

# **Expedition PCB Flow: Automation and Scripting**

Lab Workbook



Copyright © Mentor Graphics Corporation 2006. All rights reserved. This document contains information that is proprietary to Mentor Graphics Corporation and may not be duplicated in whole or in part in any form without written consent from Mentor Graphics. In accepting this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use of this information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### **RESTRICTED RIGHTS LEGEND 03/97**

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is:  
Mentor Graphics Corporation  
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

A complete list of trademark names appears in a separate [“Trademark Information”](#) document.

This is an unpublished work of Mentor Graphics Corporation.

Part Number: 070839

# Trademark Information

## The following are trademarks of Mentor Graphics Corporation:

3D Design, ABIST, ARC4, Arithmetic BIST, Accelerated Technology®, AccuPARTner, AccuParts, AccuSim®, ADEPT, ADVance, ADVance MS, ADVance RFIC, ADVance VCB, AMPLE, Analog Analyst, Analog Station, Ares®, ARTgrid, ArtRouter, ARTshape, ASICPlan, ASICVector Interfaces, Aspire, AuthExpress, AutoActive®, AutoCells, AutoDissolve, AutoFilter, AutoFlow, AutoLib, AutoLinear, AutoLink, AutoLogic, AutoLogic BLOCKS, AutoLogic FPGA, AutoLogic VHDL®, AutomotiveLib, AutoPAR®, AutoTherm®, AutoTherm Duo, Auto ThermMCM, AutoView, Autowire Station, AXEL, AXEL Symbol Genie, A-XGMAC, BISTArchitect, BLAST, Blaze, BlazeRouter, Board Station Consumer, Board Architect, Board Designer, Board Layout, Board Process Library, BoardSim®, Board Station®, BOLD Administrator, BOLD Browser, BOLD Composer, Bridgepoint®, BSDArchitect, BSPBuilder, Buy on Demand, Cable Analyzer, Cable Station, CAECO Designer, CAEFORM, Calibre®, Calibre DRC, Calibre DRC-H, Calibre DESIGNrev, Calibre CB, Calibre FRACTUREh, Calibre FRACTUREj, Calibre FRACTUREm, Calibre FRACTUREt, Calibre FRACTUREk, Calibre FRACTUREv, Calibre LFD, Calibre LITHOview, Calibre LVS-H, Calibre MDP Embedded SVRF, Calibre MDPmerge, Calibre MDPview, Calibre MDPstat, Calibre MDPverify, Calibre MPCpro, Calibre MTflex, Calibre OPCsbar, Calibre OPCverify, Calibre OPCpro, Calibre ORC, Calibre PRINTimage, Calibre PSMgate, Calibre PSMcheck, Calibre TDopc, Calibre WORKbench, Calibre RVE, Calibre MGC, Calibre Interactive, Calibre Verification Center, Calibre MDPview, Calibre xRC, Capital®, Capital Analysis, Capital Archive, Capital Bridges, Capital Documents, Capital H®, Capital H the complete desktop engineer®, Capital Harness, Capital Harness Systems, Capital Insight, Capital Integration, Capital Manager, Capital Manufacture, Capital Support, Capital Systems, Capture Station®, Catapult, Celaro, Cell Builder, Cell Station®, CellFloor, CellGen, CellGraph, CellPlace, CellPower, CellRoute, Centricity, CEOC, Chameleon ART, ChaseX, CheckMate, CHEOS, Chip Station®, ChipGraph, ChipLister, Circuit PathFinder, Co-Verification Environment, COLsim, Code/lab®, CommLib, CommLib BMC, Concurrent Design Environment, Connectivity Dataport, Constraint Editor System, Continuum, Continuum Power Analyst, CoreAlliance, CoreBIST, Core Builder, Core Factory, CTIntegrator, DataCentric Model, DataFusion, Datapath, Data Solvent, dBUG, Debug Detective, DC Analyzer, DeltaV, DesignAnalyst, Design Architect®, Design Architect®-IC, Design Architect Elite, Design Capture, Design Exchange®, Design Manager, Design Station®, DesignBook®, DesignView, DesktopASIC, Destination PCB®, Destiny RE, DFTAdvisor, DFTArchitect, DFTInsight, DMS, DMS Xchange, DxAnalog, DxDataBook, DxDesigner, DxLibraryStudio, DxParts, DxPDF, DxViewOnly, DxVariantManager, Direct System Verification, DSV, Documentation Station, DSS (Decision Support System), DxAnalog, DxDataManager, DxDesigner, DxEnterprise for Agile, DxMatrix, DxSim, DxViewDraw, E3Lcable, EDA Tech Forum®, EDT, Eldo®, EldoNet, ePartners, EParts®, EPlanner®, eProduct Designer, EProduct Services®, Empowering Solutions, Engineer's Desktop, EngineerView, Enterprise Librarian, ENRead, ENWrite, ESim, Exemplar, ExemplarLogic, Expedition, Explorer CAECO Layout, Explorer CheckMate, CheckerWare, Explorer Datapath, Explorer Lsim, Explorer Lsim-C, Explorer Lsim-S, Explorer Ltime, Explorer Schematic, Explorer VHDLsim, ExpressI/O, EZwave, FabLink, Falcon®, Falcon Framework®, FastScan, FastStart, Fire, First-Pass Design Success, First-Pass Success, FlexSim, FlexTest, FDL (Flow Definition Language), FlowTabs, FlowXpert, FORMA, FormalPro, FPGA Advantage®, FPGAAdvisor, FPGA BoardLink, FPGA Builder, FPGASim, FPGA Station®, FPGA Xchange, FrameConnect, Fusion, Galileo®, Gate Station®, GateGraph, GatePlace, GateRoute, GDT®, GDT Core, GDT Designer, GDT Developer, GENIE, GenWare, Geom Genie, HDL2Graphics, HDL Architect, HDL Architect Station, HDL Assistant, HDL Author, HDL Designer, HDL Designer Series, HDL Detective, HDL Inventor, HDL Link, HDL Pilot, HDL Processor, HDLSim, HDLWrite, Hierarchical Injection, Hierarchy Injection, HIC rules, Hardware Modeling, Hybrid Designer, Hybrid Station®, HyperLynx®, HyperSuite®, IC Design Station, IC Designer, IC Layout Station, IC Station®, Streamview, ICbasic, ICblocks, ICcheck, ICcompact, ICdevice, ICextract, ICGen, ICgraph, ICLink, IClister, ICplan, ICRT Controller Lcompiler, ICrules,

ICstudio, ICtrace, ICverify, ICview, ICX®, ICX Active, ICX Plan, ICX Pro, ICX Sentry, ICX Tau, ICX Verify, ICX Vision, ICX Custom Model, ICX Custom Modeling, ICX Project Modeling, ICX Standard Library, IDEA Series, Idea Station®, IKOS®, In All The Right Places, INFORM®, IFX, Inxelia, Innovate PCB, Innoveda, Integrated Product Development, Integra Station®, Integration Tool Kit, IT, INTELLITEST®, Interactive LAYOUT, Interconnect Table, Interface-Based Design, IB, Inventra, Inventra IPX, Inventra Soft Cores, IP Engine, IP Evaluation Kit, IP Factory, IP-PCB, IP QuickUse, IPSim, IS Analyzer, IS Floorplanner, IS MultiBoard, IS Optimizer, IS Synthesizer, iSolve, IV'locity, JobSpy, Language Neutral Licensing, Latium®, LAYOUT, LNL, LBIST, LBISTArchitect, Lc, Lcore, Leaf Cell Toolkit, Led, LED Layout, Leonardo®, LeonardoInsight, LeonardoSpectrum, . Librarian, Library Builder, Library, HotPlot®, Library Manager, LineSim®, Logic Analyzer on a Chip, Logic Builder, Logical Cable, LogicLib, logio, Lsim, Lsim DSM, Lsim Gate, LsimNet, Lsim Power Analyst, Lsim Review, Lsim Switch, Lsim XL, Mach PA, Mach TA, ManufactureView, Manufacturing Advisor, Manufacturing Cable, MaskCompose, MBIST, MBISTArchitect, MBIST Full-Speed, MBIST Flex, MBIST In-Place, MBIST Manager, MCM Designer, MCM Station®, MDV, MegaFunction, Memory Builder, Memory Builder Conductor, Memory Builder Mozart, Memory Designer, Memory Model Builder, Mentor®, Mentor Graphics®, MicroPlan, MicroRoute, Microtec®, Mixed-Signal Pro, ModelEditor, ModelSim®, ModelSim LN, ModelSim VHDL, ModelSim VLOG, ModelSim SE, ModelStation®, Model Technology, ModelViewer, ModelViewerPlus, MODGEN, Monet®, MPCIExp, Mslab, Msview, MS Analyzer, MS Architect, MS-Express, MSIMON, Nanokernal®, NetCheck, NETED, Nucleus®, Nucleus All You Need®, 0-In®, OpenDoor®, Opsim, OutNet, P&RIntegrator, PACKAGE, PADS®, PARADE, ParallelRoute-Autocells, ParallelRoute-MicroRoute, Part Foundry, Parts SpecialList, PathLink, PCB-Gen, PCB-Generator, PCB IGES, PCB Mechanical Interface, PDLsim, PE-GMAC, PE-MAC, Personal Learning Program, Physical Cable, Physical Test Manager:SITE, PLA Lcompiler, Platform Express, PX, PLDSynthesis, PLDSynthesis II, Power Analyst, Power Analyst Station, PowerLogic, PowerPCB®, Precision®, Pre-Silicon, ProjectXpert, ProtoBoard, ProtoView, QDS, QNet, QualityIBIS, Questa, Questa Codelink, QuickCheck, QuickFault, QuickConnect, QuickGrade, QuickHDL, QuickHDL Express, QuickHDL Pro, QuickPart Builder, QuickPart Tables, QuickParts, QuickPath, QuickSim, QuickStart, QuickUse, QuickUse Development System, QuickVHDL, Quiet, Quiet Expert, RAM Lcompiler, RC-Delay, RC-Reduction, RapidExpert, REAL Time Solutions!, Registrar, Reliability Advisor, Reliability Manager, REMEDI, Renoir®, RF Architect, RF Gateway, RISE, ROM Lcompiler, RTL X-Press, Satellite PCB Station, Scalable Verification, ScaleableModels, SCAP, Scan-Sequential, Scepter, Scepter DFF, Schematic View Compiler SVC, Schemgen, SDF (Software Data Formatter), SDL2000 Lcompiler, Seamless®, Seamless ASAP, Seamless C-Bridge, Selective Promotion, Sheet Planner, Signal Spy, Signal Vision, SignaMask OPC, Signature Synthesis, Simulation Manager, SimPilot, SimView, Smartgrid, SmartMask, SmartParts, SmartRouter, SmartScripts, Smartshape, SNX, SneakPath, Analyzer, Spectra®, SpeedGate, SpeedGate DSV, SpeedWave, SpeedSelect, SOS Initiative, Source Explorer, SpiceNet, SST Velocity®, Standard Power Model Format (SPMF), StorSelect, Structure Recovery, Super C, Supermax® ECAD, Super IC Station, Supermax ECAD®, Symbol Genie, Symbolscript, SymGen, SYMED, SynthesisWizard, System Architect, System Design Station, System Modeling Blocks, Systems on Board Initiative, SystemVision, Target Manager, Tau®, TeamPCB, TeraCell, TeraPlace, TeraPlace-GF, TechNotes, TestKompress®, Test Station®, Test Structure Builder, The Ultimate Site For HDL Simulation, The Ultimate Tool For HDL Simulation, TimeCloser, Timing Builder, Time-it, TNX, ToolBuilder, Transcable®, TransDesign, Truetiming, Utopia, Vlog, V-Express, V-Net, VHDLnet, VHDLwrite, Verinex, ViewBase®, ViewDraw®, ViewCreator, ViewLogic®, ViewSim®, ViewWare®, Viking, Virtual Library, VirtuaLogic, Virtual Target, Virtual Test Manager:TOP, VirtualWires, Voyager®, VRTX®, VRTXmc, VRTXoc, VRTXsa, VRTX32®, VStation, VStation-30M, Waveform DataPort, We Make TMN Easy, Wiz-o-matic, WorkXpert, xCalibre®, xCalibrate, Xconfig, XlibCreator, XML2AXEL, Xpert, Xpert API, XperBuilder, Xpert Dialogs, Xpert Profiler, XRAY®, XRAY MasterWorks®, XSH®, Xtrace®, Xtrace Daemon, Xtrace Protocol, Xtreme Design Client, Xtreme Design Session, XtremePCB, XTK, Yield Assist, Zeelan®, Zero Tolerance Verification and ZLibs are trademarks or registered trademarks of Mentor Graphics Corporation or its affiliated companies in the United States and other countries.

**The following are service marks of Mentor Graphics Corporation:**

A World of Learning, ADAPT, AppNotes, Assess2000, BIST Compiler, BIST-In-Place, BIST-Ready, Concurrent Board Process, DirectConnect, ECO Immunity, EDGE (Engineering Design Guide for Excellence), Expert2000, FastTrack Consulting, IntraStep, ISD Creation, It's More than Just Tools, Knowledge Center, Knowledge-Sourcing, Mentor Graphics Support CD, Mentor Graphics SupportBulletin, Mentor Graphics SupportCenter, Mentor Graphics SupportFax, Mentor Graphics SupportNet-Email, Mentor Graphics SupportNet-FTP, Mentor Graphics SupportNet-Telnet, Mentor Graphics We Mean Business, MTPI, Online Knowledge Center, Reinstatement 2000, SiteLine2000, SupportNet KnowledgeBase, Support Services BaseLine, Support Services ClassLine, Support Services Latitudes, Support Services OpenLine, Support Services PrivateLine, Support Services SiteLine, Support Services TechLine, Support Services RemoteLine, and VR-Process.

Mentor Graphics' trademarks may only be used with express written permission from Mentor Graphics. Fair use of Mentor Graphics' trademarks in advertising and promotion of Mentor Graphics products requires proper acknowledgement.

Mentor sometimes refers to products by using trademarks ("Marks") owned by a third-party and such use is not an attempt to indicate Mentor Graphics as a source of that product, but is intended to indicate a product from, or associated with, a respective third party. Use of third-party Marks is intended to inure to the benefit of the respective third-party. Some of these marks are listed below:

PCI Express

Rev. 050412



# Table of Contents

<b>Lab 1: Introduction to Scripting .....</b>	<b>2</b>
<b>Lab 2: Basic Scripting .....</b>	<b>4</b>
<b>Lab 3: Environment/GUI Customizations.....</b>	<b>6</b>
Lab 3.1.....	6
Lab 3.2.....	8
Lab 3.3.....	10
<b>Lab 4: Object Programming.....</b>	<b>15</b>
<b>Lab 5: IDE .....</b>	<b>20</b>
<b>Lab 6: Output Engines .....</b>	<b>44</b>
<b>Lab 7: Expedition Layout .....</b>	<b>50</b>
Lab 7A: Reporting Using the Expedition Data Model.....	50
Lab 7B: Changing a Layout Using the Expedition Data Model.....	52
<b>Appendix A: Student Challenge.....</b>	<b>55</b>

# Lab 1: Introduction to Scripting

This lab has been set up to demonstrate some basic functionality which can be accomplished through scripting.

1. Open Expedition.
2. Use the **File >Open** menu to open the document *C:\Scripting\_Labs\Vidar\_WG\pcb\vidar.pcb*.
3. Select the Expedition menu **Help > Contents > Automation** to open the help on scripting. You will be using this, as well as Microsoft VBScript help, as you develop your own scripts, both here in the classroom and in your home environment. You may want to come back to this script to get some ideas as you progress through the course.
4. In the Expedition Keyin command entry box, enter the command:

```
run c:\scripting_labs\lab1\lab1_example.vbs
```

5. Follow the directions in the dialog boxes, keeping in mind that there will times when you will want to answer **No** or **Cancel**, or look at the status bar for some directions.
6. Write down some of your impressions about what can be done with scripting:

---

---

---

---

---

---

---

---

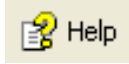

---



## Lab 2: Basic Scripting

This lab is intended to be very simple and easy just to give you an understanding of program controls. If you are already familiar with VBScript, this will be a quick review. If not, it will be a primer. The lab does not use Expedition. Expedition objects are discussed later in the course.

This lab focuses on basic programmatic controls. The intention is to introduce the usage of these controls, subroutines, and functions. You will accomplish this goal by running the script, adding comments to the script, and giving a basic description of what the script does. Every statement needs to have a comment added. You will find all the information necessary

1. Copy the file `c:\scripting_labs\lab2\lab2_template.vbs` to `lab2.vbs`.
2. Open `lab2.vbs` in VbsEdit.
3. In the VbsEdit window, select the **Help** icon  to open the Microsoft Windows Scripting Technologies help and the Help menu for help with the VbsEdit script debugger. Search the Windows Scripting Technologies help window for help regarding the scripting statements.
4. Run the script by selecting the **Start** icon .
5. Add a comment to the script for every statement. You may write comments which describe the net affect of multiple statements, as long as they include a description of what the individual statements do. (For example, “The following statements check to see if the strings entered are numeric by checking each character using....”)
6. Add titles to the message boxes and input box.
7. Given the comments you have entered into the script, and the operation of the script, write a description of the functionality of the script which includes the subroutine and the function:



## Lab 3: Environment/GUI Customizations

Lab 3 is broken up into three parts. Example code is given for each step as necessary. Use this code to complete the lab sections. If you finish early, try changing and adding your own code to better understand the controls.

### Lab 3.1

In this lab you will create a *scripts.ini* file and a startup script that will configure some keybindings for Expedition. You can name your startup script whatever you want. However, the name for your startup script and the name defined in the *scripts.ini* file must be the same.

In this lab you will add your own keybindings to your startup script. In the other sections of Lab 3 you will write more code to add menu and toolbar commands to this startup script.

1. Create a file in the local WDIR directory *C:\Scripting\_labs\local\_config* called *scripts.ini*, using either Notepad or WordPad. Add a section header for startup scripts which run whenever Expedition PCB documents are opened.

```
[Expedition PCB - Document]
```

Under this header add the name of the startup script that you are going to write. You can name it what ever you like, but in this Lab we will call the script *my\_startup.vbs*.

```
Script#0=my_startup.vbs
```

2. Save the *scripts.ini* file.
3. Next, copy the file *C:\scripting\_labs\Lab3\starter\_template\_adding\_commands.vbs* to *C:\Scripting\_labs\local\_config* directory, and give it the name of the script you entered in the *scripts.ini* file, *my\_startup.vbs*.

- 
4. Now, open the *my\_startup.vbs* file in a VbsEdit window. Notice that there are a couple of functions already defined in this file. Do not delete them. They will be used later.
  5. As shown in class, add a line of code that adds the type library MGCSDD.KeyBindings. An example of this code was shown in the class presentation, as well as below.

```
Scripting.AddTypeLibrary("MGCSDD.KeyBindings")
```

6. Next create a variable (keyBindTables) and write code that assigns the BindingsTable collection object to that variable. Use the Bindings property of the GUI object to do this. An example of this code was shown in the class presentation, as well as below.

```
Dim keyBindTables  
Set keyBindTables = Gui.Bindings("Document")
```

7. Next add at least one keybinding. You may use the example ones shown in the class material. Another example is shown below:

```
keyBindTables.AddKeyBinding "Alt+Z", "za", BindCommand,  
BindAccelerator
```

The **za** command is a standard keyin command in Expedition called **Zoom All**. It works exactly like the **Fit All** command.

8. Save the file.
9. Open Expedition and test your keybinding.
10. Using the Help facility within Expedition, identify another keyin command and add it to your script.
11. If you finish early, you can add as many keybindings as you like.
12. Close Expedition.
13. Wait for the next lecture.

## Lab 3.2

In this lab you will add code to the script, *my\_startup.vbs*, you created in Lab 3.1. You will add menus and submenus to the menu bar and associate commands with the menu items.

The order of the following statements must be maintained for the script to work.

1. First, create a variable (`docMenuBar`) and write code that assigns the `CommandBars` collection object to that variable. Use the `CommandBars` property of the GUI object to do this.

```
Dim docMenuBar
Set docMenuBar = Gui.CommandBars("Document Menu Bar")
```

2. Next, add a pulldown menu to the controls collection of the Document Menu Bar (`docMenuBar`). Use the `Add` method of the `docMenuBar.Controls` collection and assign the new menu object to a variable (`myMenu`). Use the `Caption` property of the menu object (`myMenu`) to give a name to your menu.

```
Dim myMenu
Set myMenu = docMenuBar.Controls.Add(cmdControlPopup, , , -1)
myMenu.Caption = "My Menu"
```

3. You must, now, get the `Controls` collection object from your menu object variable (`myMenu`) and assign it to a variable (`myCtrls`). Use the `Controls` property of your pulldown menu object (`myMenu`) to do this.

```
Dim myCtrls
Set myCtrls = myMenu.Controls
```

4. The `Controls` collection (`myCtrls`) is empty since you have not added any commands to your pulldown menu. Use the `Add` method of the `Controls` collection (`myCtrls`) and add a command to your menu. Assign the command object to a variable (`cmd`).

```
Dim cmd
Set cmd = myCtrls.Add
```

- 
5. Now that you have added a command, you must now define what text will appear in the new command in your pulldown menu (Caption property) and what that command will do using either the OnAction or ExecuteMethod property. For this command use the OnAction property to assign the command.

```
cmd.Caption = "Zoom All"  
cmd.OnAction = "za"
```

6. Now add a “Separator” to your pulldown menu. Do this by using the Add method of the controls collection (myCtrls) and using the enumerator “cmdControlButtonSeparator”.

```
myCtrls.Add(cmdControlButtonSeparator)
```

Now, we are going to add a submenu to your pulldown menu. Using the Controls collection (myCtrls), use the Add method to add the submenu and assign it to a variable (mySubMenu). Use the Caption property of the submenu object (mySubMenu) to give a name to your submenu.

```
Dim mySubMenu  
Set mySubMenu = myCtrls.Add(cmdControlPopup, , , -1)  
mySubMenu.Caption = "Sub Menu"
```

Next, we are going to add a command to your submenu that will execute the **Setup Parameters** command. Adding a command to your submenu is the same as adding a command to your pulldown menu (Steps 3, 4, and 5). This time, instead of using the pulldown menu controls collection (myMenu.Controls), you will use the submenu controls collection (mySubMenu.Controls).

```
Dim mySubMenuCtrls  
Set mySubMenuCtrls = mySubMenu.Controls  
  
Dim cmdSubMenu  
Set cmdSubMenu = mySubMenuCtrls.Add
```

Now that we have created the command, we need to define the caption and what the command will do. This time instead of using the OnAction property,

we will use the `ExecuteMethod` property to call a subroutine in the script. Since we are using the `ExecuteMethod` property, we must also use the `Target` property and assign it to the `ScriptEngine` enumerator. Since we are calling a subroutine, we must also tell the script not to exit.

```
Scripting.DontExit = True
cmdSubMenu.Caption = "Setup Parameters"
cmdSubMenu.ExecuteMethod = "setupParam"
cmdSubMenu.Target = ScriptEngine
```

7. Now, we need to add the subroutine that the `ExecuteMethod` calls named “`setupParam`”. This subroutine will use the GUI objects `ProcessCommand` method to run the **Setup Parameters** command. Notice the `(nID)` parameter in the subroutines definition. This is a special parameter used by the system. You will not use the `nID` parameter in your subroutine, but the subroutine will not execute if it is not defined.

```
Sub setupParam(nID)
    Gui.ProcessCommand "Setup->Setup Parameters", True
End Sub
```

8. Save the file.
9. Open Expedition and test your keybinding and menus.
10. Close Expedition.
11. Wait for the next lecture.

## Lab 3.3

In this lab you will add more code to the script you created in Lab 3.2. You will add a command to a toolbar in Expedition. We will add the **Zoom All** command to the **Standard** toolbar. If you wish, you can add your command to a different toolbar.

1. First, we use the GUI objects `CommandBars` collection object to get the **Standard** toolbar and assign it to a variable (`stdToolBar`).

---

```
Dim stdToolBar  
Set stdToolBar = Gui.CommandBars("Standard")
```

2. Next, we assign the standard toolbars Control collection to a variable (stdTBCtrls).

```
Dim stdTBCtrls  
Set stdTBCtrls = stdToolBar.Controls
```

3. Then we use the Add method of the standard toolbars control collection (stdTBCtrls) to add a button and assign the button to a variable (btn1).

```
Dim btn1  
Set btn1 = stdTBCtrls.Add(cmdControlButton,,, -1)
```

Next, we must define the properties for this button. Notice that we are using a bitmap property below. We have already placed the file in the directory. When you write a script, be sure that the file exists in the locations you specify.

```
btn1.OnAction = "za"  
btn1.ToolTipText = "View->Fit All"  
btn1.DescriptionText = "Fit all data to screen"  
btn1.BitmapFile =_  
"C:\scripting_labs\local_config\viewall.bmp"
```

4. Open and close the Vidar\_WG design a couple of times—leaving Expedition running—and take note of the changes you made to the toolbars and menus. They do not look quite right, do they?

## Fixing the Redundancy

The concept identified in step 4 is intentional to demonstrate a subtle, undesirable behavior which occurs when simply to the GUI menu and tool bars.

The following steps have you update your script so that, no matter how many times you open a design, you will only see your custom menu and toolbar commands once. Code will be added to the section of the code where you added your pulldown menu and toolbar command.

We will use the functions FindMenu and FindBtnByToolTip to try and find whether or not the menu and toolbar commands already exist. These two Functions were located in the template. The FindMenu function tries to find our pulldown menu we want to create. The FindBtnByToolTip function tries to find our toolbar command by finding its “tooltip” text.

1. First, let's fix our pulldown menu by adding the **FindMenu** command and using an “If ... Then” statement. Currently, in your script, the code to create your pulldown menu should look something like this:

```
Dim docMenuBar
Set docMenuBar = Gui.CommandBars("Document Menu Bar")

Dim myMenu
Set myMenu = docMenuBar.Controls.Add(cmdControlPopup,,, -1)
myMenu.Caption = "My Menu"

Dim myCtrls
Set myCtrls = myMenu.Controls
<create pull down menu code>
Scripting.DontExit = True
```

2. Instead of adding a pulldown menu object and assigning it to the myMenu variable, use the FindMenu function. Add the **bolded text**, as indicated below, to your script.

```
Dim docMenuBar
Set docMenuBar = Gui.CommandBars("Document Menu Bar")
Dim myMenu
Set myMenu = FindMenu("My Menu", docMenuBar)

If myMenu Is Nothing Then

    Set myMenu =
docMenuBar.Controls.Add(cmdControlPopup,,, -1)
    myMenu.Caption = "My Menu"

    <create pull down menu code>
    Scripting.DontExit = True
End If
```

- 
3. Use this same technique to update the code for our toolbar command. The following is the current toolbar command code:

```
Dim stdToolBar
Set stdToolBar = Gui.CommandBars("Standard")

Dim stdTBCtrls
Set stdTBCtrls = stdToolBar.Controls

Dim btn1
Set btn1 = stdTBCtrls.Add(cmdControlButton,,, -1)
btn1.OnAction = "za"
btn1.ToolTipText = "View->Fit All"
btn1.DescriptionText = "Fit all data to screen"
btn1.BitmapFile =
"C:\scripting_labs\local_config\viewall.bmp"
```

4. Add the FindBtnByToolTip function and an If statement. Add the **bolded text**, as indicated below, to your script.

```
Dim stdToolBar
Set stdToolBar = Gui.CommandBars("Standard")

Dim stdTBCtrls
Set stdTBCtrls = FindBtnByToolTip("View->Fit All",_
stdToolBar)

If stdTBCtrls Is Nothing Then
    Set stdTBCtrls = stdToolBar.Controls
    Dim btn1
    Set btn1 = stdTBCtrls.Add(cmdControlButton,,, -1)
    btn1.OnAction = "za"
    btn1.ToolTipText = "View->Fit All"
    btn1.DescriptionText = "Fit all data to screen"
    btn1.BitmapFile =_
        "C:\scripting_labs\local_config\viewall.bmp"
End If
```

5. Save the file.

6. You should now be able to open and close the Expedition design (leave the GUI open, and just close and open the document) as many times as you want, and your menu and toolbar commands will only appear once.

You may be asking, “Why is this important?” The fact is, you may find yourself working on multiple designs and want only to open and close the different layouts, not necessarily exit Expedition. These changes to the script allow you to do so.

7. Open Expedition and test your keybinding and menus.
8. Close Expedition.
9. Wait for the next lecture.

---

## Lab 4: Object Programming

The template script file has been created which contains the necessary statements to connect to an Expedition server and acquire a license to process your code. Both of these concepts are discussed in detail in the Expedition Layout module. Additionally, there are a few functions which will also be in the template to make the coding easier. A brief discussion regarding them is included in the directions below.

The focus of this lab is the understanding of objects and properties, some of the coding techniques which may be used to access them, and the passing of variables to subroutines.

### The Script

The script you are about write has a few key “flow” elements.

These are:

- Prompt the user for a log file to report script processing status and detailed net information
- Call a subroutine. The subroutine shall have one internal variable. The main program will call the subroutine multiple times and pass a value to the variable in the subroutine. There are two different variables in the main program which will supply the value via the Call statement. The first is simply a logical (Batch = True : Call NetTraceTotal(Batch)). The second will pass the netname to the same variable internal to the subroutine (Call NetTraceTotal(UserEnteredNetName)), and the subroutine needs to be able to choose the appropriate behavior.

The first call to the subroutine needs to start a “batch” mode process which writes a summary of each net to the data file. The summary will include the netname, the total length of the copper on each net, and number of vias on each net.

The logfile needs to be closed at the end of the “batch” mode of the script

and advise the user that it has been closed.

The user will next be prompted for a netname for interactive reporting of the detailed net data. Once the user closes the dialog box with the net information, the script will ask if they wish to process another net. If yes, the prompt dialog box needs to be displayed. This process needs to continue until the user declines.

This means that the subroutine contains selectable functionality. Thus, calls to the subroutine must be set up such that the value of the variable passed to the subroutine selects the desired functionality.

Remember to declare all your variables with the Dim statement and initialize any which require initialization.

1. Copy *C:\scripting\_labs\lab4\lab4\_template.vbs* to *lab4.vbs*.
2. Add the statement which requires the variables to be explicitly declared. (See the lecture notes.)

### Main Body of the Script

3. Looking at the class notes, prompt the user for a log filename, use the file system object to create a log file which is located in the project directory, and contains the following two lines, double-spaced, at the top of the file:

```
This is the log file opened for Lab 4.
```

```
It is located in C:\scripting_labs\vidar_wg\pcb\
```

4. Call a subroutine named `NetTraceTotal`, pass the value of a variable named `Batch` as an argument making sure that you first set `Batch` to `True` (Call `NetTraceTotal(Batch)`). You will be creating the subroutine later.
5. Add the code which will write a closing statement, for example, "Closing Logfile.", which is two lines below the last entry in the file; close the logfile, and inform the user that the logfile has been closed successfully.

- 
6. While a variable ContInput is True, continue to prompt the user to enter netnames from the design and call the NetTraceTotal script, this time, passing the net name the user entered in the prompt to the subroutine. (While/Wend, If/Then/Else/End If, MsgBox, vbYesNo, True, False, InputBox)
  7. After the message box report is dismissed, ask the user if they would like to view another net. If yes, prompt them for the next net name, if not, immediately stop prompting the user and issue a message box that the interactive reporting has been terminated at the user's request.

### Subroutine Script

8. Create a subroutine named NetTraceTotal which will accept NetName as an argument. (For example, pass a variable value from the main program to the subroutine.) Place your subroutine **after** the Private Validation Function found in the template. (Sub, End Sub)
  - a. Have the script decide if the batch or interactive mode should be run.

### Batch Mode

- b. Get a collection of all the nets.
- c. Lock the server and unlock the server at the appropriate points.
- d. For each net,
  - i. Determine the name of the net and assign it to a variable called PassedNetName.
  - ii. Create a NetVias collection and from that, determine the total number of vias on the net, and assign that value to a variable named ViaCount
  - iii. Create a collection of segments for the current net, determine the segment lengths, and then add them all up in a variable called TotalSegLength.

- iv. Create a variable named TotalInInches and convert the TotalSegLength, which is in ths of an inch, to inches. (Divide by 100.)
- v. Report the total length of each net and number of vias to a log file using the following method:

```
LogFile.Write ("Net Name is " & NetName & vbCrLf &_
    " Number of traces are " & SegCollCount & vbCrLf &_
    " Total number of vias are " & ViaCount & vbCrLf &_
    " Total trace length is " & TotalSegLength & "ths of an inch"& vbCrLf &_
    " Total trace length is " & TotalInInches & " inches" & vbCrLf)
```

- vi. Separate each net report with a blank line.

## Interactive Mode

You will notice that the batch and interactive modes are very similar, and most of the statements for the batch mode can just be copied and used as-is in the interactive mode.

- e. Create an object from the variable passed from the main program using the FindNet method.
- f. Create a NetVias collection and from that, determine the total number of vias on the net, and assign that value to a variable named ViaCount.
- g. Using the Name property, assign the net name to a variable called NetName.
- h. Create a collection of segments for the net, determine the segment lengths; then add them all up in a variable called TotalSegLength.
- i. Create a variable named TotalInInches and convert the TotalSegLength, which is in ths of an inch, to inches. (Divide by 100.)

- 
- j. Report the total length of each net and number of vias to a message box.  
Use the following statement with the addition of a message box title:

```
MsgBox "Net Name Entered " & NetName & vbCrLf &_
```

```
    "Number of traces is " & SegCollCount & vbCrLf &_
```

```
    "Total number of vias are " & ViaCount & vbCrLf &_
```

```
    "Total trace length is " & TotalSegLength & "ths of an inch"& vbCrLf &_
```

```
    "Total trace length is " & TotalInInches & " inches"
```

## Lab 5: IDE

In this lab you will create a simple cell preview script form window. The script form will contain three controls.

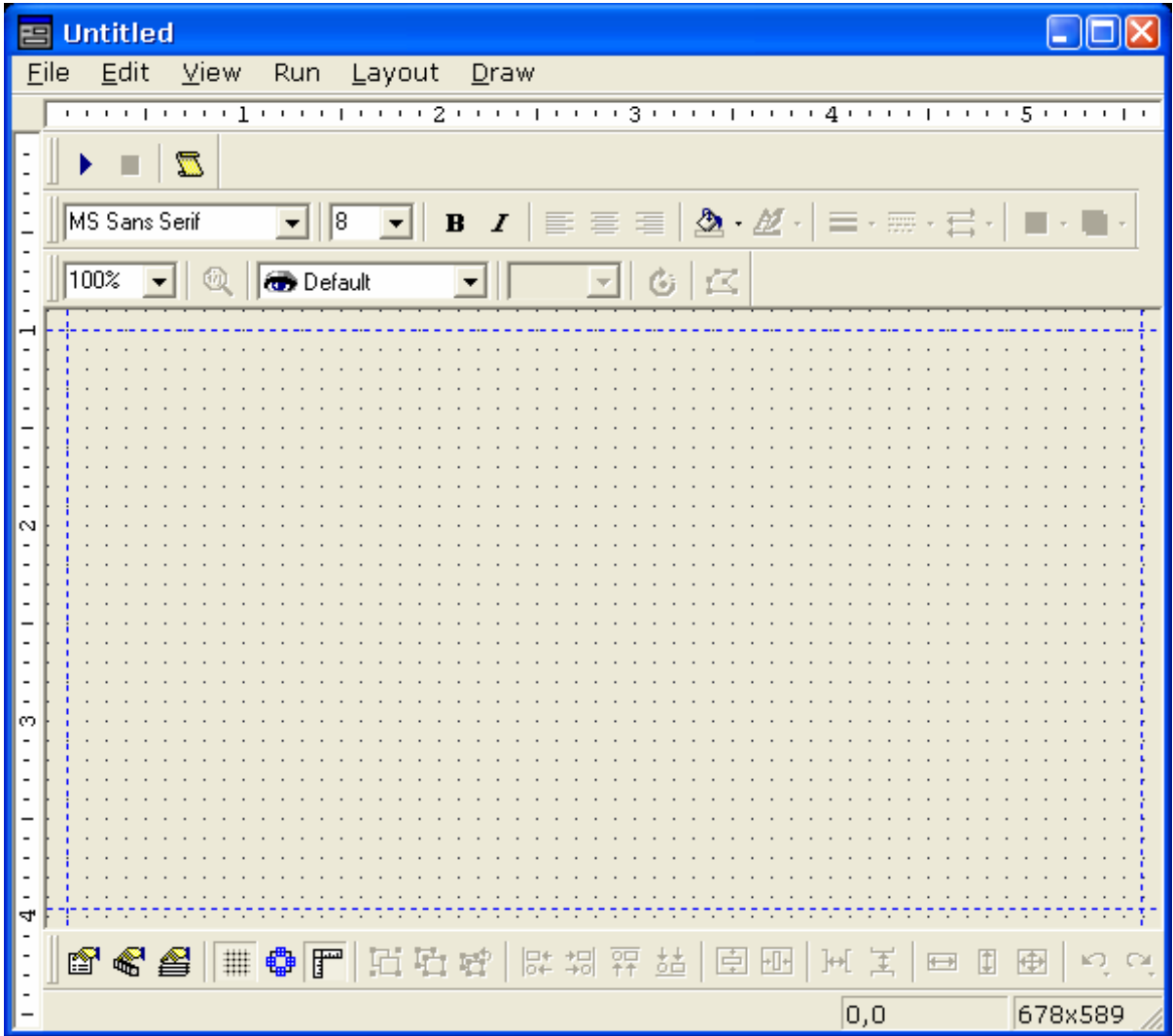
- The first control is a list box that will be loaded with the reference designators of the parts in the design.
- The second control will be a command button. Clicking the button will load the reference designators into the list box.
- The last control is the ActiveX control called “CelleZView”.

You will find some statements and concepts which will be covered in a later module. The Focus on this module is the creation of the Form itself, not the subroutine statements; although, they will be explained along the way.

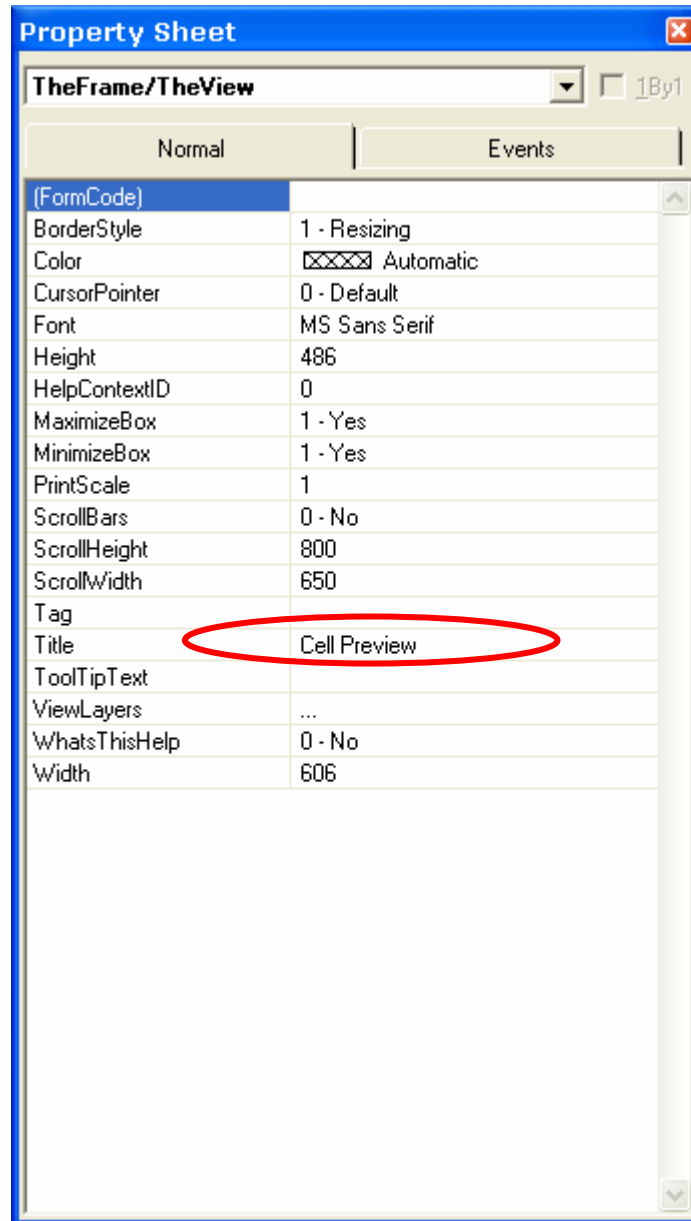
First, we will build the list box and the command button, and write the code to load the reference designators into the list box. We will then add the CelleZView ActiveX control. Lastly, we will add the code that will show the cell for any reference designator selected in the list box.

1. In the *C:\scripting\_labs\Lab5* directory copy *startup\_template.efm* to *CellPreview.efm*.

2. Open the form in Expedition, and use the control key while opening the form so the form does not run. You should see the following window:



- First, let's change the name of the form in the title area of the script form window from *untitled* to *Cell Previewer*. Right-click in the form area, and select **Form Properties** in the popup menu. The following properties window will appear:

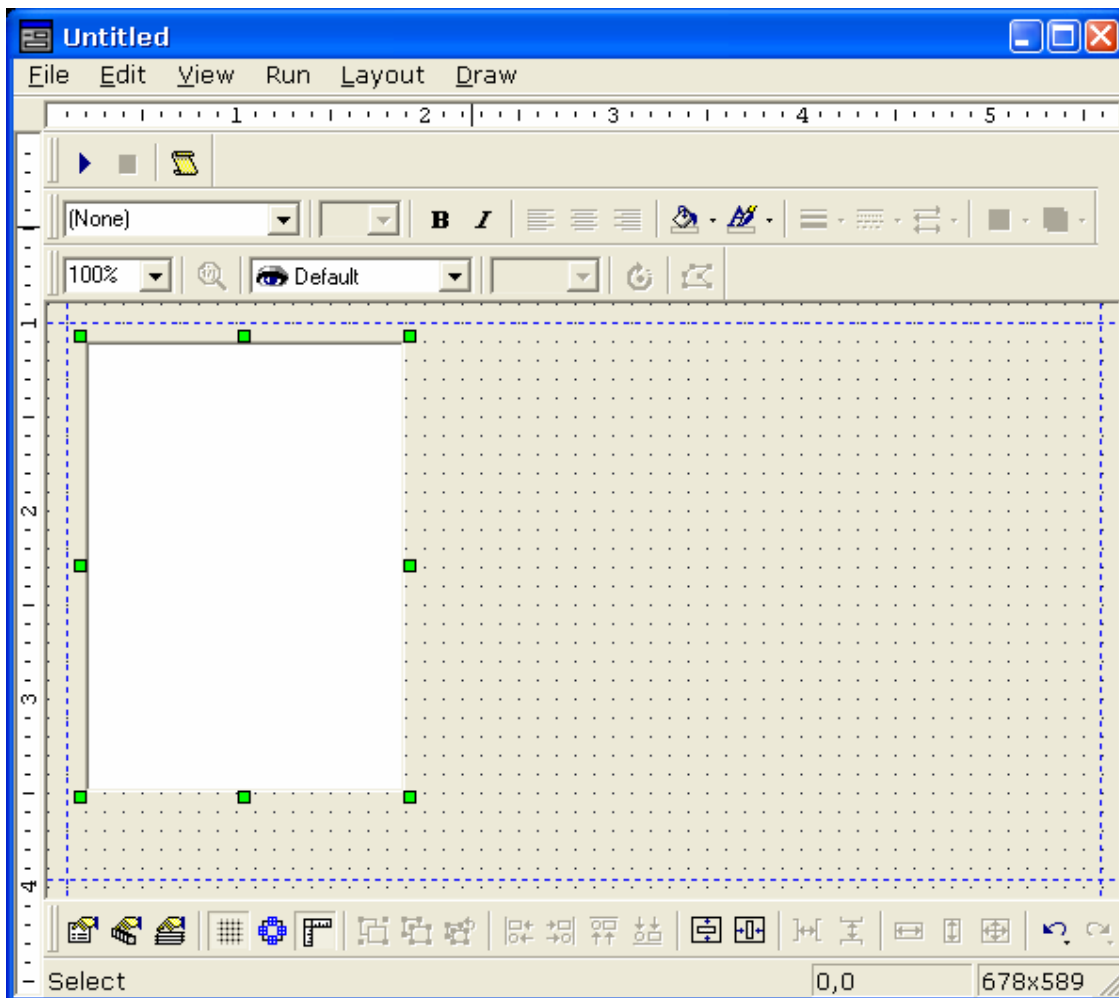


Change the Title property to *Cell Previewer*, and notice the title is now changed in the script form.

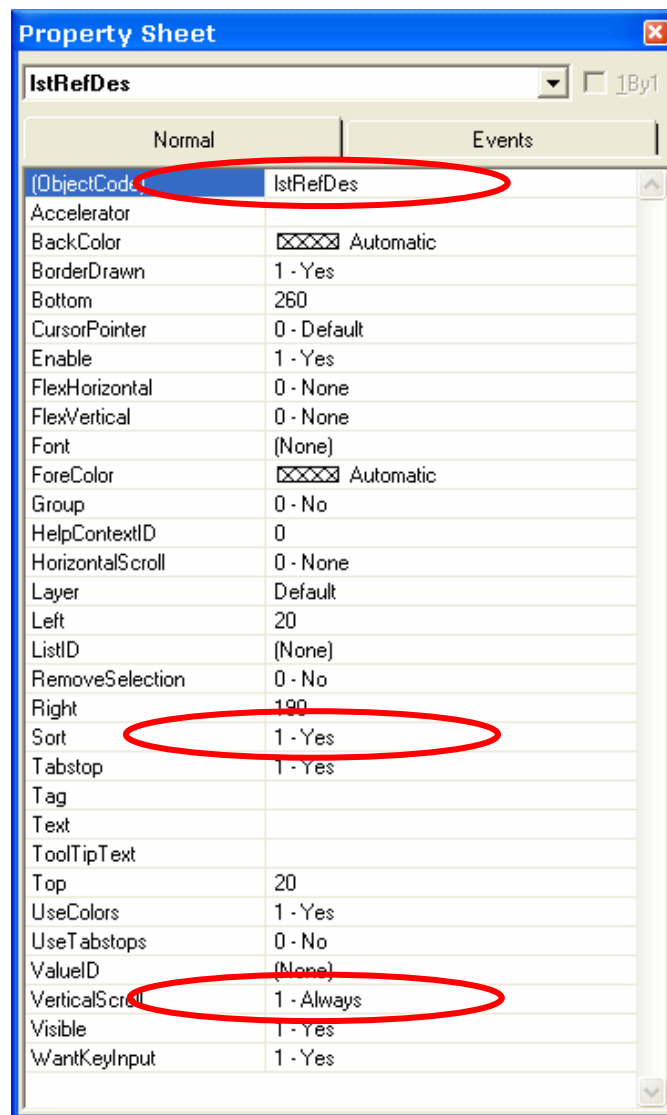
- 
4. Now, let's add a list box to the form that will hold a list of all the reference designators in the design. To do this click the **ListBox** button on the **Object Bar**, and then draw a rectangle on the form.



- When you are finished, the list box should appear like shown below. You can adjust the size of the list box by using the handles on the corners and side of the box when it is selected.



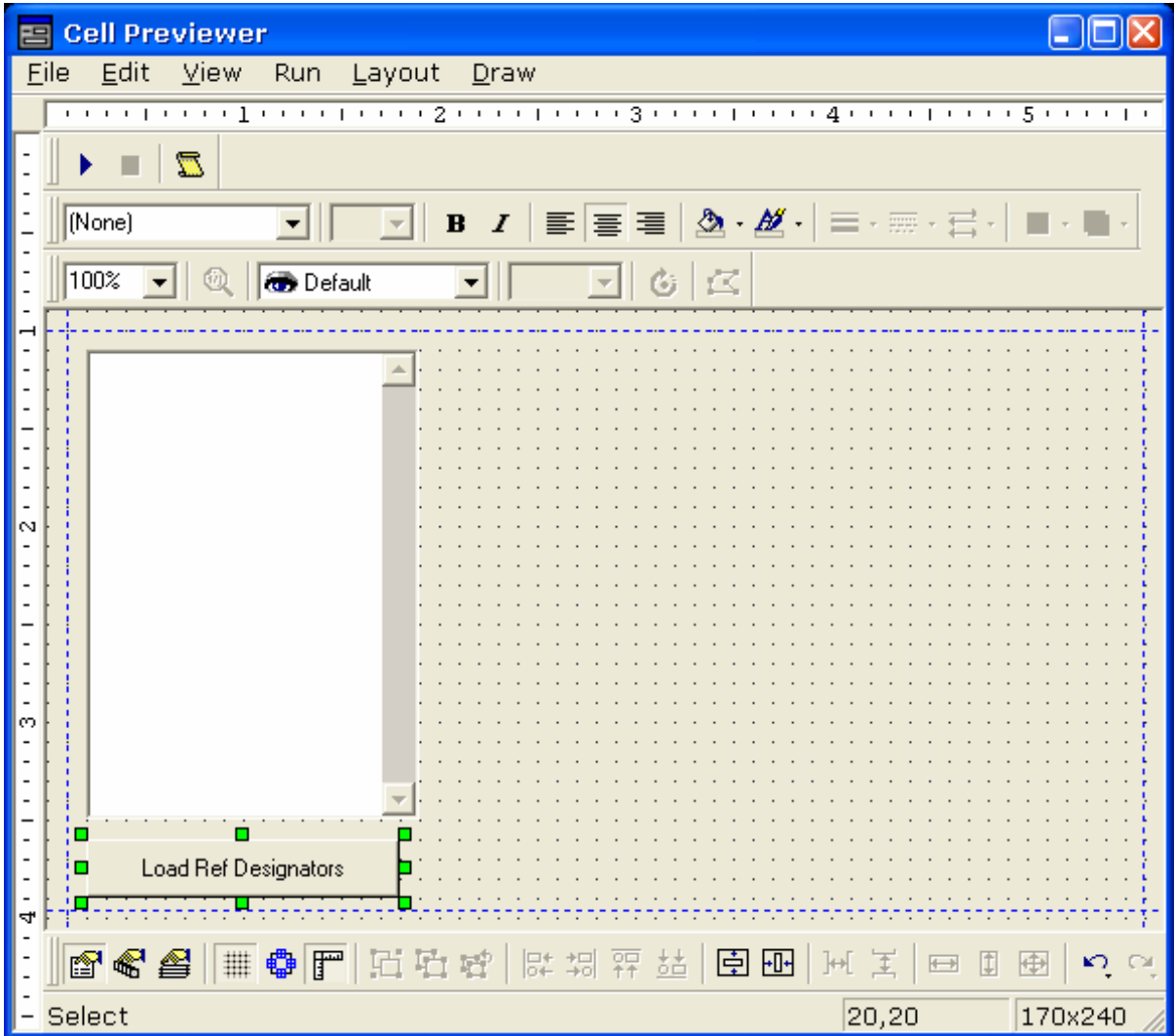
- Right-click on the listbox and open the properties dialog box as shown below. Set the (ObjectCode) property to *LstRefDes*. The (ObjectCode) property is just a fancy name for the “Name” of the list box object. Every control you add to the form will have to have a unique name or (ObjectCode) property.
- Set the Sort property to *Yes* to ensure the list box sorts any entries it receives. Set the VerticalScroll property to *Always* to ensure we get a vertical scroll bar.



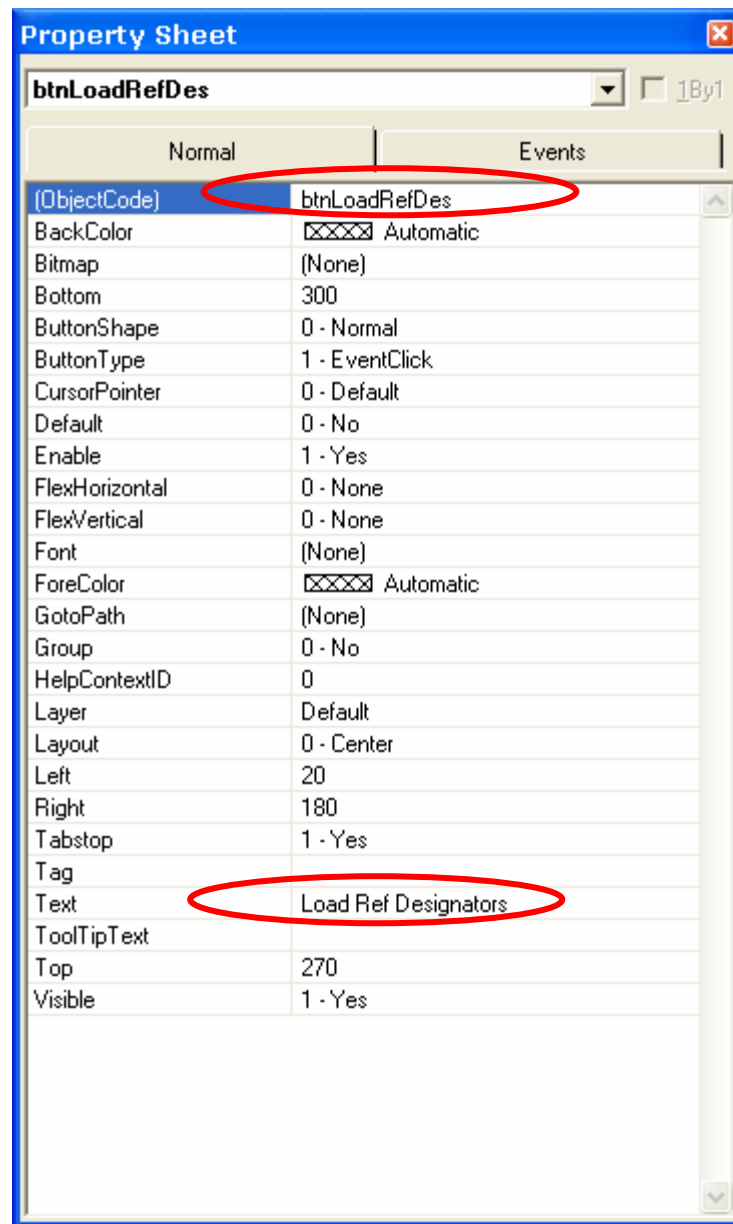
- Next, add a command button to the form right below the list box. Click the **Button** command and draw a rectangle below the list box.



- Adjust the size of the button as you did with the list box by using the handles on the button that are shown when the button is selected.



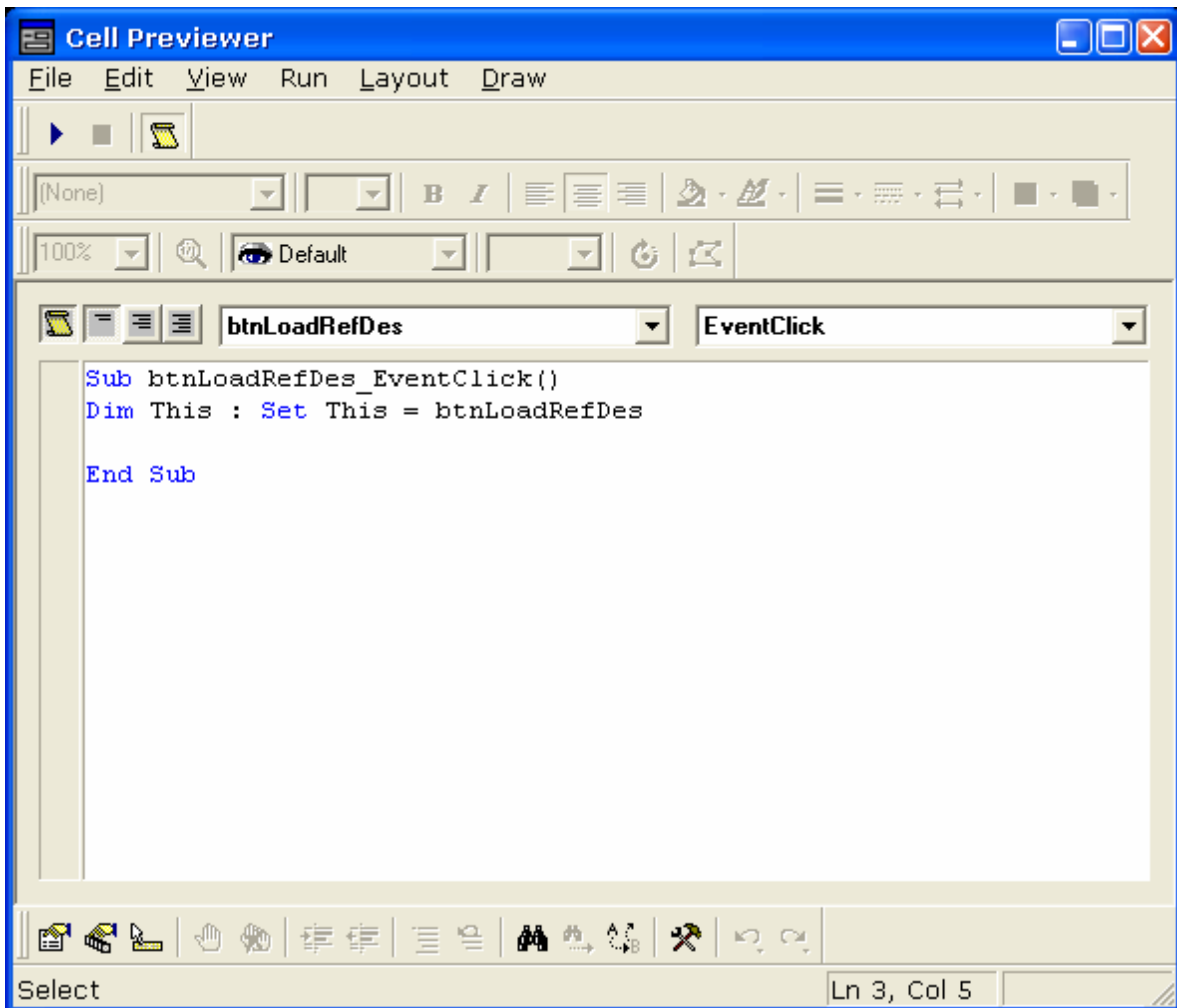
10. Right-click on the button, and open the properties dialog box. Set (ObjectCode) to *btnLoadRefDes*. Also change the Text property to *Load Ref Designators*.



11. Now, let's add some code so all the reference designators in the design will be loaded into the list box when you click the **Command** button. To do this we must add the code to the **btnLoadRefDes** button's event subroutine named "EventClick". There are a few ways to invoke the script editor to add

---

code to this event. The simplest way is to double-click the button. This will change the IDE from a form editor to the script editor. The window is shown below:



12. The code we need is pretty basic. The code needs to get a collection of all the components in the design and then add each reference designator to the list box. Since there could be a lot of components, we should also lock the server when trying to do this. We will need two variables—one for the components collection, and one to hold the component object for each component as we iterate through the collection.

```
Dim compObj  
Dim compsColl
```

```
Application.LockServer ' Lock Server
Set compsColl = pcbDoc.Components ' Component Collection
```

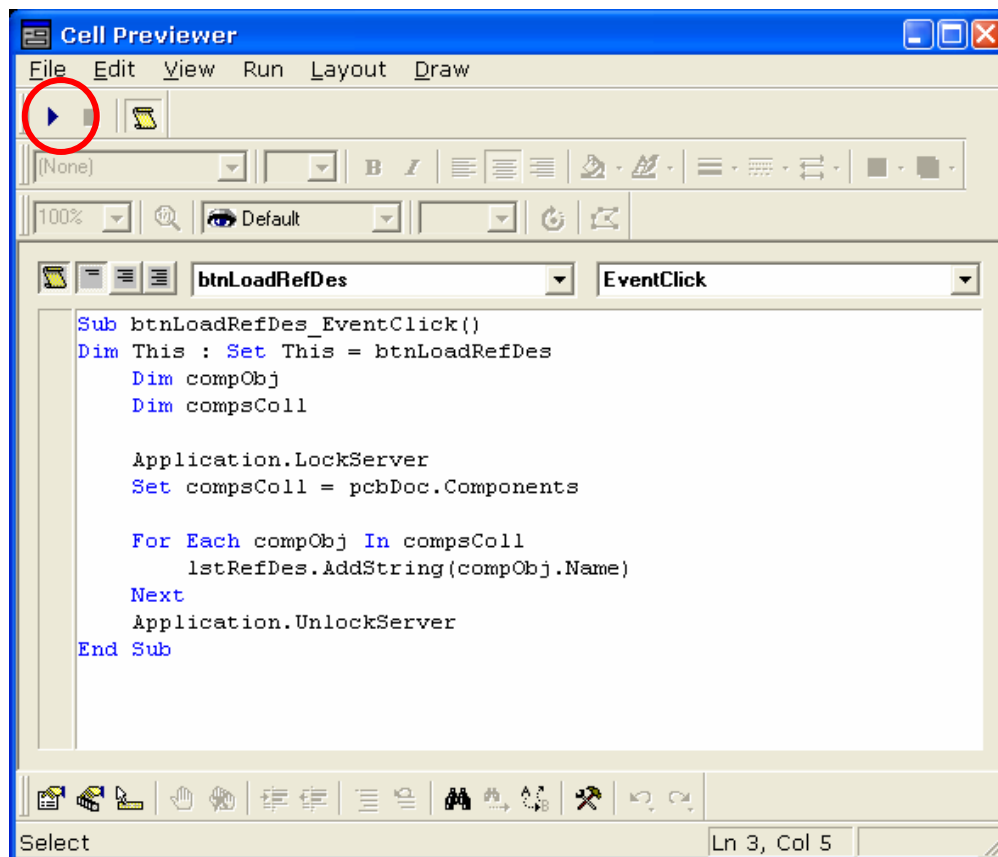
- 13.A “For Each” loop iterates through the collection. In the loop we call the LstRefDes list box’s AddString method to add the reference designator for each component.

```
For Each compObj In compsColl
    lstRefDes.AddString(compObj.Name)
Next
```

- 14.Then we unlock the server.

```
Application.UnlockServer
```

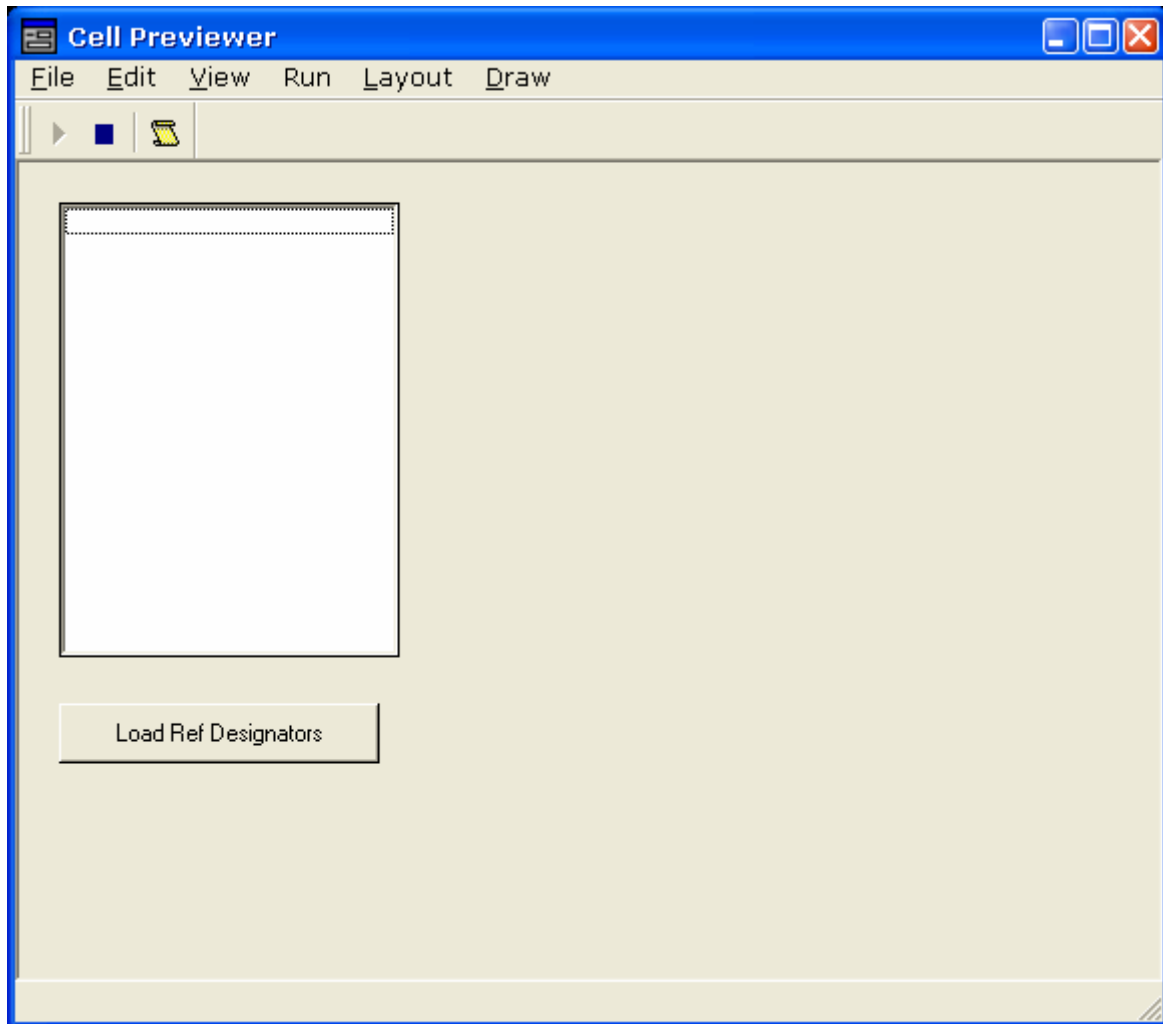
Your code should now look like this:



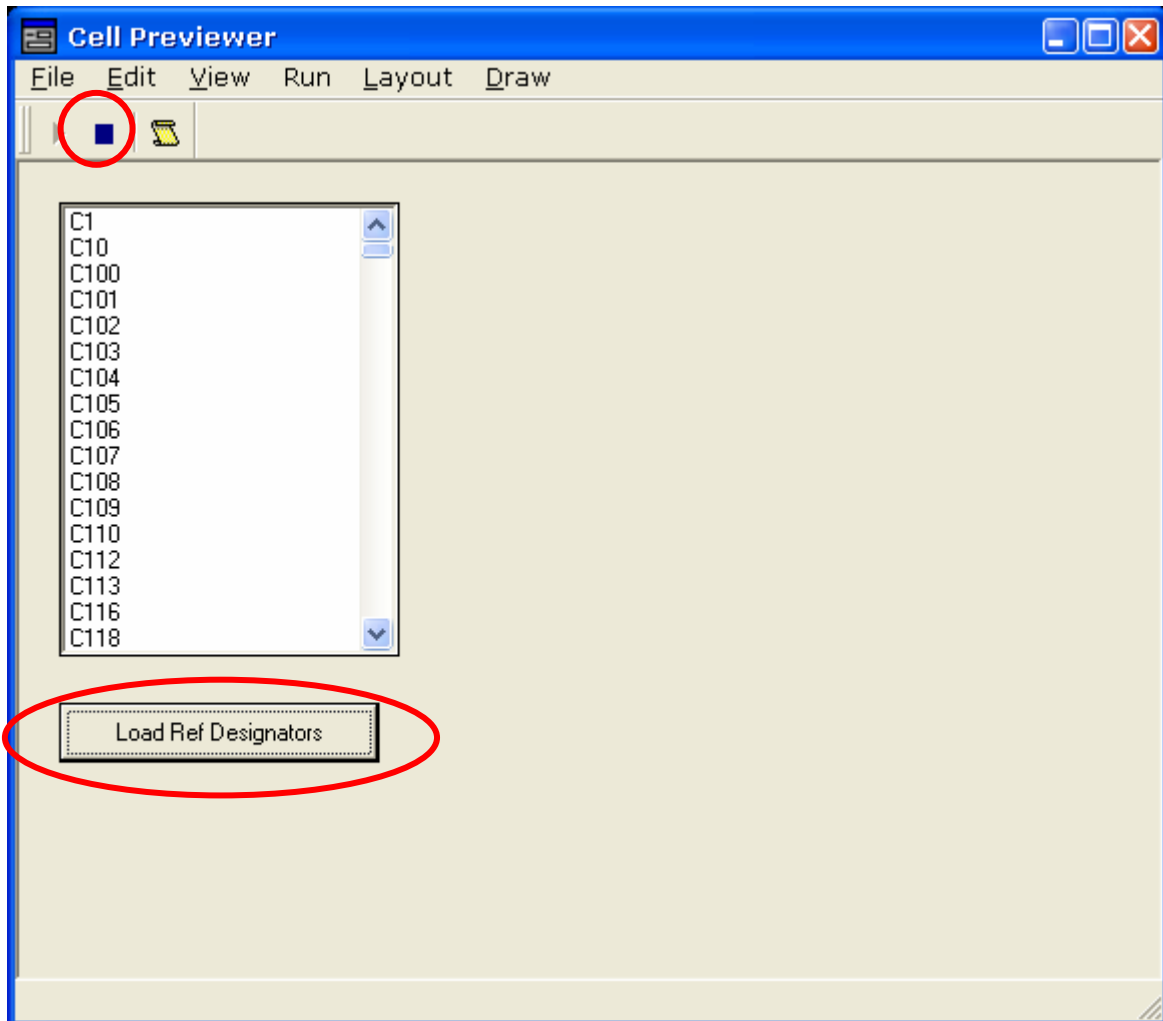
---

15. Invoke the script form by clicking the **Run Form** button (circled on previous page).

You should see the following window:



16.If you click the **Load Ref Designators** button, the code will execute, and the reference designators will be loaded into the list box as shown below:



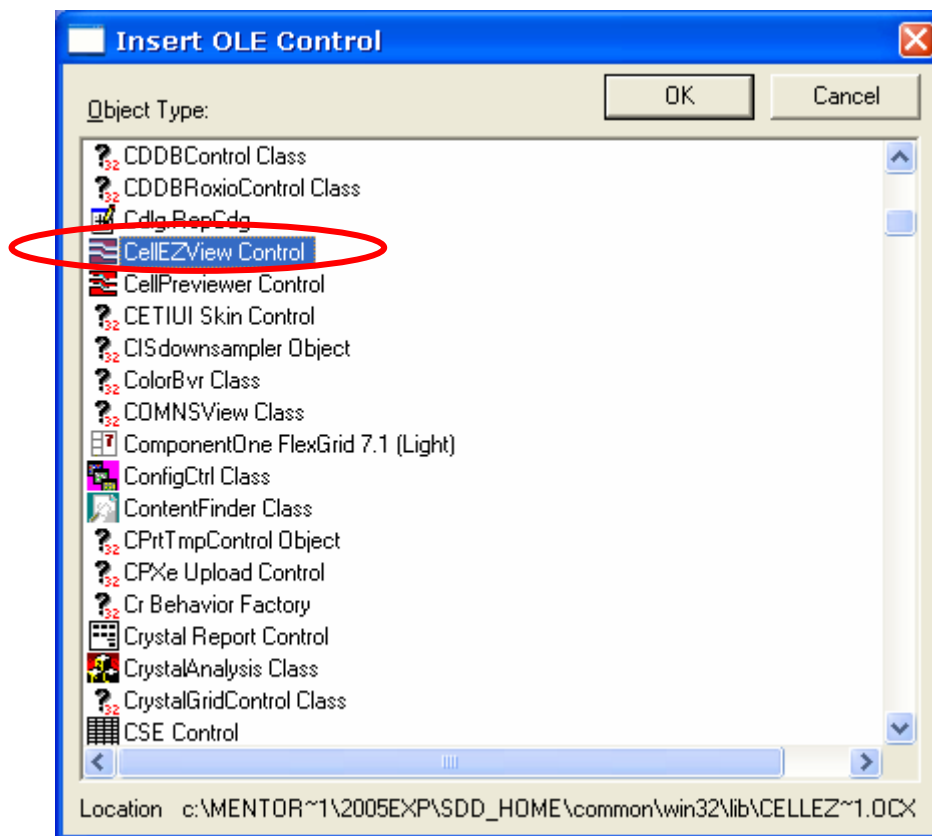
17.Click the **stop form** button circled above and continue the lab.

---

18. Now, let's add the cell preview ActiveX control to our form. Click the **ActiveX** button and draw a box on the form.

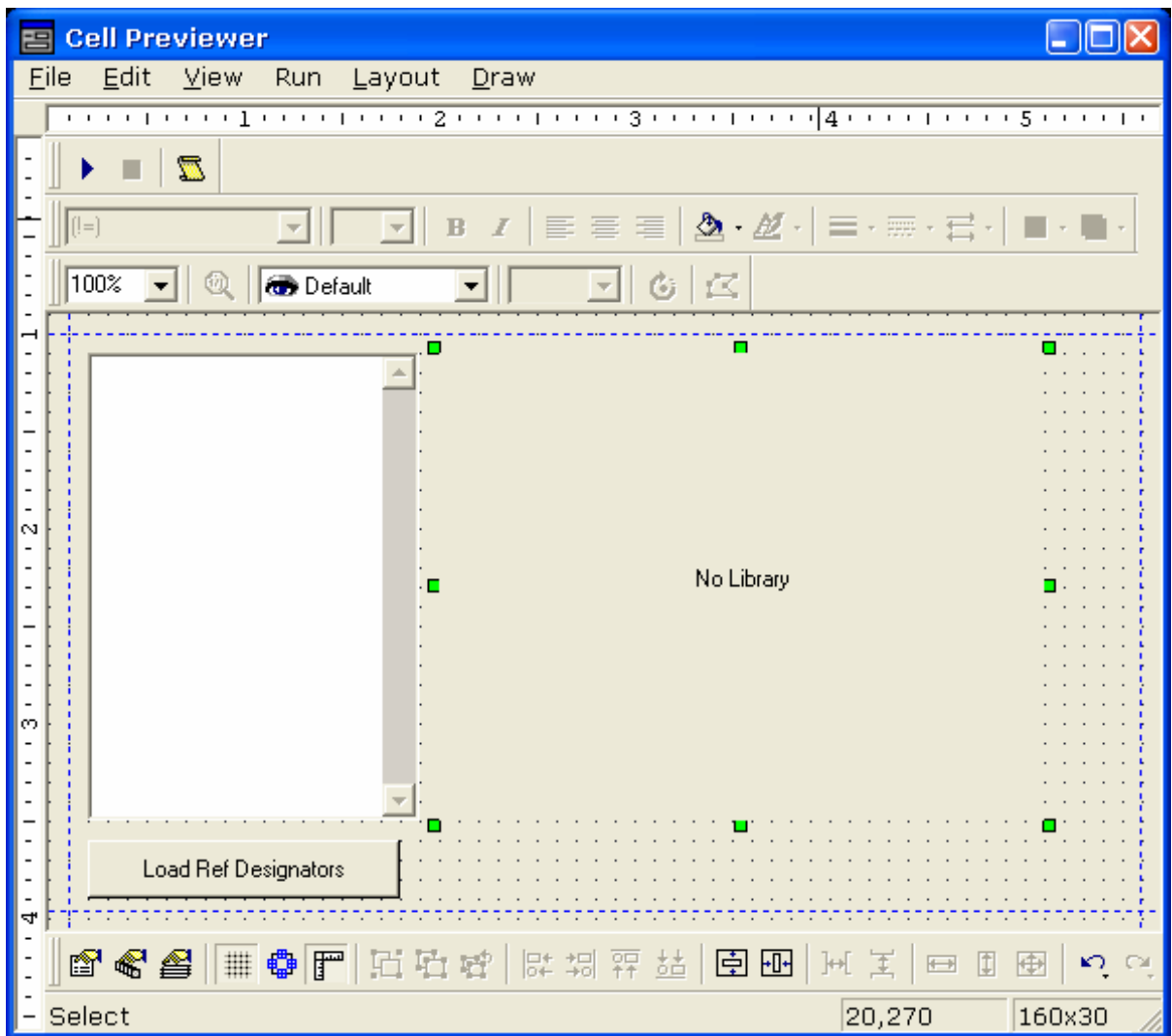


19. The following window will open after you draw the box:



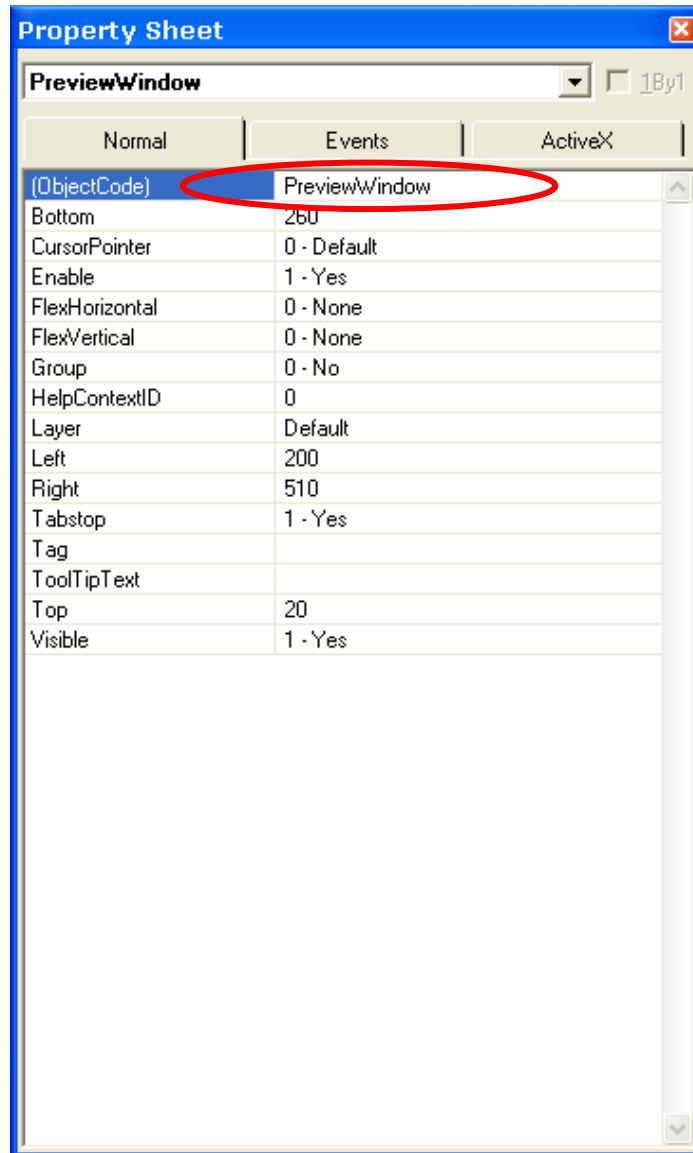
20. Find the CelleZView control in the list and click **OK**.

The form now contains the CelleZView control as shown below:

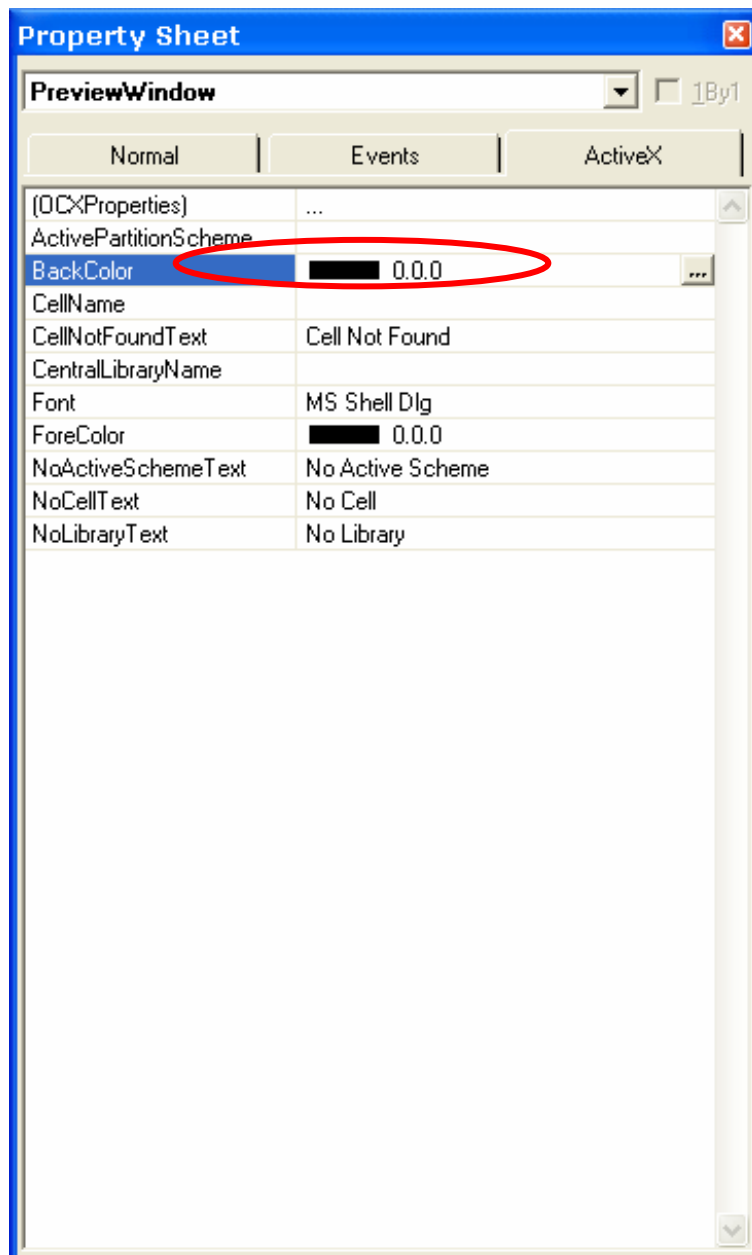


---

21. Notice that the background color is not black for this control. We need to right-click on the CelleZView control and open the properties window. First, change the name of the control to PreviewWindow.




22. Next, click the **ActiveX** tab, and change the BackColor property to black. Notice the properties CentralLibraryName and CellName. If you wanted to, you could select a library name and cell name in this window, and that cell would appear in the preview window. But we do not want to do that for this script.



---

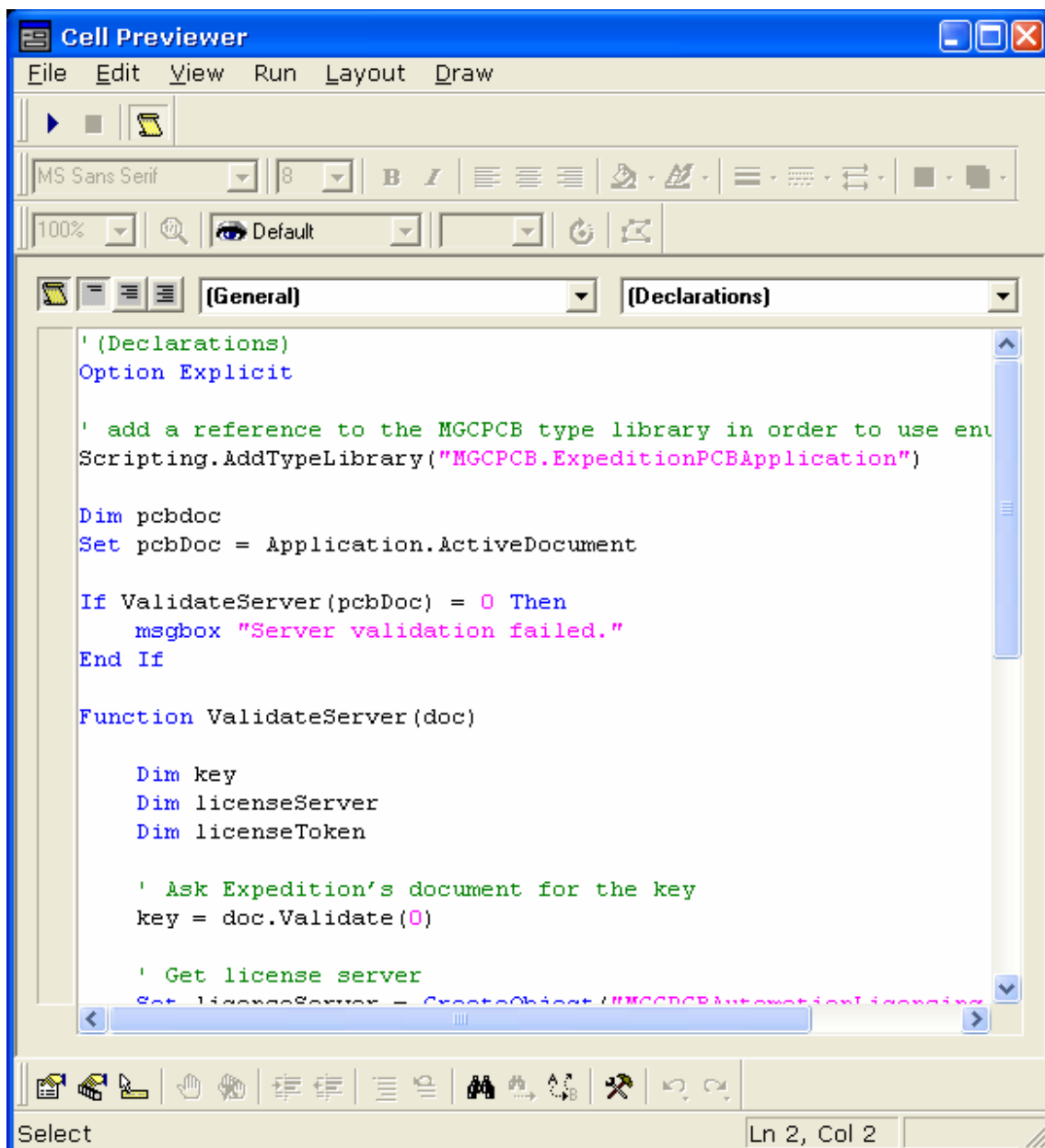
23. The next thing we are going to do is get the central library name the current design is referencing. We will write code that will assign the central library name property to the library this design is using. The next section explains how to do this.

Click the **Script** button to switch the view to the script editor. Make the script editor look like the window below on the next page. You may have to click on the dropdown list and select **(General)**.

 <p><b>Note</b></p>	The (General) (Declarations) section of the code is where you can define global variables and your own functions and subroutines.
--	---

Notice that the “pcbDoc” variable has been defined and is set with the applications active document object.

Also, the ValidateServer function has been included to ensure we check out a license for automation.



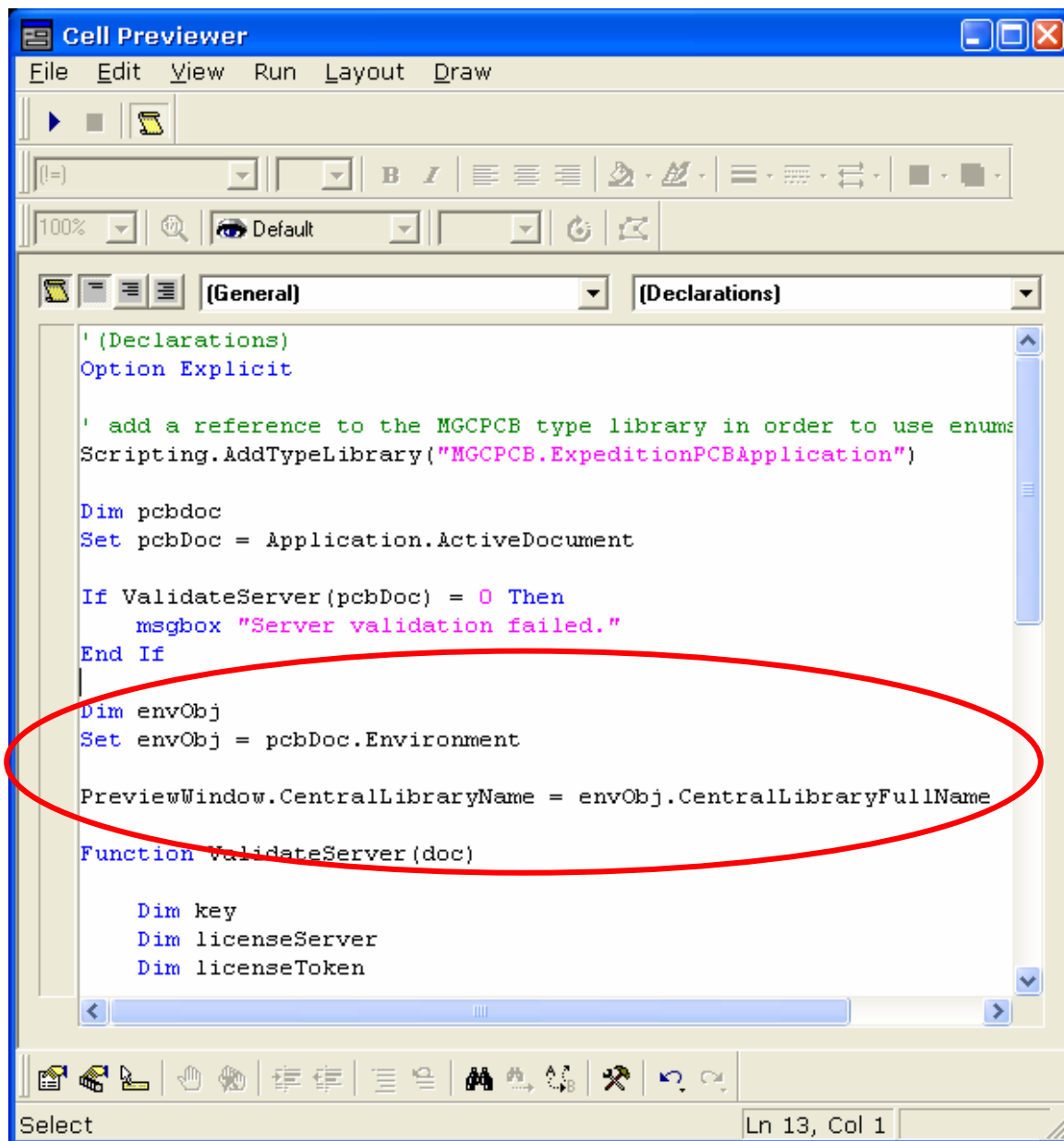
24. Now, we need to get the name of the central library used by this design. The name of the library is stored in the Environment Object of the active document. In the Environment object, the CentralLibraryFullName property stores the name of the central library. So we assign the CentralLibraryFullName property to the CelleZView controls property called CentralLibraryName.

---

The code will look like this:

```
Dim envObj
Set envObj = pcbDoc.Environment
PreviewWindow.CentralLibraryName = envObj.CentralLibraryFullName
```

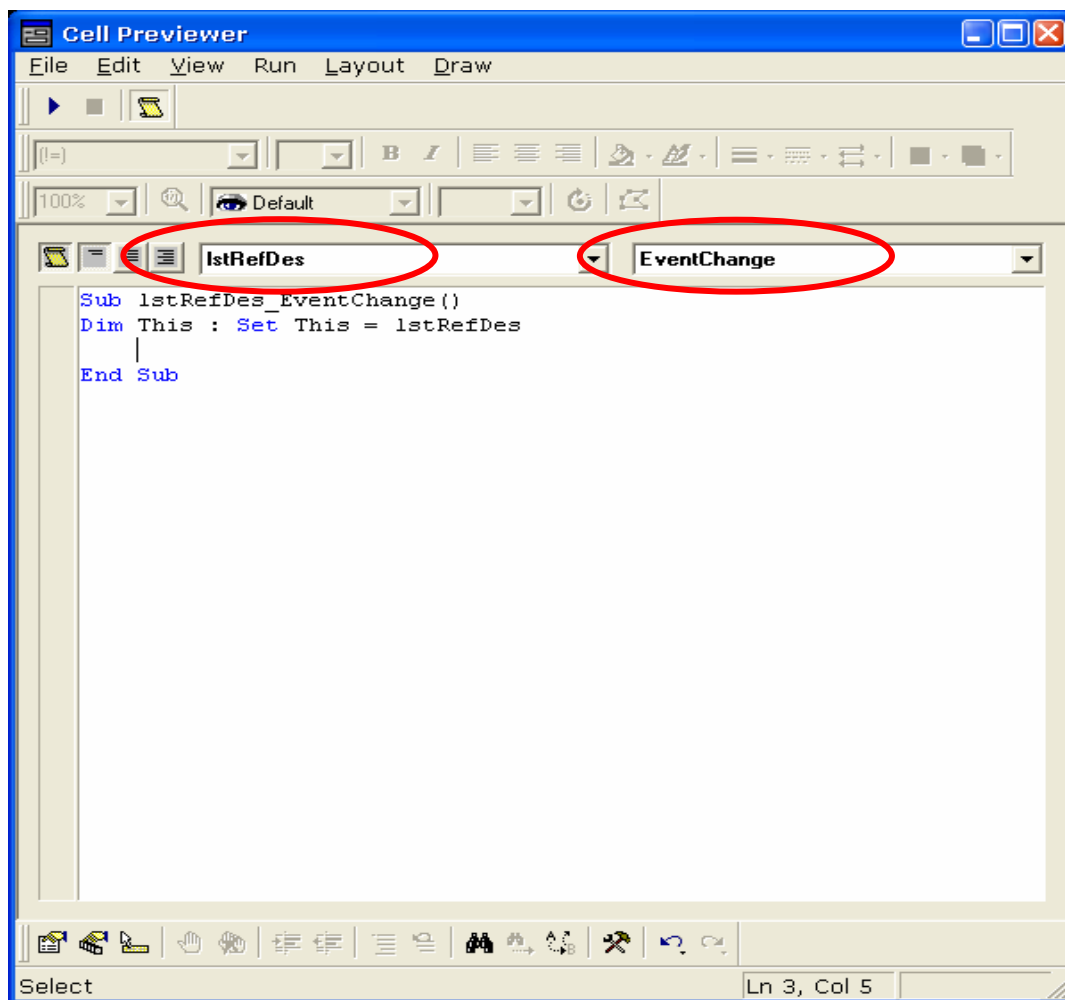
This code will be entered into the (General) (Declarations) sections before the ValidateServer function as shown below:



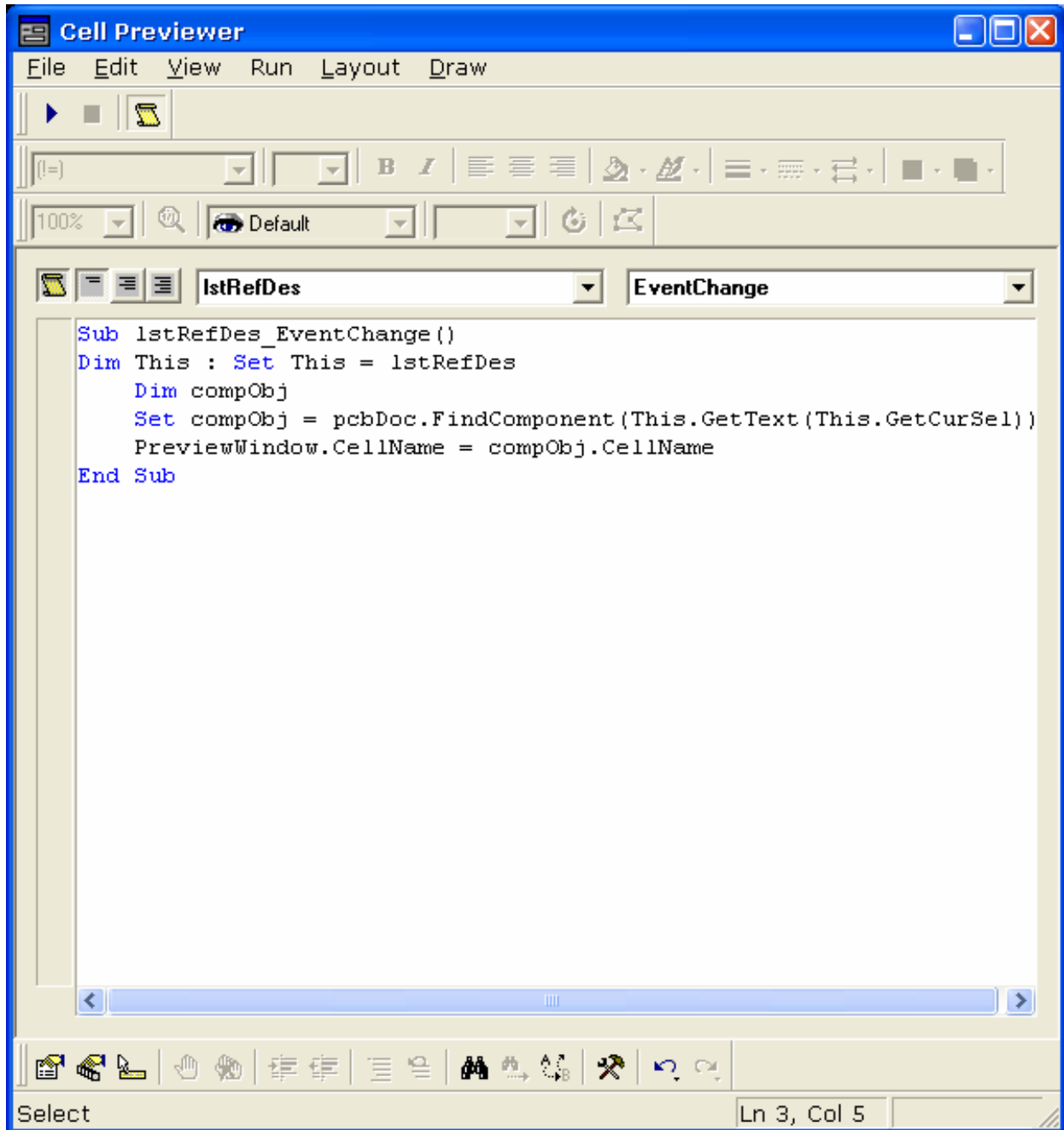
25. The last thing we need to do is write the code that will allow us to click on a reference designator in the list box, and show the cell in the cell preview window.

When you click on a reference designator in the list box, an event is triggered (or fired). The event is called “EventChange”. As shown earlier, the name of the list box is *lstRefDes*. So the subroutine *lstRefDes\_EventChange* is called when you click on a reference designator.

So, we must add our code to that event or subroutine. In order to add our code, we must get to that subroutine in the script editor. The easiest way to do that is by changing the object dropdown that currently says (General) to *lstRefDes*. Then change the event dropdown from (Declarations) to *EventChange*. This is shown on the following page:



26. After clicking on a reference designator, we must use it to find the component object in the design. Then we assign the CellName property for that component object to the CellName property of the cell preview window. The code will look like the following:



Let's look at the following line of code:

```
Set compObj =  
pcbDoc.FindComponent(This.GetText(This.GetCurSel))
```

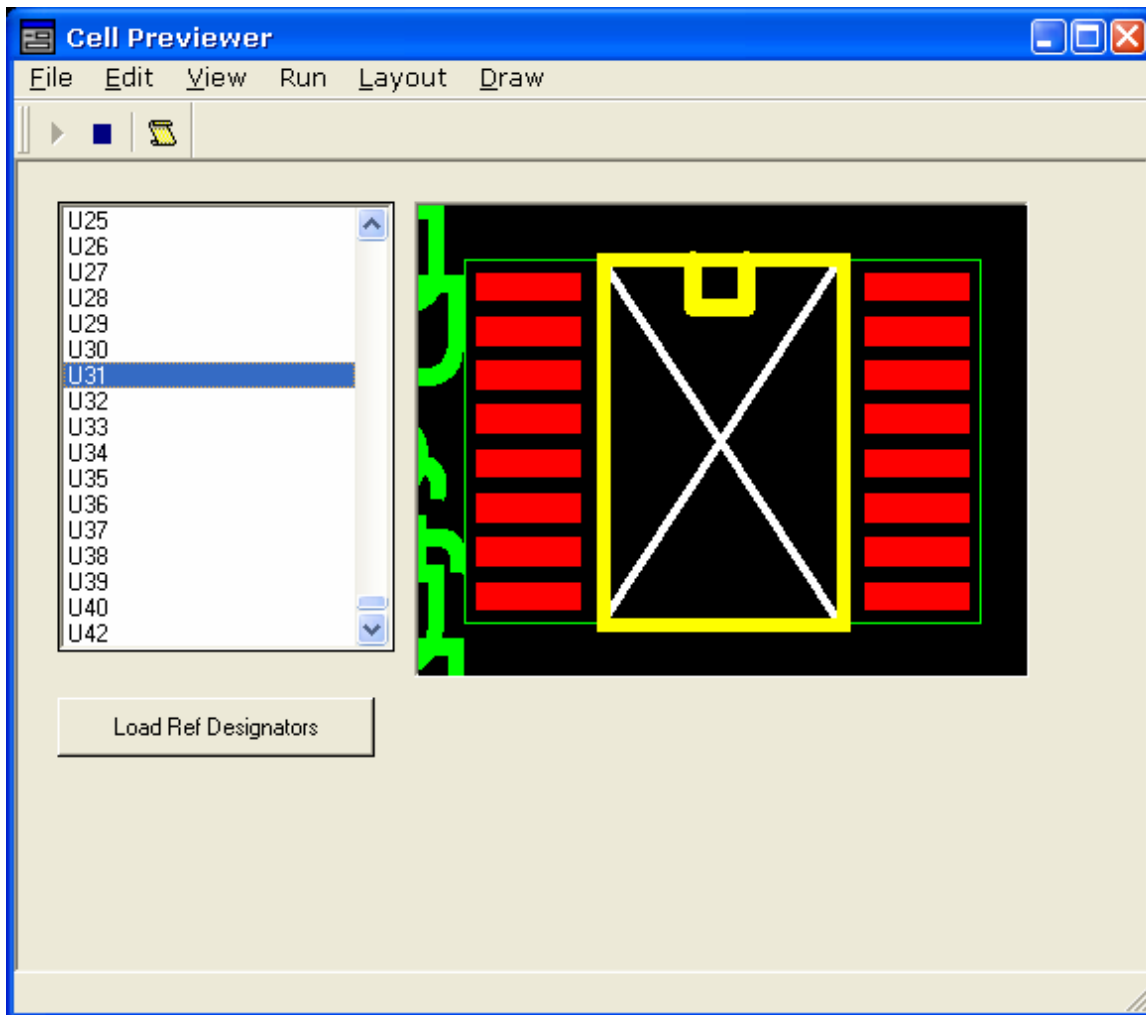
First the “This” variable is used instead of the name of the object control `lstRefDes`. The control `lstRefDes` is assigned to the “This” variable in the second line of code shown on the previous page. This is done because we are in an Event subroutine for the `lstRefDes` control object.

The `This.GetCurSel` property returns the ID number of the position in the list box of the reference designator that was chosen. This number is then passed to the `This.GetText` method and returns the Text of the current selection, which is the reference designator. The reference designator is then passed to the `FindComponent` method and assigns the component object to the “`compObj`” variable.

Once we have the component object we also have its cell name. We assign the component objects cell name to the cell preview windows cell name with this code:

```
PreviewWindow.CellName = compObj.CellName
```

27. Now, we can run our script and select a reference designator, and the cell used for that reference designator will appear in the cell preview window, as is shown in the screen shot on the following page.



28. Close Expedition without saving.

29. Wait for the next lecture

## Lab 6: Output Engines

Due to the sheer quantity of settings available for the different engines, the focus of this lab is how to run the engines from a script. The *Run<engine>.vbs* scripts have been provided with the software as templates which you can modify, if necessary.

We do, however, recommend that you spend about 10 minutes or so becoming familiar with one or more of the scripts and the engine help—Help on the actual engine itself—for a better understanding of the statements in the scripts (for example, parametric settings and terminology).

You will also find a little review of scripting initialization which we discussed in Module 3.

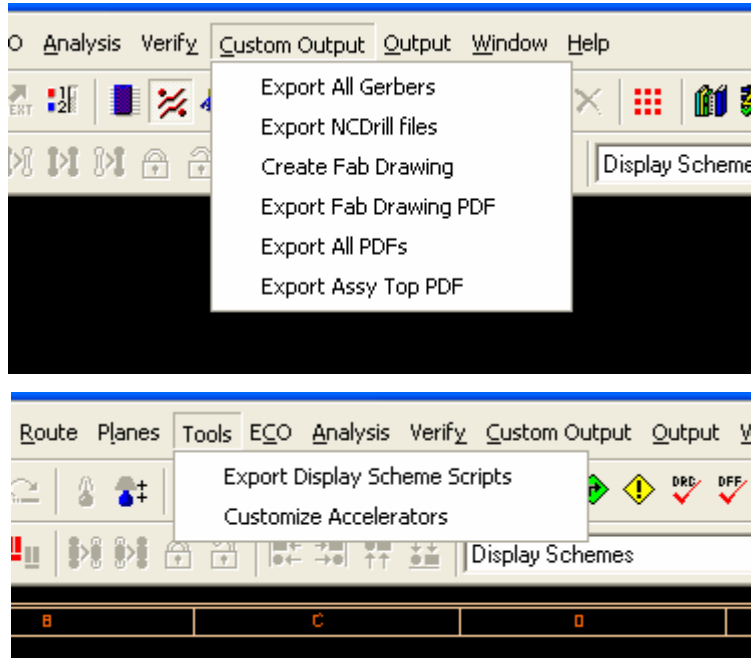
### Setup and Configuration

- Extract the contents of the *c:\scripting\_labs\lab6\scripts\_pro.zip* file to the local directory defined by *WDIR*, named *c:\scripting\_labs\local\_config*.
  - A *scripts.ini* file has been setup for you in *c:\scripting\_labs\local\_config*. It is different than the one you worked on in Lab 3.
  - Check that everything is working properly by opening the lab design in Expedition, and ensure you receive no error messages. Also ensure that there are two new pulldown menus called **Tools** and **Custom Output**.
1. Open several of the *run<engine>.vbs* files in a text editor, and identify some of the parameters which may be set up. The files are located in:  
  
*%SDD\_HOME%\standard\examples\pcb\Automation\AutoProEngines*
  2. Open the Vidar design in Expedition, if not already open.
  3. From the **Expedition Analysis** and **Output** menus, open the GUI for the engines so you can access the **Help**. Look at a few of the parameters in the script, and then review the help on the parameters.

---

## Custom Menus

4. After loading the Vidar design in Expedition, there are two custom menus that were created by the start up scripts in the *scripts.ini* file.

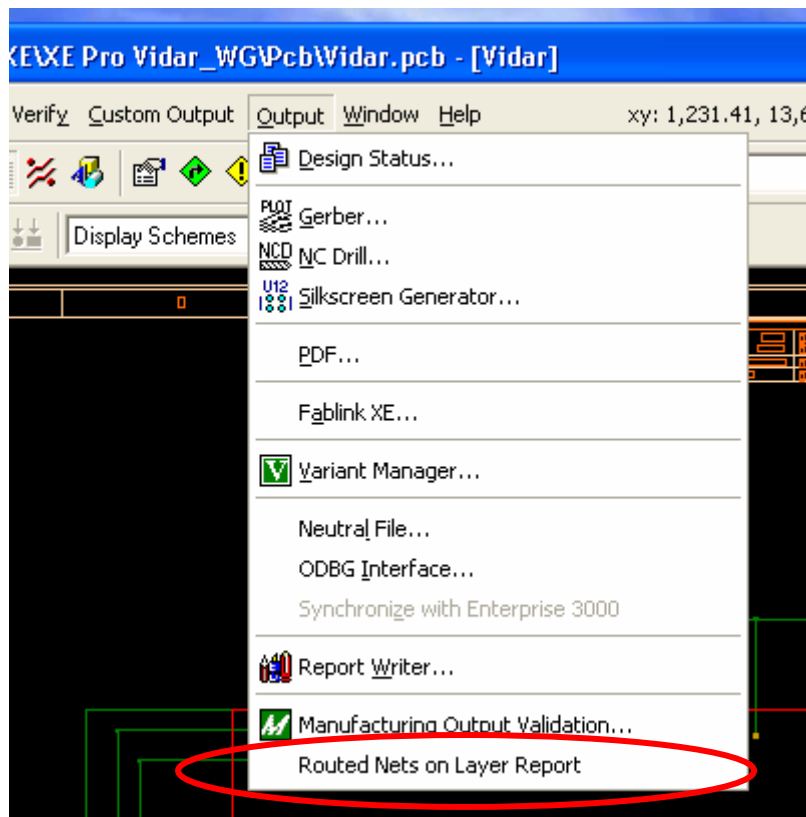


Locate the *scripts.ini* file and identify the two scripts which created the **Tools** and **Custom Output** menus.

Tools Menu: \_\_\_\_\_

Custom Output: \_\_\_\_\_

5. Open the **Output** Menu and notice that Automation also added the command **Routed Nets on Layer Report** to the bottom of the menu. This shows how automation can also modify the standard Expedition menus.



6. Look at the **Utilities** toolbar. You will see that the Routed Nets on Layer Report has been added to it. If the **Utilities** toolbar is not present, then open the **Utilities** toolbar by executing the **View > Toolbars > Utilities** command.



## Extract Gerber and NCDrill Automation

7. Run the **Export All Gerbers** command on the **Custom Output** menu. Messages will be displayed in the status bars about running and completing the export.

The script associated with that menu selection uses the *PlotSetup.gpf* file. This file has hard-coded directory paths. If the lab files were extracted to a directory other than that specified in the set up document, the paths will be wrong and the command will fail. If this happens, simply correct the paths

---

within the file. The file is located in the  
<where\_you\_extracted>\Scripting\_Labs\Vidar\_WG\pcb\Config directory. If  
they were extracted to top level directory, the local WDIR directory is  
C:\Scripting\_Labs\Vidar\_WG\pcb\Config

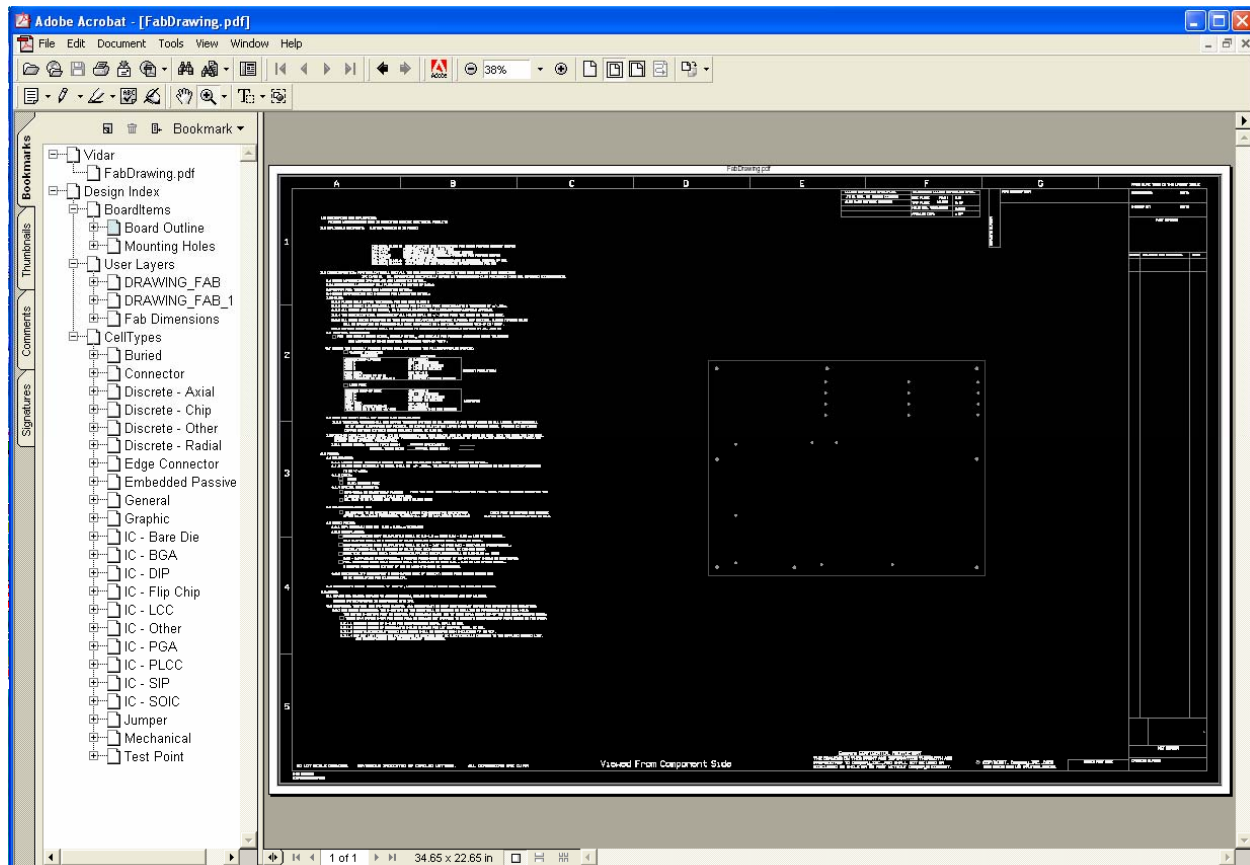
8. Next, run the **Export NCDrill Files** command on the same menu.

To confirm the output of these files open Windows Explorer, go to the  
*Gerber* and *NCDrill* output directories, and list the files.

## Drawing Creation and PDF Output

9. Create the Fab Drawing by running the **Create Fab Drawing** command on the **Custom Output** menu. This script places the border and notes cells in the design, changes the display scheme to Fab Drawing, and then fits the view around the border.
10. You now must **Save** the design. The PDF extractor extracts data from the saved database only.
11. Next, run the **Extract Fab Drawing PDF** command from the **Custom Output** menu.

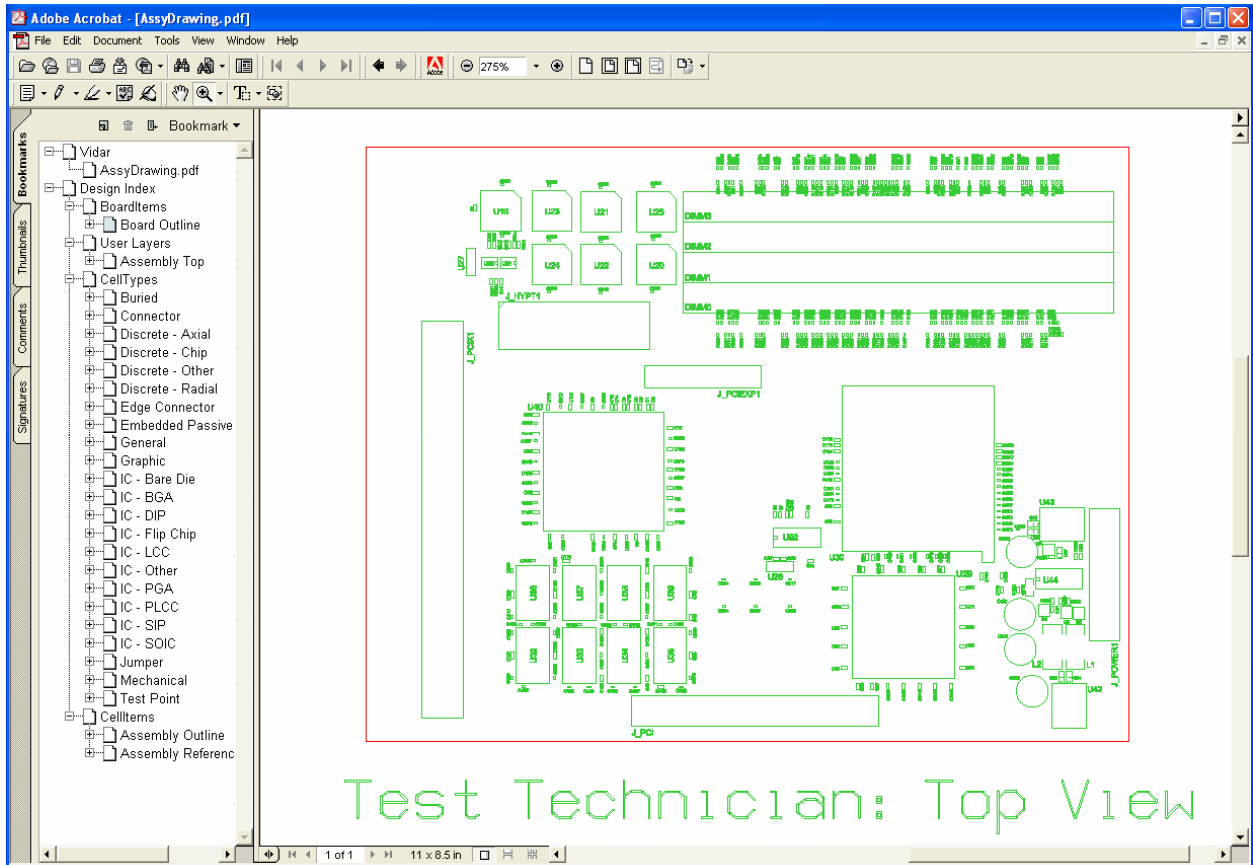
Adobe Acrobat will open automatically with the drawing in it.



## PDF Output

12. Use the **Export Assy PDF** command on the **Custom Output** menu to extract the Assembly PDF.
13. The script will load the Assembly Top display scheme and then do the **Fit Board** command.

The PDF will then be extracted, using the statements in the script—not the visible graphics—and open a file named *AssyDrawing.pdf*.



## Lab 7: Expedition Layout

You will be writing several scripts in this lab to help you focus on specific concepts without being “overwhelmed”. The ultimate goal would be for you to be able to put the individual concepts together into a single script.

Lab 7A requires you to write a script almost from “scratch”. The template file for Lab 7 A contains a “verbal” framework for the statements. Lab 7 B is mostly completed and you will be adding targeted statements which were discussed during the “Module 7 B” lecture. You will, however, need to “explore” the functionality of the script to better understand the desired functionality and add the correct statements.

**Use the Windows Scripting Technologies, Expedition Automation Help, and the lecture notes for help regarding the statements, and their formats, which you may want to use. Be sure to comment your code.**

### Lab 7A: Reporting Using the Expedition Data Model

1. Copy *C:\scripting\_labs\lab7\lab7a\_template.vbs* to *lab7a.vbs* and open the file.

Using *lab7a.vbs*, write, and debug, a script which:

2. Requires that all the variables be declared, and do not forget to declare them as your script will generate compilation errors.
3. Automatically opens and connects to the Expedition layout file:  
*C:\scripting\_labs\vidar\_wg\pcb\vidar.pcb*, and displays the application.
4. Validates the license using the If conditional and Validation function necessary for license acquisition.
5. Creates a Log File
  - a. Create the File System Object.
  - b. Prompt the user for a logfile name.

- 
- c. Acquire the project path of the open document.
  - d. Create a variable whose value is the concatenation of the project directory name and the logfile name entered by the user.
  - e. Create the logfile in the current pcb project directory.
  - f. Formulate and write the following six lines to the file:
    - i. First indicating the filename.
    - ii. Two blank lines.
    - iii. A line which lists the directory into which the file is written.
    - iv. Two blank lines.
6. Creates a collection of all the nets and reports the netname and pin count to a message box and a file. Give the user control to skip any net (move on to the next net) or to stop the reporting and move on to the next “process” of the script (cancel the current net reporting).
- a. Create the nets Collection object.
  - b. Process each net:
    - i. Create the net object.
    - ii. Create the pins Collection object and create variable with the pin count.
    - iii. Prompt user to report, skip, or cancel.
    - iv. Use the value returned from the prompt with an If conditional to control the reporting.
7. Writes some blank lines to the file; indicate that the pins/net report is complete; write some more blank lines; then indicate that the next section of the file is a refdes report (for example, “Beginning of RefDes report...”).

8. Prompts the user for a REFDES prefix (for example, U, C, etc.) to process until the user decides they do not want to process anymore prefixes. The script must accept both cases of a character.
9. Verifies that it is a valid prefix value one of these values U, C, R, P, J, X, or L, and set the script to ignore the case of the character (both upper and lower case are acceptable as input) and, if the prefix is “good”, create a component collection based upon each refdes prefix entered by the user and call the function described in step 11.
10. Issues a message box if not appropriate, and allow them to try again.
11. Reports all the components with the filtered prefix, their x,y coordinates, and the total part type count to the open log file. Use a Private Function which can be called, as there will be many different part types. The data should include both the refdes and the X,Y location pair. (Remember to include an indication of what the data means in the log file.)
12. When no more prefixes are requested, write a line to the logfile indicating that the user has closed the file, and issue a message box advising the user that the file has been closed.

## Lab 7B: Changing a Layout Using the Expedition Data Model

You will be completing a script in this lab. Once you have inserted the appropriate statements and run the script, you will be able to add rows of vias to a board. This functionality would be useful for adding stitching vias around the edges of the board to prevent radiation from board edges. You will also be able to identify areas, using the mouse, and report the number of vias in that area.

A template with most of the code written has been provided as you are to focus on the new statements introduced in this section of the module. Some steps have been written very generally, and the expectation is that you will open up one or more of the earlier scripts and simply copy and paste code (licensing, typical script beginnings, etc). At this juncture of the class, we also expect that you are familiar enough with help that you will be able to find the statement formats. One

---

more expectation is that you will create “scriptlets” or “reuse” scripts or bits and pieces of the scripts.

1. From the file system (for example, Windows Explorer), go to the directory named *C:\scripting\_labs\lab7*.
2. Copy the *lab7b\_template.vbs* to *lab7\_b.vbs*.
3. Add the “typical script beginnings” as well as the licensing If/Then conditional and Function. The locations of the statements have been highlighted in the script with a Hint and three asterisks. (See module 7A if you need more information.)
4. Attach the PCB Document scripting events. There is a PCB Document event named *OnPreClose*. Using the event will allow you perform some functionality before the script closes. Use the function prefix in the Attach events statement.
5. Add a commented line with *CreateObject* to ensure the object browser in the debugger gets populated with the Expedition type library
6. Add the Scripting type library.
7. Add a call to the Prompt of the Gui Status bar prompting the user with the following text: `Add vias - Click a Start point.`
8. Write the statement which will use *Pick* to select the padstack objects. Use **obs** for the collection object variable as the statements which process the collection are expecting that name.
9. Add a call to the motion graphics which will create a rectangle. You will find some hints in the template and the lecture notes.
10. Add a statement which loads the padstack from the database. You will find some hints and other directions in the template.
11. Create a net object, named *netObj*, and associate it with the signal GND. This will be used to connect the via to the ground plane.

When this script is executed, it will use mouse movements to determine the direction, horizontal or vertical, in which the vias are added. Briefly describe how the script determines the correct direction.

---

---

---

---

12. Add the statement which will put a via on the board, and then expand them horizontally.
13. Add the statement which will put a via on the board, and then expand them vertically.
14. Add the statement which returns the prompt to "Add vias - Click a Start point."
15. Run and debug the script. You will know that the script is properly working when you are able to add rows of vias to the design and, once the **Add Row of Vias** command has terminated, be able to draw a box around the vias and get a message box informing you of how many you have added.

When you terminate the **Via Count in Region** command, you will be prompted, via a message box, to complete your course evaluation.

---

## Appendix A: Student Challenge

In this lab, you will write a script that adds filled circles of 50 mils in diameter to all vias in a design. The circles must be 50 mils in diameter no matter what units are used in the PCB design. This allows the script to be run on any design.

The circles are to be created on a user-defined layer. The name of the user-defined layer is Via\_Circles. The user defined layer must exist in the PCB design, so create the user-defined layer in your design, if necessary. Nothing else should exist on this layer because part of the script will be to delete all data on the layer and then put in the circles.

This script must be able to run many times on the same design. So all the current circles in the design must be deleted before adding the circles back into the design. The effect is that if vias are moved, deleted, or added to a design, this script can run and basically update all the circles. This is accomplished by deleting all the old circles and then adding them back in.

Since there could be thousands of vias in a design it would be best to use the LockServer, UnlockServer, TransactionBegin, and TransactionEnd methods. This will increase the speed of the script and allow for a single undo for all the via circles.

When the script is running, let the user know what is happening by sending messages to the Expedition Status Bar.

Use the automation help to find how to use the methods and properties you need for this script. The automation help is located in the *C:\Scripting\_Labs\HelpMe* directory.

Remember to declare any variables you need with the Dim statement.

You will be using the following statements, properties, collections, enumerations, and methods:

**Option Explicit, Dim, Set, GetObject, ActiveDocument, LockServer, Transaction, TransactionEnd, Call, Gui.StatusBarText, CurrentUnit, UtilityConvertUnit, UserLayerGfxs, Delete, Vias, FindUserLayer,**

**Utility.CreateCircleXYR, PutUserLayerGfx, epcbStatusField1, epcbStatusField2, epcbStatusField3, epcbUnitsMils, UnselectAll, For Each...In, Utility.CreateCircleXYR, Count, Cstr, and MsgBox**

1. If not already open, open the *C:\scripting\_labs\Vidar\_WG\Pcb\Vidar.pcb* layout file in Expedition.
2. In Expedition, create a user layer named *Via\_Circles*.
3. Create a script file in the *C:\scripting\_labs* directory named *AddViaCircles.vbs*.
4. Set the On Error such that it will continue if run time errors are encountered. You may actually find that you do not want to set this until you have written an error handling routine. For this lab, let's experiment just to see what will happen.
5. Lock the server as you will be working with collections
6. Start a "Transaction" using a DRC mode of none.
7. Issue a message to the first status field informing the user that the script is now running.
8. Unselect all elements in your design because we will be selecting all the vias and we do not want anything else selected. Use the Unselect method in the PCB document object to do this.
9. Declare a variable to hold the current design units. Set the variable to the current design units by using the CurrentUnit method in the PCB document object.
10. The radius of the circle will always be 25 mils (50 mil diameter), no matter what units the design is using. So you must convert the 25-mil radius to the current design unit.
  - a. First declare a variable to hold the circle radius (not diameter).

- 
- b. Set that variable by using the ConvertUnit method of the Application Utilities object to do the conversion.

The reason for using the radius of the circle is that the CreateCircleXYR method creates the circle using the radius, not the diameter.

11. Declare a variable to hold the name of the user-defined graphics layer where on the circles will be created. Set the variable equal to Via\_Circles. This is the layer name you used in step 2 above. No other graphics should reside on this layer.
12. As this script may be run on the design multiple times, we want to only have one set of circles. Therefore, before we can add all the circles we must delete the ones that currently exist. In order to delete the circles that exist we must get a collection of any existing circles.
  - a. Advise the user that they are deleting the old circles using the second field of the GUI status bar
  - b. Declare a variable for the graphics collection.
  - c. Use the UserLayerGfxs method of the PCB document object to get the collection and set it to the variable for the collection.
  - d. Then use the Delete property to delete all the objects in the graphics collection.
13. Advise the user that they are adding new circles using the second field of the GUI status bar
14. Declare a variable for the vias collection. Use the Vias property of the PCB document object to get the collection of all vias and set it to the variable for the collection.
15. In order to add graphics to a user layer, you must set that user layer to a variable to be used as a “layer object”.
  - a. Declare a variable to hold the layer object.

- b. Use the `FindUserLayer` method in the PCB document object to set the variable with the layer object.

16. We must now use a “For Each” loop to go through each via in the via collection and use each via's XY coordinates to help place the circles. There are only two lines of code in the “For Each” loop. The “For Each” loop should look something like this:

```
For Each viaObj In viasColl
    <statements>
Next
```

- a. Create the circle object using the `CreateCircleXYR` method
- b. Use the `PutUserLayerGfx` method to place it into the design.
17. When calling the `CreateCircleXYR` method you must provide the XY coordinates for the center of the circle and the radius of the circle. Use the via object's XY coordinates and the radius variable you created in step 7 to do this. This method returns a points array. So you must create a variable for the points array and set that variable to what is returned by the `CreateCircleXYR` method.



It is important to note that the points array returned has three points. You will need this information when using the points array with the `PutUserLayerGfx` method in the next step.

18. The `PutUserLayerGfx` method has many parameters. Look in the automation help for the descriptions of the parameters.

The parameters, in order, are:

- Layer Object—Use the layer object created in step 11
- Display Width of line use to draw the circle—just set this to zero
- Number of points in the points array for the circle—There are three points in the circle points array.

- 
- Points Array—Use the points array created in step 14.
  - Filled—Set to *True* because the circles are to be filled
  - Component to attach the graphics to—We are not attaching the circles to any components, so set this to *Nothing*.
  - Units used by the points array—This should be set to *epcbUnitCurrent*, since we created the circle with the current design units This will complete all the statements that are needed in the “For Each” loop.

19. After the “For Each” loop, write out a message stating how many circles you added to your design. Write the message to the third field GUI status bar.

20. End the transaction you started in step 6.

21. Unlock the server using the `UnlockServer` method.

22. Print a message to the first field of the status bar that the script has completed.

# NOTES:

**Part Number: 070839**