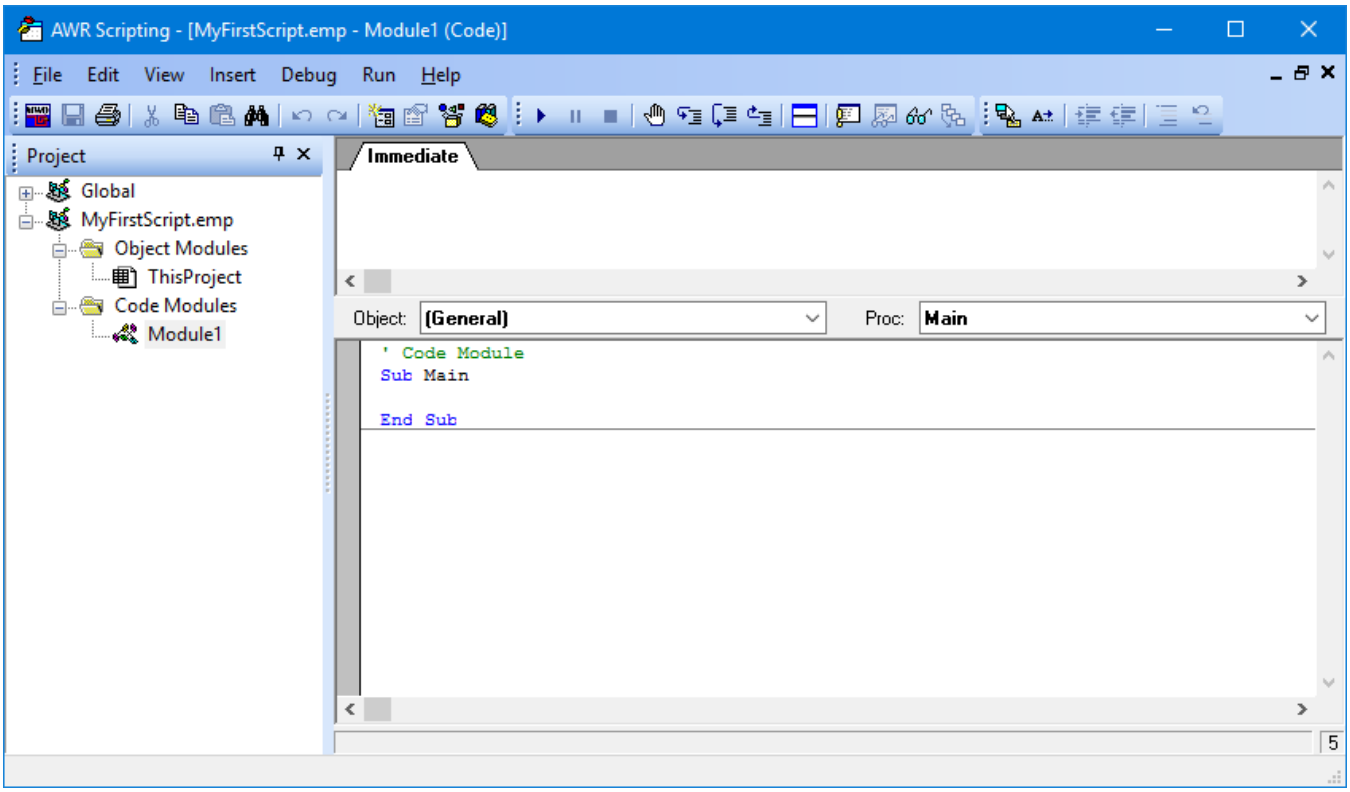


Creating a New Code Module

From the SDE Project browser, right click over the **MyFirstScript.emp** node and select **Insert Module**

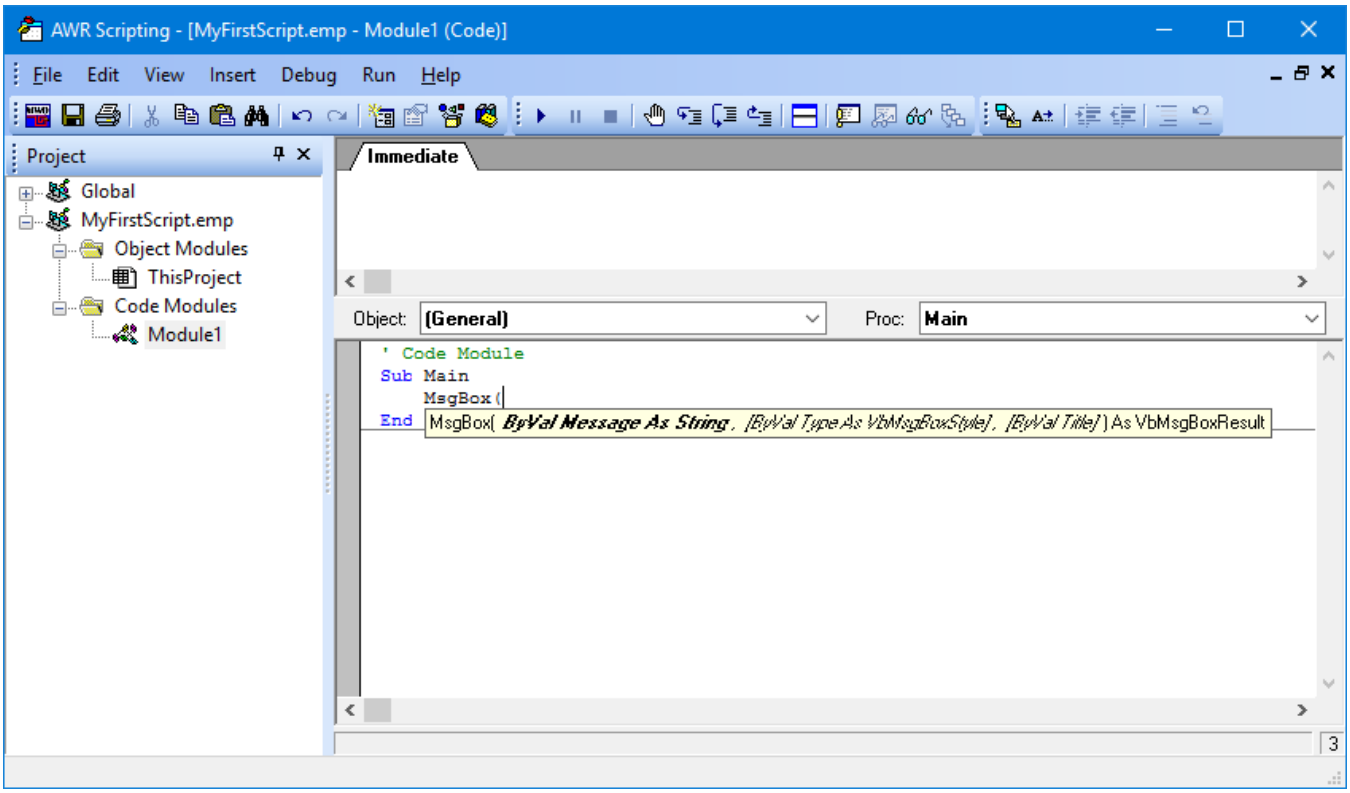


This command adds a new module named **Module1** under the **Code Modules** node in the project The code module window will also open for editing. You might need to maximize this window to see the full contents. Notice this command also adds the **Sub Main** and **End Sub** code.



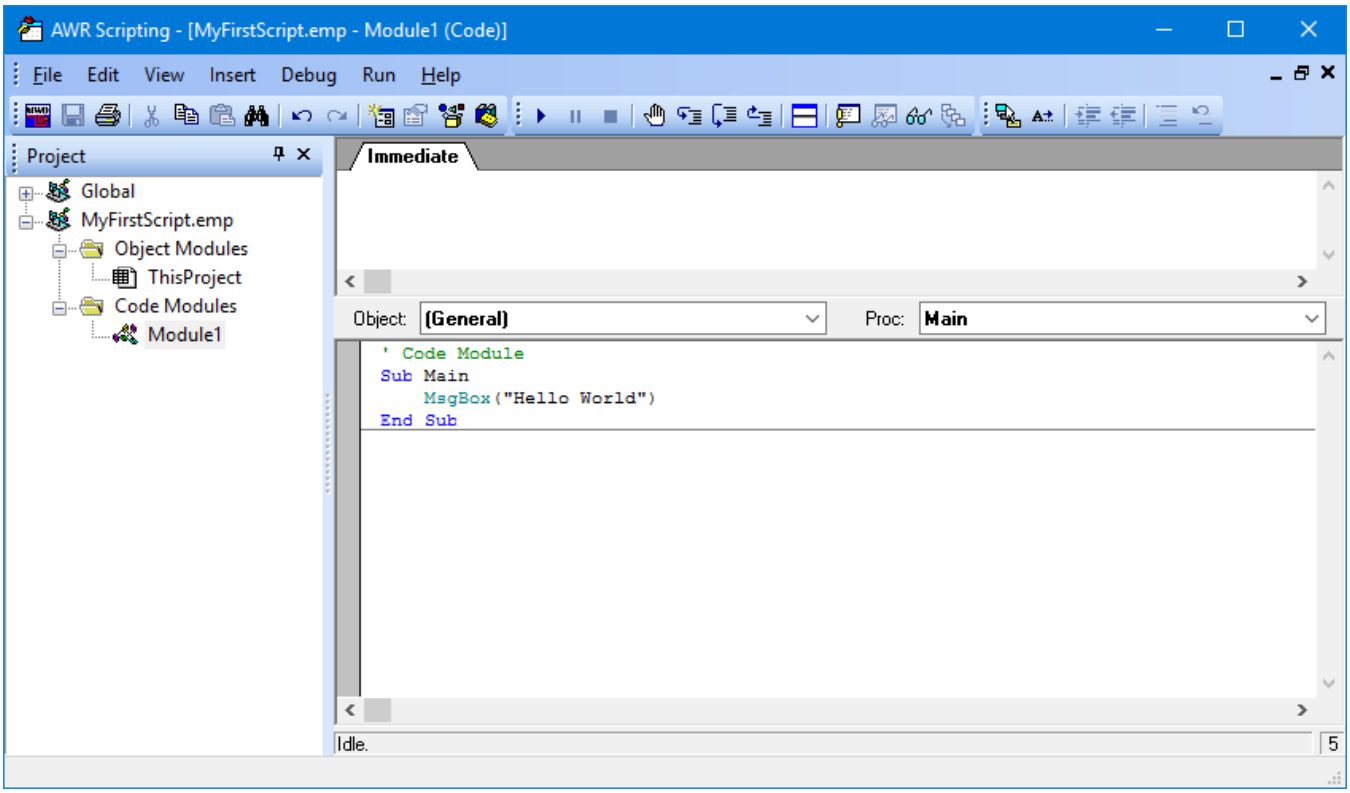
Adding the Code

In the open code module, type "msgbox(" as shown below



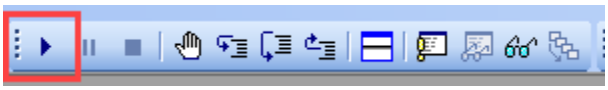
Notice as soon as you type the open parenthesis, information displays about the inputs to the function. The first input is the message to display in the message box. The second two are optional to set the type of message box and the title of the message box.

Now finish the rest of the line so that it reads "msgbox("Hello World")"

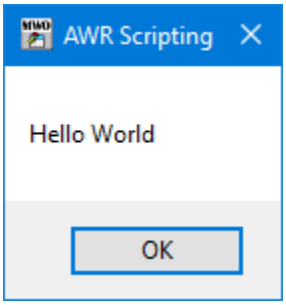


Running the Code

Your code is ready to run. To run the code, use the **Run** toolbar as shown below.



When you run this code, you will see a message box display as shown below.



Congratulations, you have created and run your first script!

Creating a Filter Simulation

This section will cover coding a script to create the same circuit as in the linear chapter of the Microwave Office getting started guide. In general, this will work through the various steps of generating the filter. Each step will be a function or subroutine in the Sub Main. The guide will show you how to change the Sub Main and then just the code to implement that function or subroutine. This organization will make showing the code changes at each step simpler.

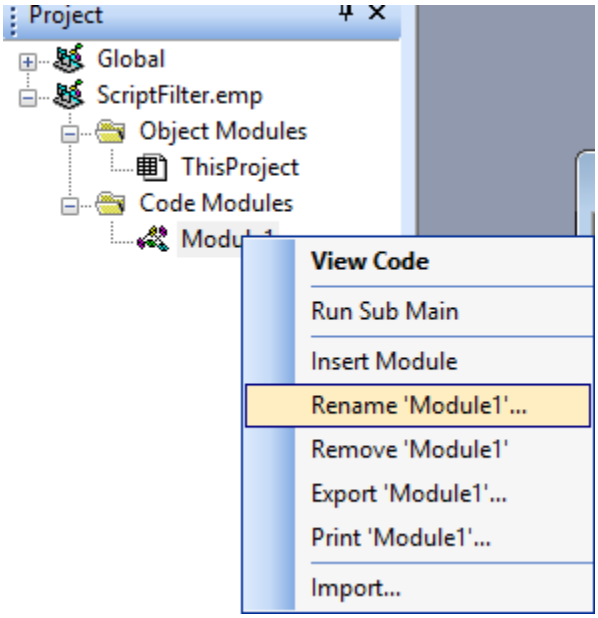
Note: The code you type is case sensitive. When using the built-in functions, the case is fixed for you. Where this can be problematic is with subroutine or function names.

Create a New Project

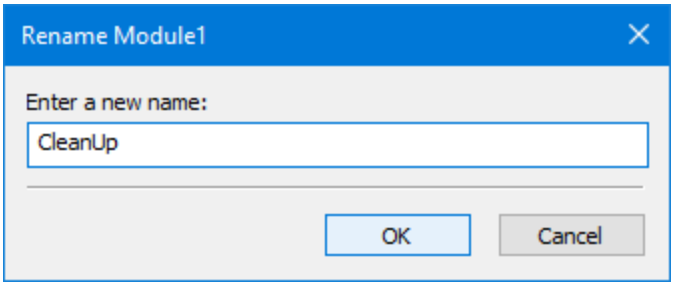
Either start a new project or select **File > New Project**. Then save the project with the name **ScriptFilter.emp.**

Clean Up Code

When developing scripts, it is common that the script will change your project. It is a good idea to have code to clean up the project while you are developing code to save the work of doing manual clean up. For this example, create a new code module. Right click on the code module, select **Rename 'Module1'...**



Then type **CleanUp** in the dialog that opens.



Copy the code below into the module.

```
Option Explicit
Sub Main
        Dim s As Schematic
        Dim g As Graph
        Dim opt As OptGoals

        For Each s In Project.Schematics
                Project.Schematics.Remove(s.Name)
        Next s

        For Each g In Project.Graphs
                Project.Graphs.Remove(g.Name)
        Next g

        Project.OptGoals.RemoveAll

        MWOffice.Windows.Close
End Sub
```

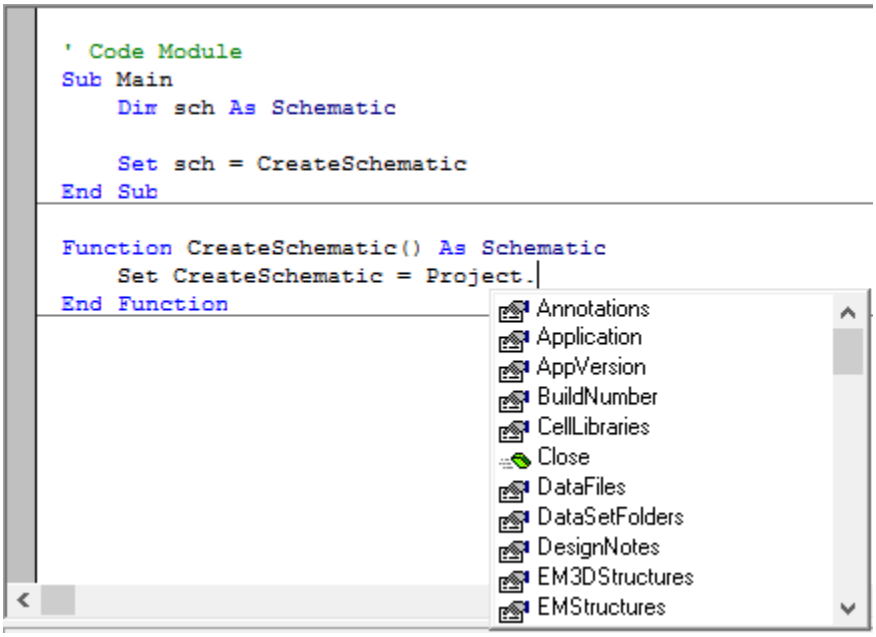
Create Schematic

Create a new code module and rename it to "Filter."

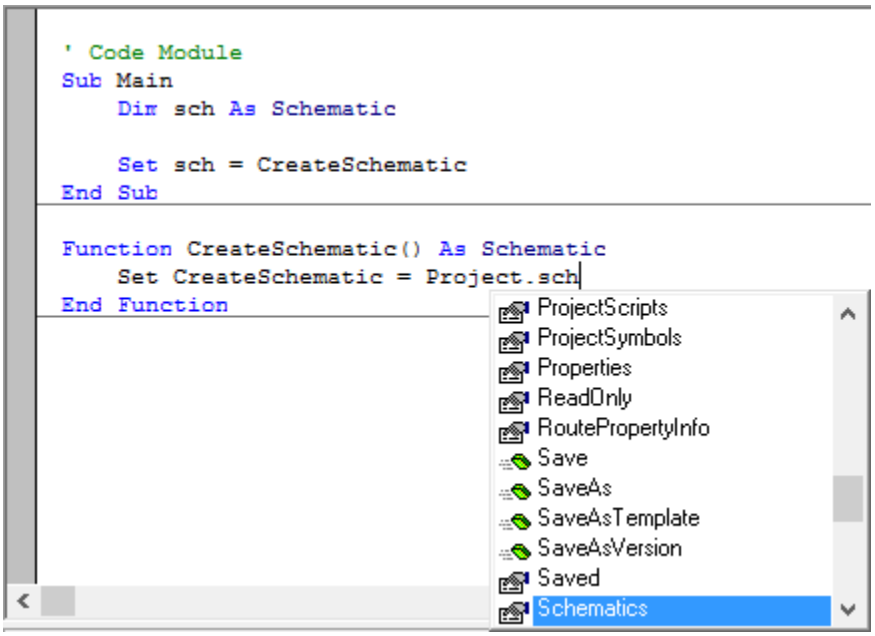
The code is below. We recommend you type the code to get used to typing and viewing the autocomplete.

```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        Set sch = CreateSchematic
End Sub
Function CreateSchematic() As Schematic
        Set CreateSchematic = Project.Schematics.Add("lpf")
End Function
```

For the autocomplete, when you type "project.", you will get a list of the objects available under the Project object.



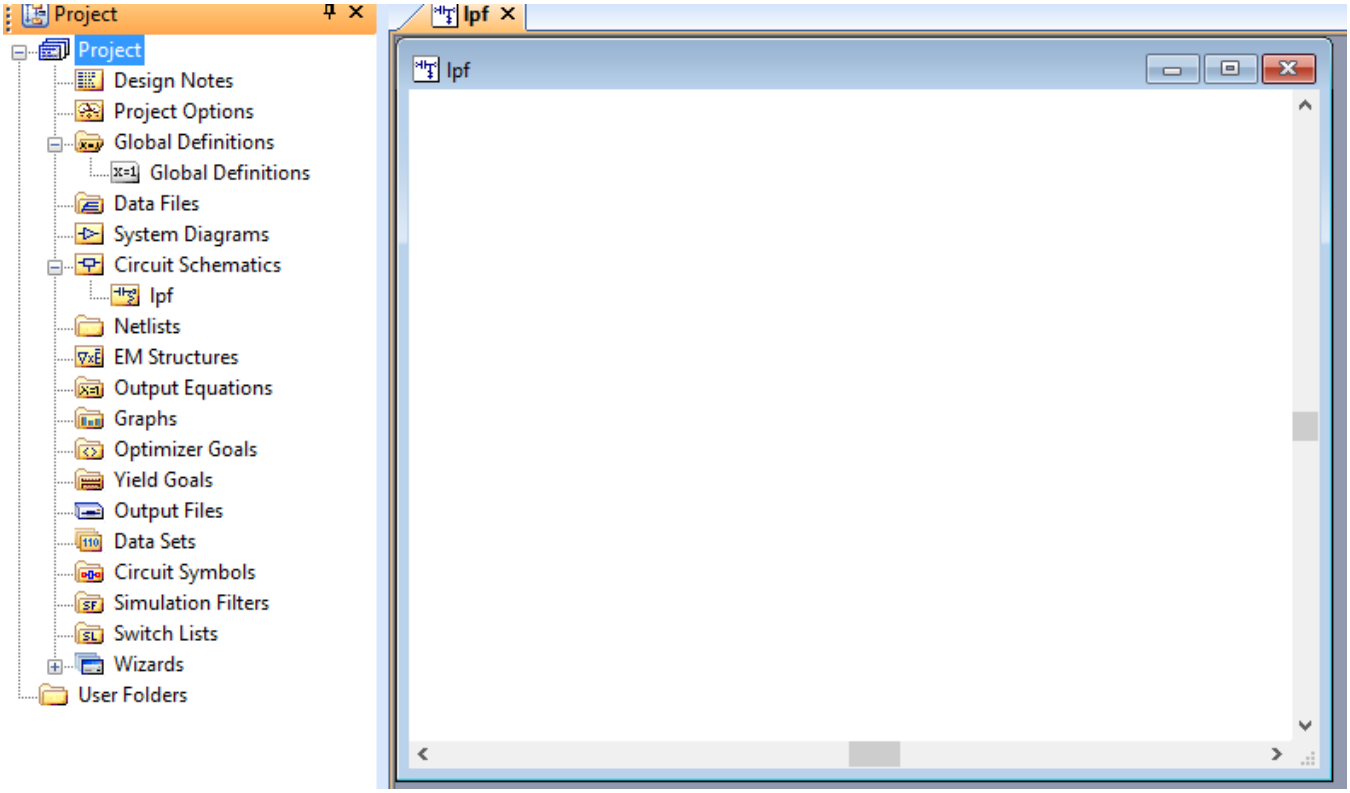
As you continue typing, it will match and highlight the closest child object. For example, typing "project.sch" will show as below.



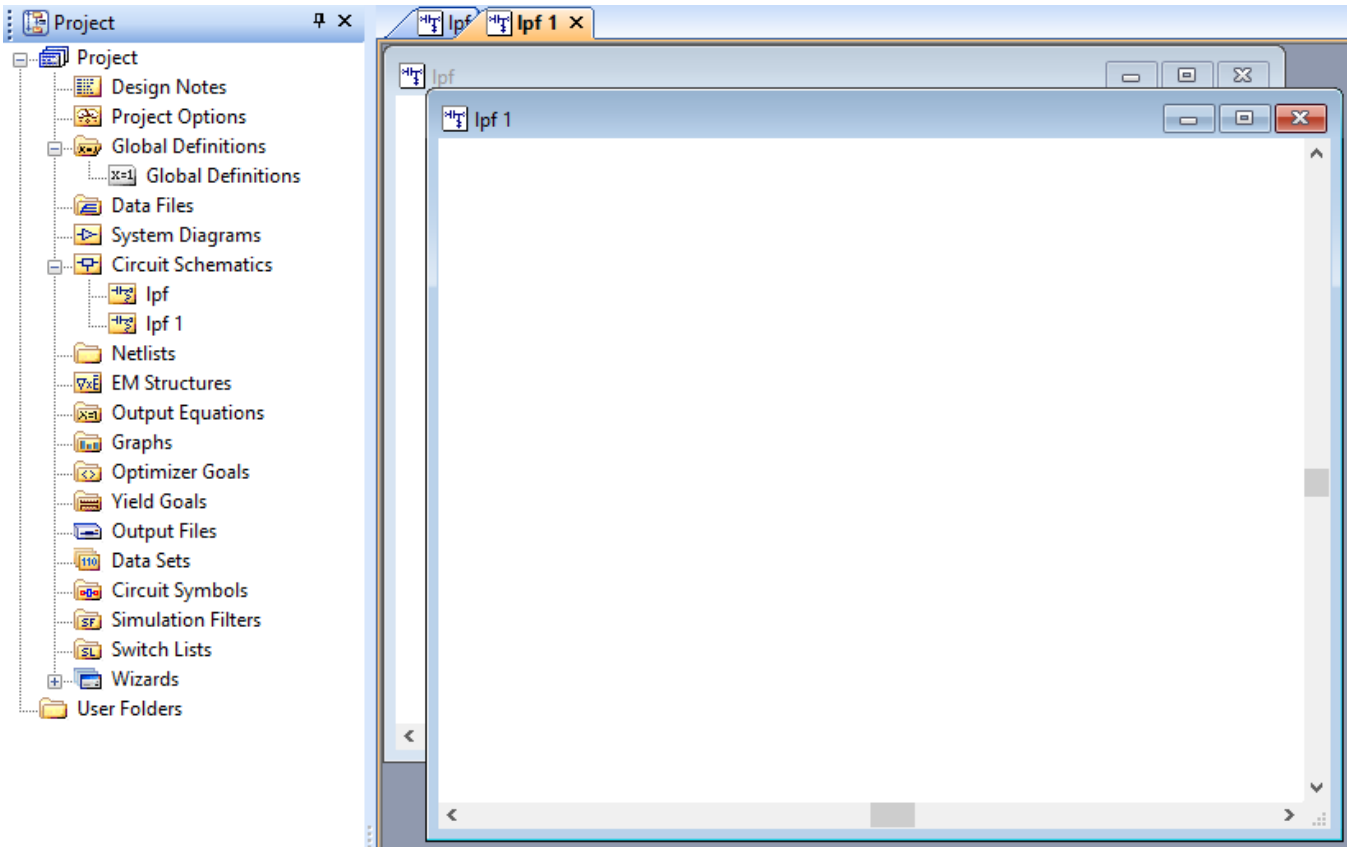
Notice the highlighted object. Pressing the **Tab** key will complete that word.

Now, if you run this code, a new schematic is added.

The code adds a new schematic and opens a new blank schematic window.



Note: If you run the script again, it will create a new schematic with a slightly different name.



The CreateSchematic code was done as a function to be able to send up an object for the schematic created. Keeping track of the schematic reference is vital in case you create objects by name that already exist; you can reference the schematic as created this way.

At this point, you want to run the **CleanUp** script to remove these schematics before going to the next step.

To make things easier, run the CleanUp code at the start of the script. The main subroutine should now look like below:

```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic
End Sub
```

Then add the cleanup code after the sub main subroutine.

```
Sub CleanUp
        Dim s As Schematic
        Dim g As Graph
        Dim opt As OptGoals

        For Each s In Project.Schematics
                Project.Schematics.Remove(s.Name)
        Next s

        For Each g In Project.Graphs
                Project.Graphs.Remove(g.Name)
        Next g

        Project.OptGoals.RemoveAll

        MWOffice.Windows.Close
End Sub
```

Now, you can run the code over and over, and you will not get multiple schematics.

One final piece to help the development step is to add code to arrange windows and zoom the schematic.

The main subroutine should now look like below:

```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        ArrangeWindows
End Sub
```

Then add the window arranging code after the sub main subroutine.

```
Sub ArrangeWindows
        Dim w As Window
        For Each w In MWOffice.Windows
                If w.Caption = "lpf" Then
                        w.ViewAll
                End If
        Next w

        MWOffice.Windows.Tile(mwWTD_Vertical)

End Sub
```

Now when you run the code, the schematic will be maximized.

Add Elements to the Schematic

Next, we need to build the schematic by adding elements, wires, ports and ground nodes.

The main subroutine should now look like below:

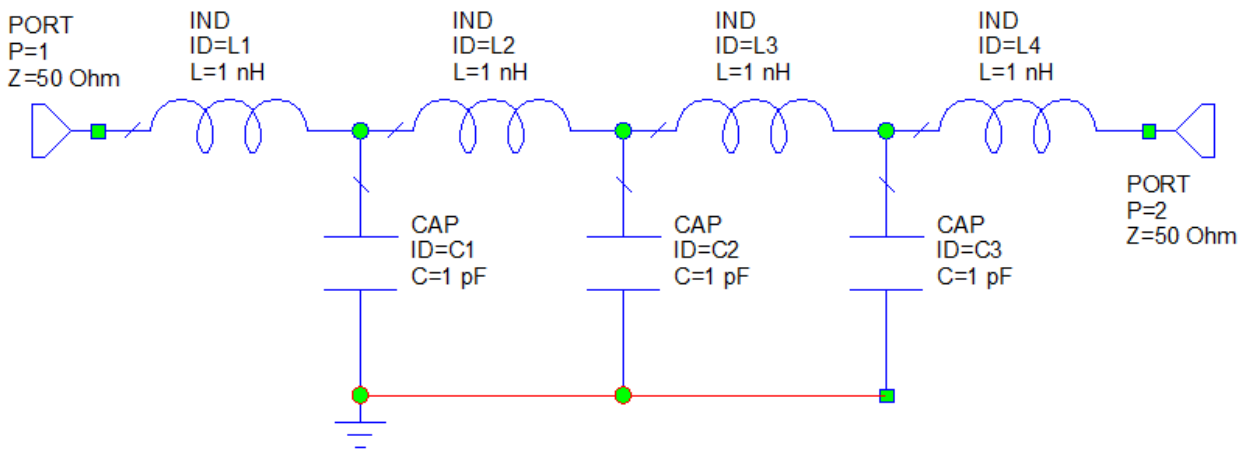
```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        ArrangeWindows
End Sub
```

Then type or add the schematic building code after the sub main subroutine. Notice that we added no wires between each element. When element nodes overlap, they are automatically connected.

```
Sub BuildSchematic(s As Schematic)
        'add inductors
        'each visible grid in the schematic has a value of 100 for the coordinates
        s.Elements.Add("IND",0,0)
        s.Elements.Add("IND",1000,0)
        s.Elements.Add("IND",2000,0)
        s.Elements.Add("IND",3000,0)
        'add capacitors
        s.Elements.Add("CAP",1000,0,270)
        s.Elements.Add("CAP",2000,0,270)
        s.Elements.Add("CAP",3000,0,270)
        'add wire
        s.Wires.Add(1000,1000,3000,1000)
        'add ports
        s.Elements.Add("PORT",0,0)
        s.Elements.Add("PORT",4000,0,180)
        'add ground
        s.Elements.Add("GND",1000,1000)
End Sub
```

This code is a subroutine because we pass no information out of the code at completion. We pass in the schematic object just in case there were naming issues as previously discussed.

After this point, you will see this schematic in a window that is maximized and centered around the elements.



Change Parameter Values

Next, we need to change parameter values in the already created elements. We could have done these value change at the same time as adding the elements. It was done this way to more closely follow the same steps of creating the filter by hand.

The main subroutine should now look like below:

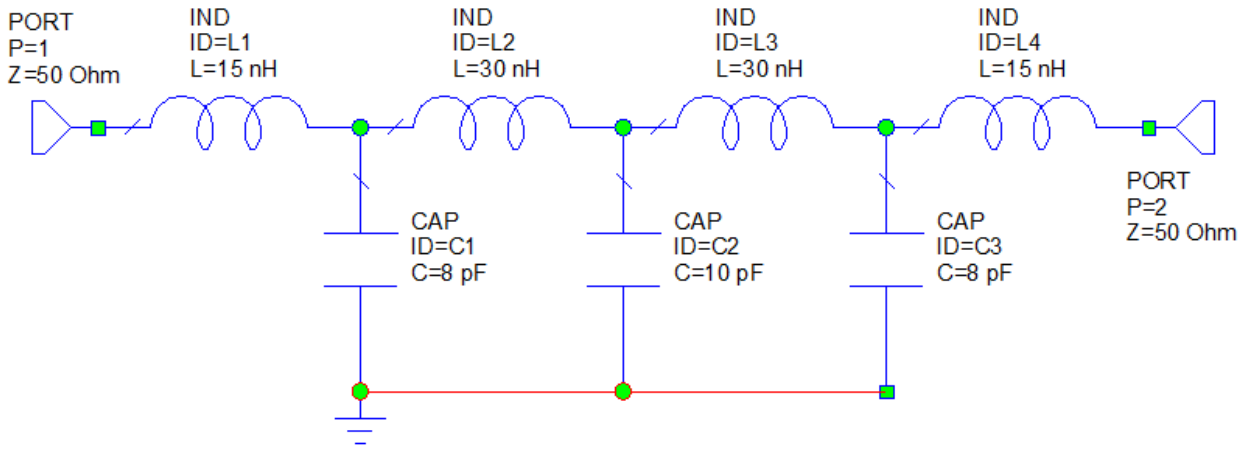
```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

        ArrangeWindows
End Sub
```

Then type or add the code for setting element parameter values after the sub main subroutine. A few things to notice.

1. We reference the elements by the element name, a dot, and then the element ID.
2. We set the values in base units, such as Hz, Henrys, Farads, etc. because each project could have different project units, but the code will still work just fine. The schematic display converts to the proper units.

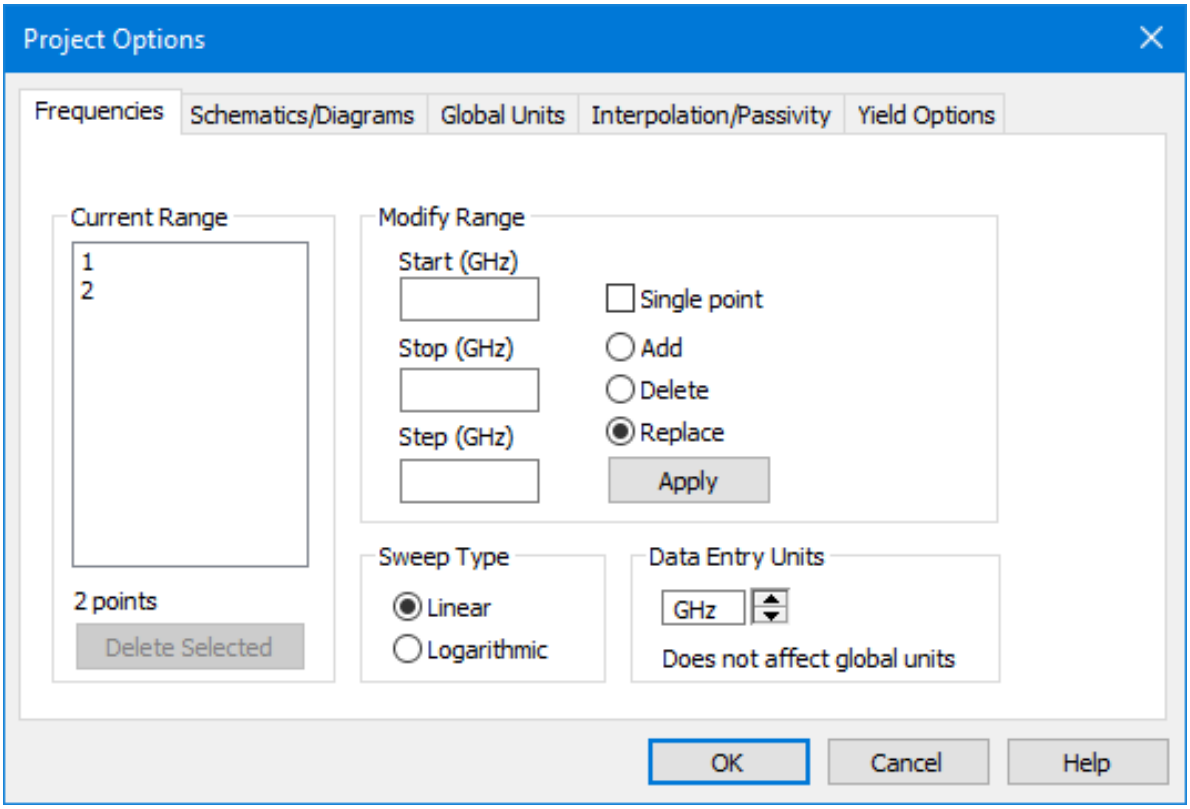
```
Sub SetValues(s As Schematic)
        'sets the parameter values of the elements to be non-default
        'values as double will always be in base units (Farads, Henries, etc)
        s.Elements("IND.L1").Parameters("L").ValueAsDouble = 15e-9
        s.Elements("IND.L2").Parameters("L").ValueAsDouble = 30e-9
        s.Elements("IND.L3").Parameters("L").ValueAsDouble = 30e-9
        s.Elements("IND.L4").Parameters("L").ValueAsDouble = 15e-9
        s.Elements("CAP.C1").Parameters("C").ValueAsDouble = 8e-12
        s.Elements("CAP.C2").Parameters("C").ValueAsDouble = 10e-12
        s.Elements("CAP.C3").Parameters("C").ValueAsDouble = 8e-12

End Sub
```

When you run the script after this point, you will see this schematic in a window that is maximized and centered around the elements with new parameter values.



Change Frequencies

Next, we need to change the simulation frequencies as the defaults of 1, and 2 GHz would not result in meaningful simulation results. If you select **Options > Project Options** from the Microwave Office menus, the frequency list is displayed, this is to show the frequencies before the change.



The main subroutine should now look like below:

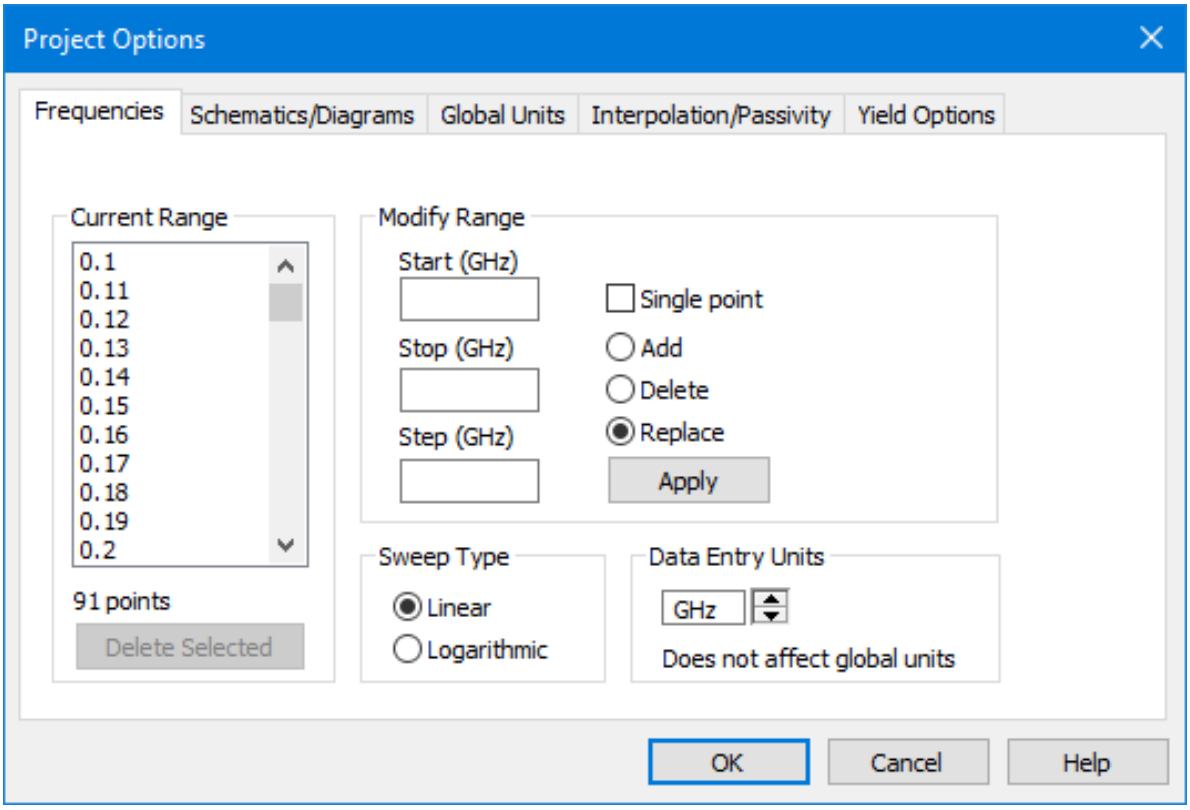
```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

    SetFrequencies

        ArrangeWindows
End Sub
```

Then type or add the frequency setting code after the sub main subroutine. A few things to notice.

1. We must clear the previous frequencies.
2. We are creating an array of doubles, filling in the array and then only doing one call to set the frequencies. This array is the fast way to implement these changes.

```
Sub SetFrequencies
	Dim freqs() As Double
	Dim fstart,fstop,fstep As Double
	Dim num,cnt,i As Integer

	fstart = 100
	fstop = 1000
	fstep = 10
	num = (fstop-fstart)/fstep
	ReDim freqs(num)
	Project.Frequencies.Clear
	cnt = 0
	For i = fstart To fstop Step fstep
		freqs(cnt) = i * 1e6 'enter in base units, Hz.
		cnt = cnt + 1
	Next i
	Project.Frequencies.AddMultiple(freqs)
End Sub
```

After running the code, check the frequencies again to see they have changed.



As an aside, simpler code to add frequencies is below. It is fewer lines but much slower to execute due to all the calls to add frequencies.

```
Sub SetFrequencies
	Dim i As Integer
	Project.Frequencies.Clear
	For i = 100 To 1000 Step 10
		Project.Frequencies.Add(i * 1e6) 'enter in base units, Hz.
	Next i
End Sub
```

Add Graph

Next, we need to add a graph and some measurements to the graph.

The main subroutine should now look like below:

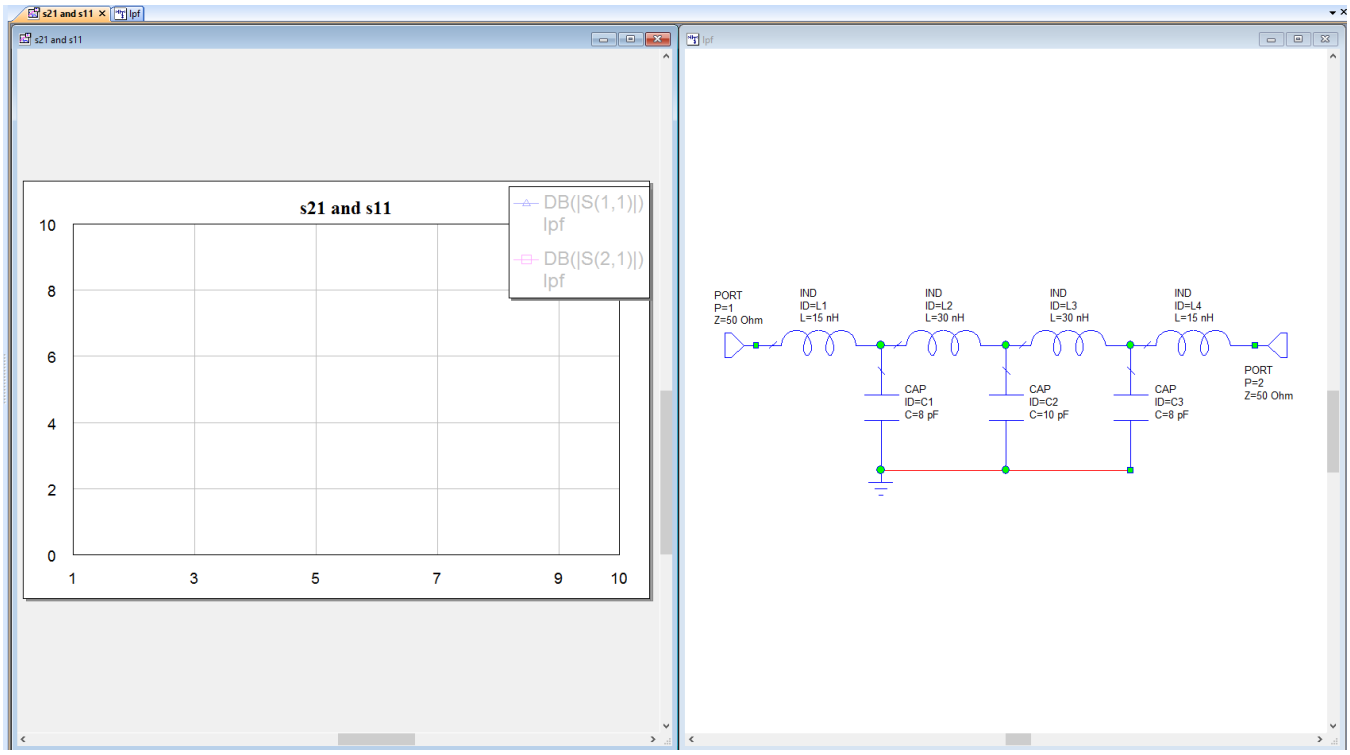
```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

    SetFrequencies

        AddGraph(sch)

        ArrangeWindows
End Sub
```

Then type or add the graph and measurement code after the sub main subroutine. A few things to notice.

1. The auto-complete will help you get the right setting for the graph type.
2. We pass in the schematic so we can get the schematic name as the first argument (i.e. the *SourceDoc*) for the measurement.
3. The measurements have a specific syntax. An easy way to get this is to add the measurement to a project and then use a script to retrieve the measurement syntax.

```
Sub AddGraph(s As Schematic)
        Dim g As Graph
        Set g = Project.Graphs.Add("s21 and s11",mwGT_Rectangular)
        g.Measurements.Add(s.Name,"DB(|S(1,1)|)")
        g.Measurements.Add(s.Name,"DB(|S(2,1)|)")
End Sub
```

After running the code, you will have your schematic and graph tiled.



Note: The code below is sample code to print out measurement syntax. You would add the measurement by hand and then run this code to get measurement syntax correct.

```
 Option Explicit
' Code Module
Sub Main
        Dim g As Graph
        Dim m As Measurement
        Debug.Clear

        For Each g In Project.Graphs
                Debug.Print g.Name
                For Each m In g.Measurements
                        Debug.Print vbTab &  m.Type
                Next m
        Next g
End Sub
```

Simulate

Next, we need to add a graph and some measurements to the graph.

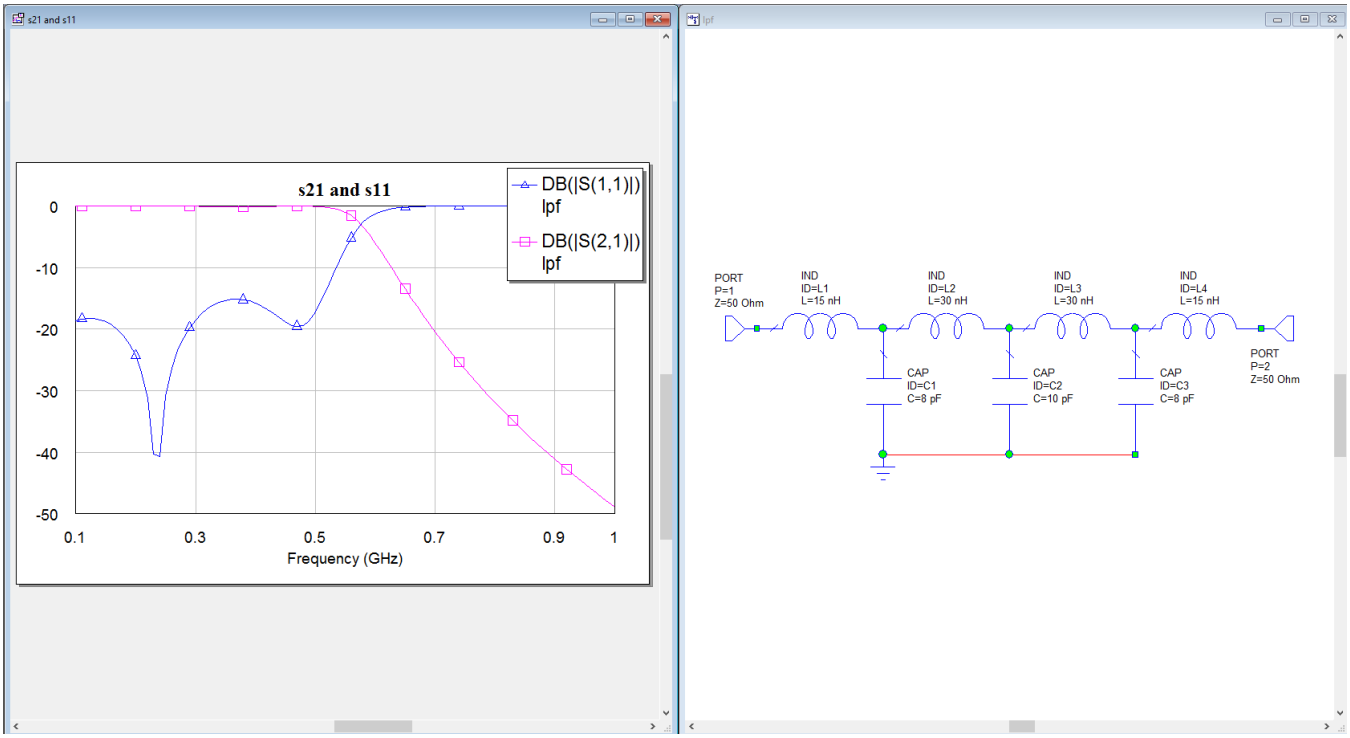
The main subroutine should now look like below:

```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

    SetFrequencies

        AddGraph(sch)

        Simulate

        ArrangeWindows
End Sub
```

Then type or add the simulate code after the sub main subroutine. It is a little silly to add a one-line subroutine. It was done this way for consistency with the rest of the steps.

```
Sub Simulate
        Project.Simulator.Analyze
End Sub
```

After running the code, you will have your schematic and graph tiled now with simulation results.



Adding Markers

Next, we need to add markers to the traces to help find the 3dB roll-off frequency

The main subroutine should now look like below:

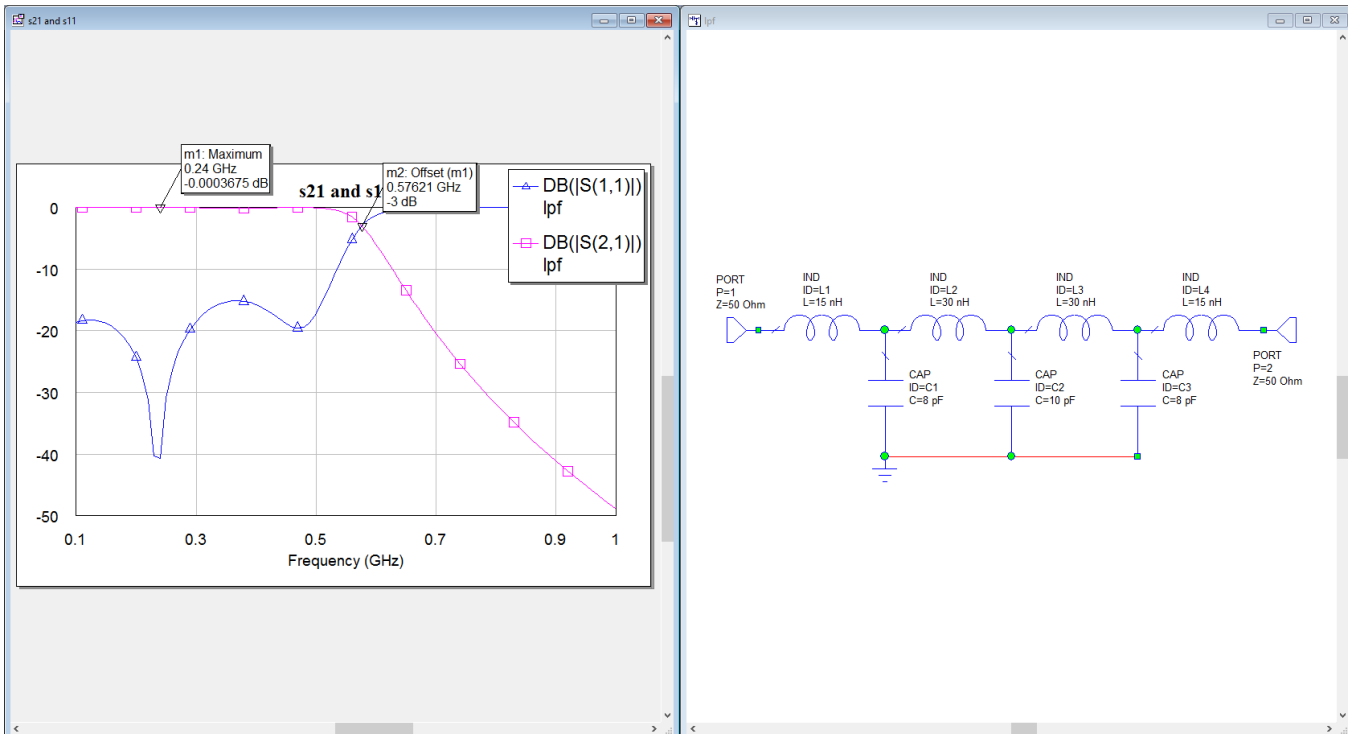
```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

    SetFrequencies

        AddGraph(sch)

        Simulate

        AddMarkers(sch)

        ArrangeWindows
End Sub
```

Then type or add the marker adding code after the sub main subroutine. A few things to notice.

1. After adding the markers, we used marker objects to change their properties.
2. We reference the measurement by its entire string, the schematic passed in gives the name, and then the measurement is added by concatenating strings using the & operator.

```
Sub AddMarkers(s As Schematic)
        Dim m As Marker
        Dim m2 As Marker

        Set m = Project.Graphs(1).Markers.Add(s.Name & ":DB(|S(2,1)|)",1,0.1e9)
        m.Type = mwMT_AutoSearch
        m.AutoSearch.mode=mwMAM_Max
        Set m2 = Project.Graphs(1).Markers.Add(s.Name & ":DB(|S(2,1)|)",1,0.5e9)
        m2.Type =mwMT_Offset
        m2.Offset.Distance = -3
        m2.Offset.mode=mwMOM_Y
        m2.Offset.ReferenceMarker="m1"

End Sub
```

When the code is run, you will have your schematic and graph tiled now with simulation results and markers. You can now read the 3D roll-off frequency from the "m2" marker.



Adding Equations

Next, we need to add equations and assign some parameters to use these equations. We know the filter needs to be symmetric and this is one way to create this symmetry.

The main subroutine should now look like below:

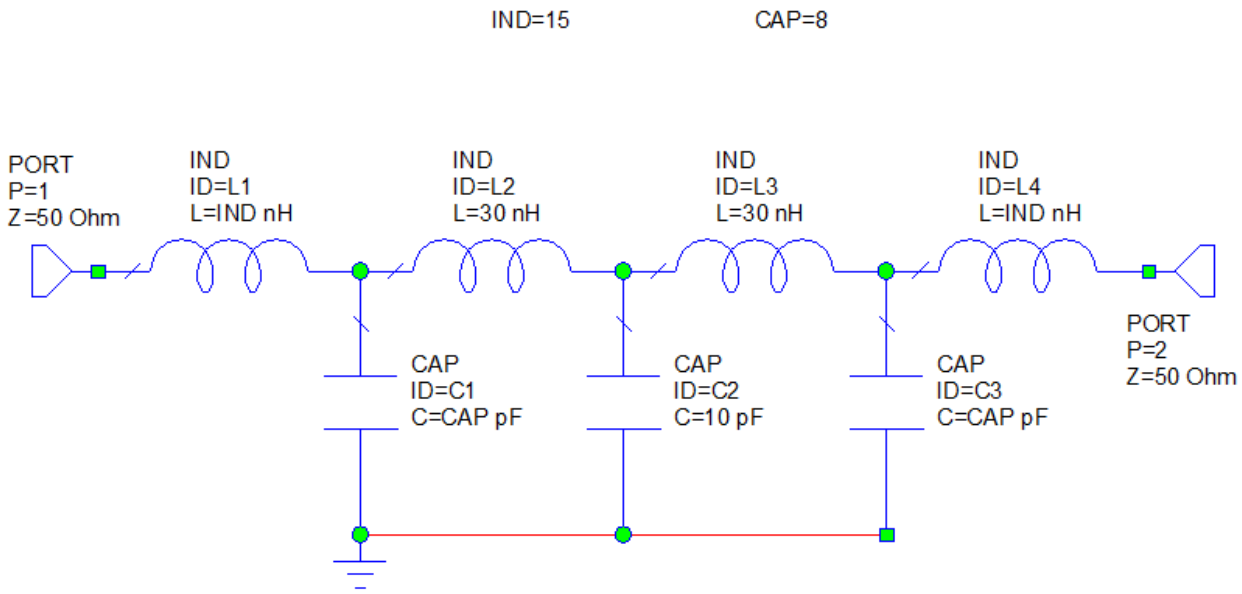
```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

        SetFrequencies

        AddGraph(sch)

        Simulate

        AddMarkers(sch)

        AddEquations(sch)

        ArrangeWindows
End Sub
```

Then type or add the equation adding code after the sub main subroutine. A few things to notice.

1. Equations are similar to adding elements with the location. However, you enter the exact equation syntax; equations don't have a concept of units.
2. When assigning parameters to equations, we use the '.ValueAsString' command.

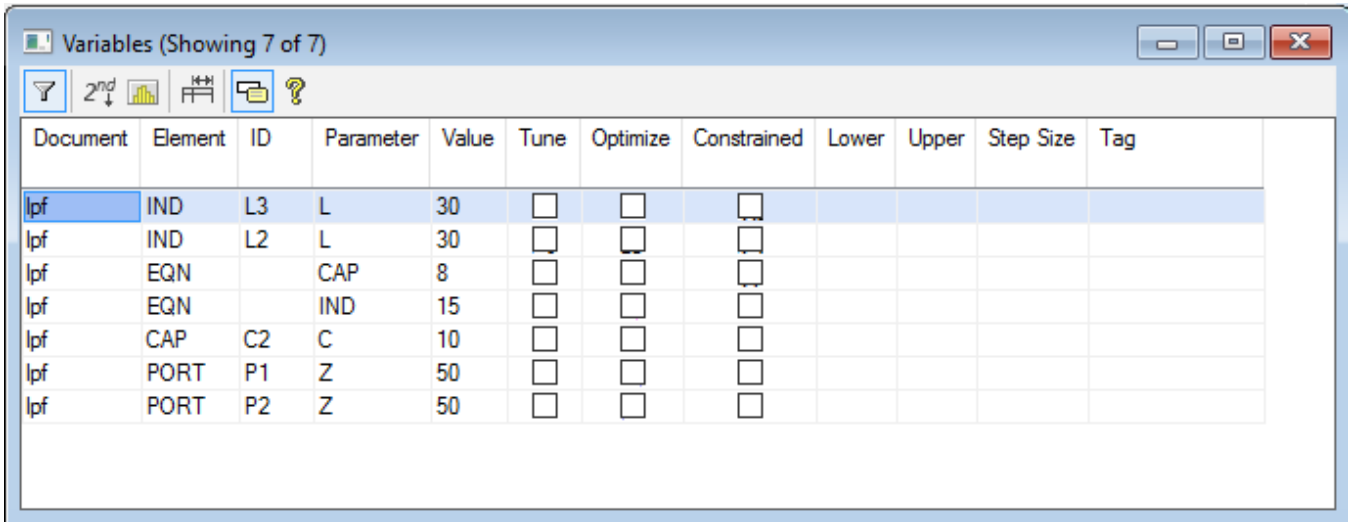
```
Sub AddEquations(s As Schematic)
        'add equations and assign parameters to use the equation.
        s.Equations.Add("IND=15",1500,-1000)
        s.Equations.Add("CAP=8",2500,-1000)
        s.Elements("IND.L1").Parameters("L").ValueAsString = "IND"
        s.Elements("IND.L4").Parameters("L").ValueAsString = "IND"
        s.Elements("CAP.C1").Parameters("C").ValueAsString = "CAP"
        s.Elements("CAP.C3").Parameters("C").ValueAsString = "CAP"
End Sub
```

After this point, you will see this schematic in a window that is maximized and centered around the elements with the new equations. The graph at this point will be grayed out because the schematic has been edited and not yet simulated again. You could call the simulate function again if you wanted to at this point.



Setting Optimization Parameters and Limits

Next, we need to set up the parameters we want to optimize and set optimization limits. The best way to see the before and after is to open the variable browser, **View > Variable Browser**.



Document	Element	ID	Parameter	Value	Tune	Optimize	Constrained	Lower	Upper	Step Size	Tag
lpf	IND	L3	L	30	☐	☐	☐				
lpf	IND	L2	L	30	☐	☐	☐				
lpf	EQN		CAP	8	☐	☐	☐				
lpf	EQN		IND	15	☐	☐	☐				
lpf	CAP	C2	C	10	☐	☐	☐				
lpf	PORT	P1	Z	50	☐	☐	☐				
lpf	PORT	P2	Z	50	☐	☐	☐				

The main subroutine should now look like below:

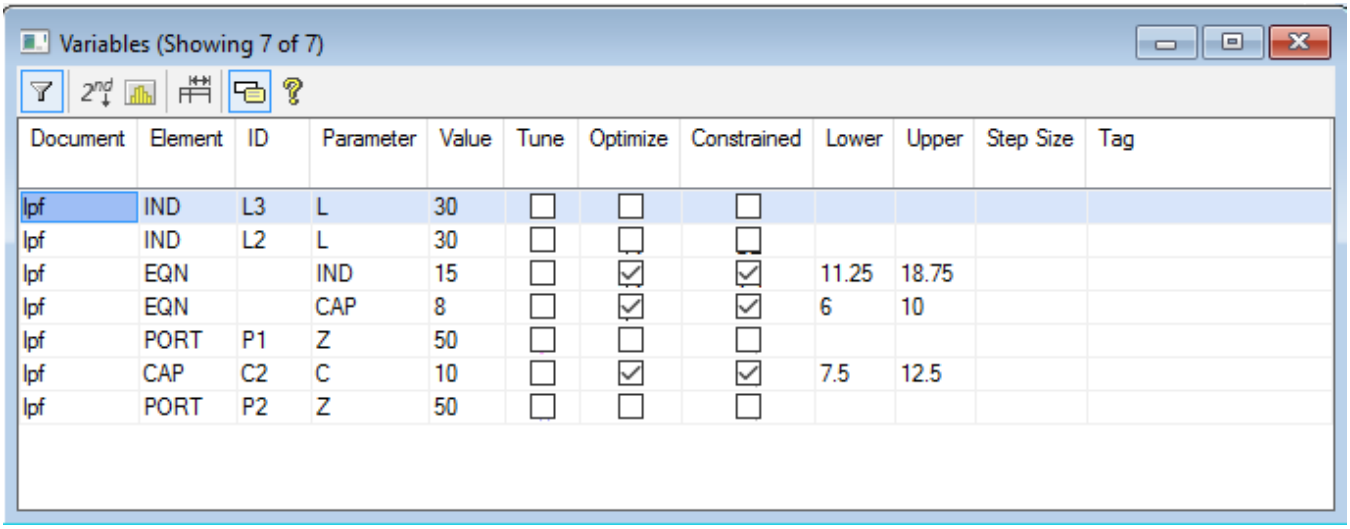
```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

        SetFrequencies

        AddGraph(sch)

        Simulate

        AddMarkers(sch)

        AddEquations(sch)

        OptLimits(sch)

        ArrangeWindows
End Sub
```

Then type or add the optimization variables and limits code after the sub main subroutine. A few things to notice.

1. If we don't set the '.Constrain=True', then the limits won't matter.
2. The goal was to set the lower and upper limits 25% of the nominal value. The code cheats a bit as we know the parameter values from previous steps. It would be cleaner to get the parameter value and then apply the percentages.

```
Sub OptLimits(s As Schematic)
        'setup various values for optimization including constraints.
        s.Equations("CAP").Optimize=True
        s.Equations("CAP").Constrain=True
        s.Equations("CAP").LowerConstraint = 8*0.75
        s.Equations("CAP").UpperConstraint = 8*1.25
        s.Equations("IND").Optimize=True
        s.Equations("IND").Constrain=True
        s.Equations("IND").LowerConstraint = 15*0.75
        s.Equations("IND").UpperConstraint = 15*1.25
        s.Elements("CAP.C2").Parameters("C").Optimize=True
        s.Elements("CAP.C2").Parameters("C").Constrain=True
        s.Elements("CAP.C2").Parameters("C").LowerConstraint = 10*0.75*1e-12
        s.Elements("CAP.C2").Parameters("C").UpperConstraint = 10*1.25*1e-12
End Sub
```

If you check the variable view again after this code, it will look like below.



Document	Element	ID	Parameter	Value	Tune	Optimize	Constrained	Lower	Upper	Step Size	Tag
lpf	IND	L3	L	30	☐	☐	☐				
lpf	IND	L2	L	30	☐	☐	☐				
lpf	EQN		IND	15	☐	☑	☑	11.25	18.75		
lpf	EQN		CAP	8	☐	☑	☑	6	10		
lpf	PORT	P1	Z	50	☐	☐	☐				
lpf	CAP	C2	C	10	☐	☑	☑	7.5	12.5		
lpf	PORT	P2	Z	50	☐	☐	☐				

Adding Optimization Goals

Next, we need to add the optimization goals.

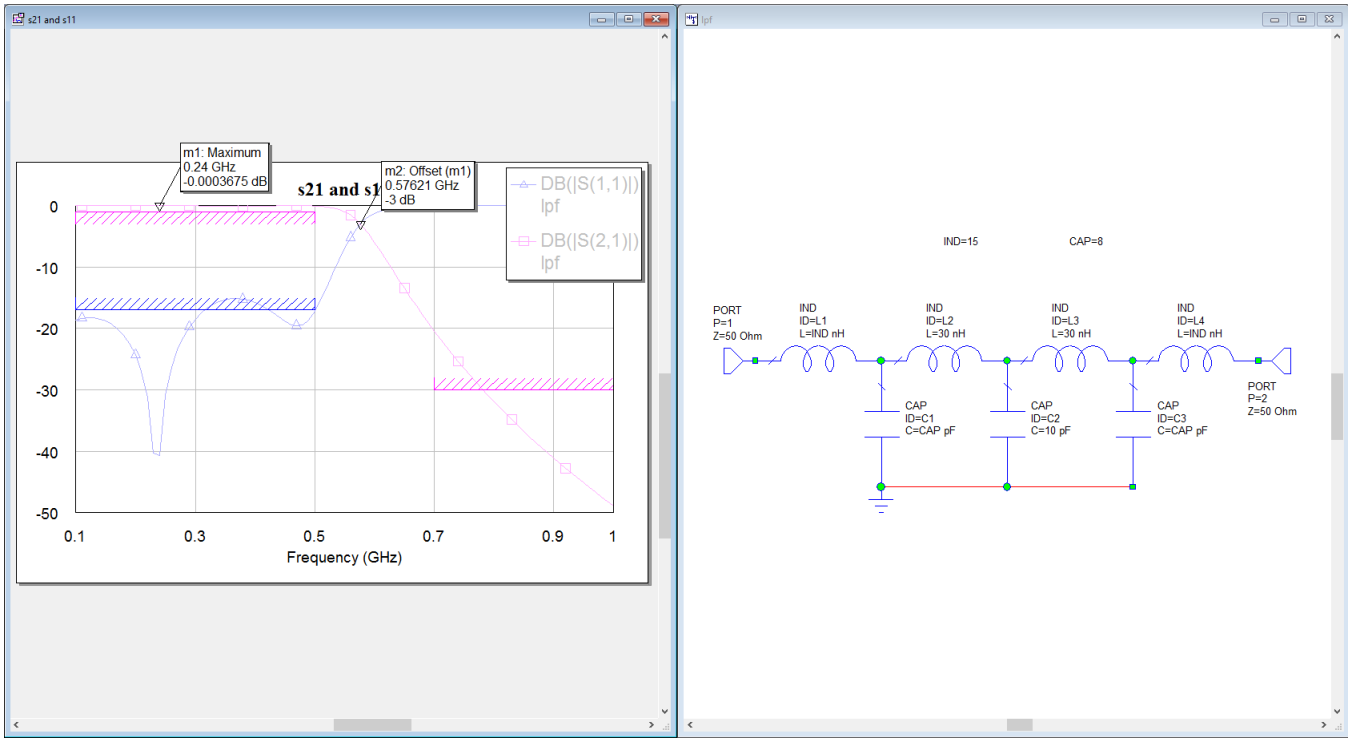
The main subroutine should now look like below:

```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

        SetFrequencies

        AddGraph(sch)

        Simulate

        AddMarkers(sch)

        AddEquations(sch)

        OptLimits(sch)

        SetOptGoals(sch)

        ArrangeWindows
End Sub
```

Then type or add the optimization goals code after the sub main subroutine. The syntax is a bit involved and usually takes a little trial and error to get it right.

```
Sub SetOptGoals(s As Schematic)
        Project.OptGoals.Add(s.Name,"DB(|S(1,1)|)",mwOGT_LessThan,1,2,0,500e6,mwUT_Frequency,-17,-17,mwUT_DB)
        Project.OptGoals.Add(s.Name,"DB(|S(2,1)|)",mwOGT_GreaterThan,1,2,0,500e6,mwUT_Frequency,-1,-1,mwUT_DB)
        Project.OptGoals.Add(s.Name,"DB(|S(2,1)|)",mwOGT_LessThan,1,2,700e6,1000e6,mwUT_Frequency,-30,-30,
mwUT_DB)
End Sub
```

After running the code, you will have your schematic and graph tiled now with simulation results and see the optimization goals on the graph.



Optimize

Next, we will optimize the circuit.

The main subroutine should now look like below:

```
Option Explicit
' Code Module
Sub Main
	Dim sch As Schematic

	CleanUp

	Set sch = CreateSchematic

	BuildSchematic(sch)

	SetValues(sch)

	SetFrequencies

	AddGraph(sch)

	Simulate

	AddMarkers(sch)

	AddEquations(sch)

	OptLimits(sch)

	SetOptGoals(sch)

	ArrangeWindows

	Optimize
End Sub
```

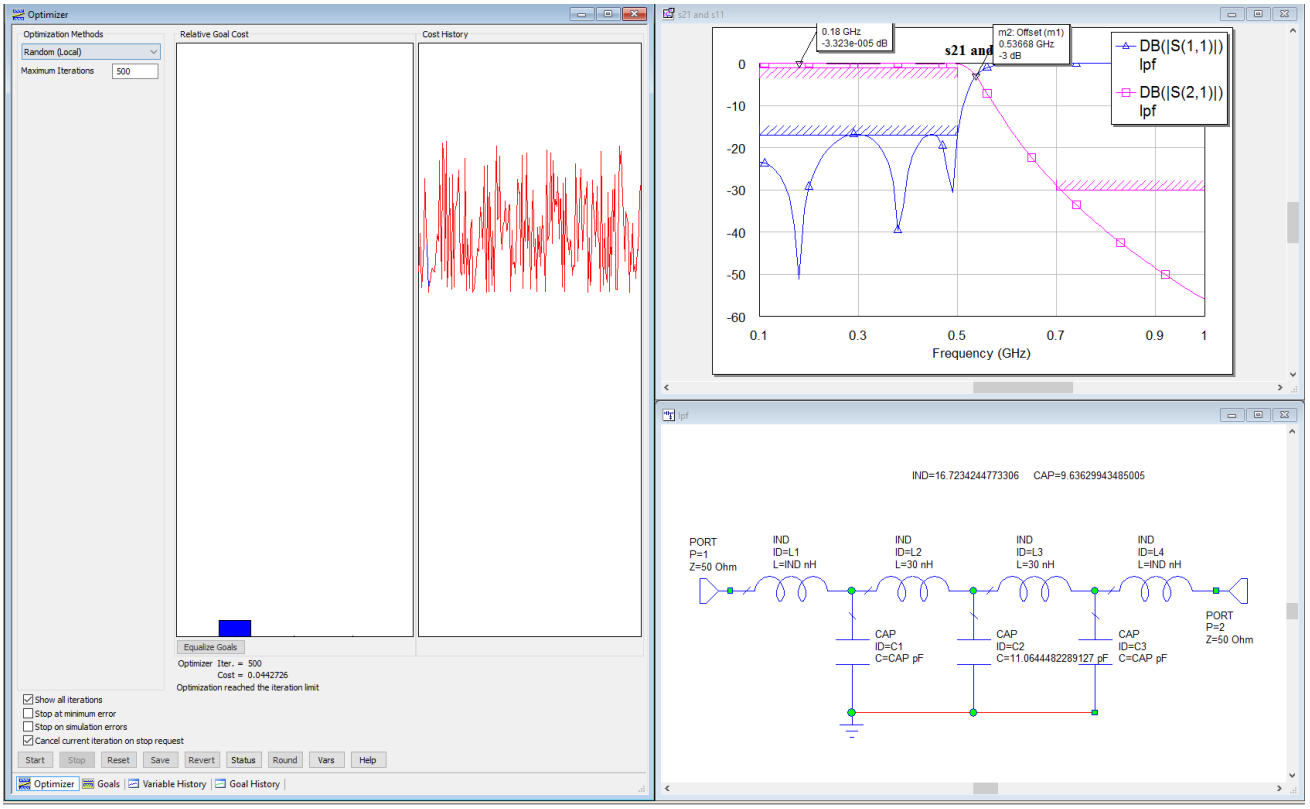
Notice the optimization step is coming after arranging the windows, so we have a reasonable window arrangement while watching the optimization happen.

Then type or add the optimization goals code after the sub main subroutine. A few things to notice.

1. There is a helper function that helps set the optimizer type. The type is set by an integer and not name, so the function finds the name and returns the integer. Different versions of the software can change the order or number of optimizers, so this is the best way to use the right optimizer.
2. There is a loop to wait for the optimizer to be done as we don't know in the code how long this will take.

```
Sub Optimize
	Project.Optimizer.MaxIterations = 500
	Project.Optimizer.Type = find_opt_type("Random (Local)")
	Project.Optimizer.NewWindow
	Project.Optimizer.Start
	While Project.Optimizer.Running=True
		Wait(0.5)
	Wend
End Sub

Function find_opt_type(nm As String) As Integer
	Dim typ As Integer
	Dim i As Integer
	typ = -1
	For i = 1 To Project.Optimizer.TypeCount
		If nm = Project.Optimizer.TypeName(i) Then
			typ = i
		End If
	Next i
	If typ = -1 Then
		MsgBox ("could not find optimizer name specified:" & nm)
	End If
	find_opt_type = typ
End Function
```

When the code is run, you will have your schematic and graph tiled now with simulation results and the optimization running with the optimization window opened.



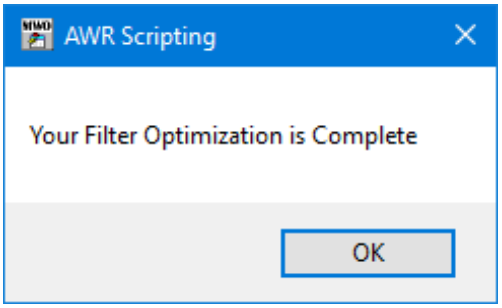
Script Complete

Since the optimization will take some time, it is a good idea to display a message box upon completion.

The main subroutine should now look like below:

```
Option Explicit
' Code Module
Sub Main
        Dim sch As Schematic

        CleanUp

        Set sch = CreateSchematic

        BuildSchematic(sch)

        SetValues(sch)

        SetFrequencies

        AddGraph(sch)

        Simulate

        AddMarkers(sch)

        AddEquations(sch)

        OptLimits(sch)

        SetOptGoals(sch)

        ArrangeWindows

        Optimize

        MsgBox("Your Filter Optimization is Complete")
End Sub
```

When the script is done, you will see the message box shown below.



Some Topics to Think About

Option Explicit

This option was used in the guide and we highly recommend you use it since it forces all variables used to be defined and properly typed. VB is not strongly typed, so it's easy to get "spaghetti" code going. This option helps to cut down on it and helps the user instantly find typos in variable names.Option Explicit

F1 Help

When the cursor is in a word in the scripting editor, type the **F1** button for help on that keyword. F1 Help in Scripting

Adding a Form (also called a Custom Dialog Box)

You may need to build a custom dialog for your script. This dialog is made easy by having a graphical user interface to develop your dialog. Scripting How-To: Adding a Custom Dialog

Dictionaries

Dictionaries are an incredibly useful concept when working with visual basic. They are a great way collect up lists of items, especially if you don't know how many items you will need to have on your list. You can think of a dictionary like a classic dictionary; there are two pieces of information. In the classic dictionary there is a word and then the definition. In the dictionary object, there is a key and an item. The key is the word, and the item is the definition. Please see the article for details on using dictionaries. Scripting How-To: Using a Dictionary in AWR Scripting

File System Objects

The file system objects allow working with files and folders very quickly. For example, if you need to process a folder and all subfolders and look for files, these utilities make that very easy. Please see the article for details on using file system objects. Scripting How-To: File System Objects and Scripting How-To:How Can I Easily Scan Drives, Directories, Files in a Directory, and Perform Basic File Operations

Browse to a File

A simple command to get a dialog to box to ask the user to browse to a file. Scripting How-To: Using GetFilePath

Browse to a Folder

Not as simple, but sometimes you want to get a folder location instead of a file location. Scripting How-To: How to Prompt for a Folder in AWR Scripting

The Dialog Function

The dialog function allows for advanced dialog functionality such as initializing the state of the dialog, changing if items are enabled or not based on other actions, keeping the dialog open after command completion, etc. Scripting How-To:Generating S-Parameter files from AWR Scripting and Scripting How-To: Creating a Dialog with Status Text and a Progress Bar in AWR Scripting. Some shipping scripts also heavily use the dialog function, Generate MDIF from Collection of Files and Modify Graph Properties

Referencing Other Modules

You may want to use code in other modules; there are various ways to do this. Scripting How-To: Referencing Subs and Functions in Other Files

Python

The examples so far have shown working with the AWR SDE. Python is a common scripting language and could also be used to accomplish the same tasks (see AWR Scripting in Python for setup). We show the same code to complete the same functionality in python below.

```
# -*- coding: utf-8 -*-
# python 3

# Script to create LPF_lumped linear portion through scripting
# To make it easier to match up to the Visual Basic script the
# function names are CamelCase.

import win32com.client as win32
import time


# we're setting up the connection to AWRDE as a global along with the constants from the type library.
awr = win32.Dispatch("MWOApp.MWOffice")
awrc = win32.constants


def one_based_range(i):
    return range(1, i + 1)


def Cleanup():
    for s in awr.Project.Schematics:
        awr.Project.Schematics.Remove(s.Name)
    for g in awr.Project.Graphs:
        awr.Project.Graphs.Remove(g.Name)
    awr.Project.OptGoals.RemoveAll()
    awr.Windows.Close()
```

```
def CreateSchematic():
    s = awr.Project.Schematics.Add("lpf")
    return s


def BuildSchematic(s):
    """ add inductors
        each visible grid in the schematic has a value of 100 for the coordinates"""
    awr.Project.Schematics(s.Name).Elements.Add("IND", 0, 0)
    awr.Project.Schematics(s.Name).Elements.Add("IND", 1000, 0)
    awr.Project.Schematics(s.Name).Elements.Add("IND", 2000, 0)
    awr.Project.Schematics(s.Name).Elements.Add("IND", 3000, 0)
    "add capacitors"
    awr.Project.Schematics(s.Name).Elements.Add("CAP", 1000, 0, 270)
    awr.Project.Schematics(s.Name).Elements.Add("CAP", 2000, 0, 270)
    awr.Project.Schematics(s.Name).Elements.Add("CAP", 3000, 0, 270)
    "add wire"
    awr.Project.Schematics(s.Name).Wires.Add(1000, 1000, 3000, 1000)
    "add ports"
    awr.Project.Schematics(s.Name).Elements.Add("PORT", 0, 0)
    awr.Project.Schematics(s.Name).Elements.Add("PORT", 4000, 0, 180)
    "add ground"
    awr.Project.Schematics(s.Name).Elements.Add("GND", 1000, 1000)


def SetValues(s):
    """ sets the parameter values of the elements to be non-default
        values as double will always be in base units (Farads, Henries, etc)"""
    awr.Project.Schematics(s.Name).Elements("IND.L1").Parameters("L").ValueAsDouble = 15e-9
    awr.Project.Schematics(s.Name).Elements("IND.L2").Parameters("L").ValueAsDouble = 30e-9
    awr.Project.Schematics(s.Name).Elements("IND.L3").Parameters("L").ValueAsDouble = 30e-9
    awr.Project.Schematics(s.Name).Elements("IND.L4").Parameters("L").ValueAsDouble = 15e-9
    awr.Project.Schematics(s.Name).Elements("CAP.C1").Parameters("C").ValueAsDouble = 8e-12
    awr.Project.Schematics(s.Name).Elements("CAP.C2").Parameters("C").ValueAsDouble = 10e-12
    awr.Project.Schematics(s.Name).Elements("CAP.C3").Parameters("C").ValueAsDouble = 8e-12


def ArrangeWindows():
    for w in awr.Windows:
        w.ViewAll()
    awr.Windows.Tile(1)


def SetFrequencies():
    freqs = [f * 1e6 for f in range(100, 1000, 10)]
    awr.Project.Frequencies.Clear
    awr.Project.Frequencies.AddMultiple(freqs)


def AddGraph(s):
    g = awr.Project.Graphs.Add("s21 and s11", awrc.mwGT_Rectangular)
    g.Measurements.Add(s.Name, "DB(|S(1,1)|)")
    g.Measurements.Add(s.Name, "DB(|S(2,1)|)")


def Simulate():
    awr.Project.Simulator.Analyze()


def AddMarkers(s):
    m = awr.Project.Graphs(1).Markers.Add(s.Name + ":DB(|S(2,1)|)", 1, 0.1e9)
    m.Type = awrc.mwMT_AutoSearch
    m.AutoSearch.mode = awrc.mwMAM_Max
    m2 = awr.Project.Graphs(1).Markers.Add(s.Name + ":DB(|S(2,1)|)", 1, 0.5e9)
    m2.Type = awrc.mwMT_Offset
    m2.Offset.Distance = -3
    m2.Offset.mode = awrc.mwMOM_Y
    m2.Offset.ReferenceMarker = "m1"


def AddEquations(s):
```

```
    awr.Project.Schematics(s.Name).Equations.Add("IND=15", 1500, -1000)
    awr.Project.Schematics(s.Name).Equations.Add("CAP=8", 2500, -1000)
    awr.Project.Schematics(s.Name).Elements("IND.L1").Parameters("L").ValueAsString = "IND"
    awr.Project.Schematics(s.Name).Elements("IND.L4").Parameters("L").ValueAsString = "IND"
    awr.Project.Schematics(s.Name).Elements("CAP.C1").Parameters("C").ValueAsString = "CAP"
    awr.Project.Schematics(s.Name).Elements("CAP.C3").Parameters("C").ValueAsString = "CAP"


def OptLimits(s):
    awr.Project.Schematics(s.Name).Equations("CAP").Optimize = True
    awr.Project.Schematics(s.Name).Equations("CAP").Constrain = True
    awr.Project.Schematics(s.Name).Equations("CAP").LowerConstraint = 8 * 0.75
    awr.Project.Schematics(s.Name).Equations("CAP").UpperConstraint = 8 * 1.25
    awr.Project.Schematics(s.Name).Equations("IND").Optimize = True
    awr.Project.Schematics(s.Name).Equations("IND").Constrain = True
    awr.Project.Schematics(s.Name).Equations("IND").LowerConstraint = 15 * 0.75
    awr.Project.Schematics(s.Name).Equations("IND").UpperConstraint = 15 * 1.25
    awr.Project.Schematics(s.Name).Elements("CAP.C2").Parameters("C").Optimize = True
    awr.Project.Schematics(s.Name).Elements("CAP.C2").Parameters("C").Constrain = True
    awr.Project.Schematics(s.Name).Elements("CAP.C2").Parameters("C").LowerConstraint = 10 * 0.75 * 1e-12
    awr.Project.Schematics(s.Name).Elements("CAP.C2").Parameters("C").UpperConstraint = 10 * 1.25 * 1e-12


def SetOptGoals(s):
    awr.Project.OptGoals.Add(s.Name, "DB(|S(1,1)|)", 1, 1, 2, 0, 500e6, 1, -17, -17, 15)
    awr.Project.OptGoals.Add(s.Name, "DB(|S(2,1)|)", 2, 1, 2, 0, 500e6, 1, -1, -1, 15)
    awr.Project.OptGoals.Add(s.Name, "DB(|S(2,1)|)", 1, 1, 2, 700e6, 1000e6, 1, -30, -30, 15)


def Optimize():
    awr.Project.Optimizer.MaxIterations = 500
    opt_type = find_opt_type("Random (Local)")
    if opt_type:
        awr.Project.Optimizer.Type = opt_type
        awr.Project.Optimizer.NewWindow()
        awr.Project.Optimizer.Start()
        while awr.Project.Optimizer.Running:
            time.sleep(0.5)


def find_opt_type(nm):
    typ = None
    for i in one_based_range(awr.Project.Optimizer.TypeCount):
        if nm == awr.Project.Optimizer.TypeName(i):
            typ = i
    if typ is None:
        print("could not find optimizer name specified:" + nm)
    return typ


if __name__ == '__main__':
    # main program
    Cleanup()
    sch = CreateSchematic()
    BuildSchematic(sch)
    SetValues(sch)
    SetFrequencies()
    AddGraph(sch)
    Simulate()
    AddMarkers(sch)
    AddEquations(sch)
    ArrangeWindows()
    OptLimits(sch)
    SetOptGoals(sch)
    Optimize()
```