Step-byStep Creation of a Simple PRAXIS Script:
A Capacitance / Inductance Meter
Revised August 2004

Creating a Script is a convenient way to customize or expand the features of the PRAXIS measuring system. The Liberty Script Designer, provided with the PRAXIS installation (even the one you can download for free), is a quite elegant development environment in which you can drag and drop controls to design your forms, and then write code to perform the desired operations. Your code can do normal programming operations (loops, if..then statements, procedure or function calls, math functions, etc.) and also can control special functions and properties of the PRAXIS measurement system.

Probably the best way to become comfortable with Script creation is to go through the process. After following along with this text while using the Liberty Script Designer, you may not qualify as a master programmer, but you should be familiar with the basics of making a Windows application to control PRAXIS programmatically and to customize its output. I'll lead you through the creation of a script that, for generality, can also be used with Praxis in its "free" or "demo" mode (which doesn't require purchase of an AudPod). You can even do the script creation described in this paper using Praxis in Free Mode. The script is intended to be used with Praxis 2.02 or later.

For this example, we will provide a simple but useful function. It is best to print this text to hardcopy (rather than read it on-screen), so you can keep your screen as uncluttered as possible during script design. It would be a good idea to also close or minimize any unnecessary Windows applications.

The Task:
An advertised feature of some measurement systems is capacitance and/or inductance measurement. Engineers will know that this is nothing more than impedance measurement in disguise -- if you know that you have a pure inductor, and you can find that its impedance is "Z " ohms when measured at "F" Hertz, then its inductance "L" is just

$$L=Z/(2*pi*F).$$

For a capacitor, the capacitance "C" is just

$$C=1/(2*pi*F*Z)$$

Of course, this means that if you can measure impedance -- and you *can*, using PRAXIS-- and can operate a calculator, you don't really need a capacitance or inductance meter. And since PRAXIS can measure complex impedance, we can take the idea a step further and also measure dissipation terms of inductors and capacitors.

Having a Script do the math and run the measurement will make LC measurement a little easier to do. And an LC measuring function is a simple enough process to be used as an example of Script development.

Step 1: Start Praxis, if it is not already running, and look at its Main Form. If you are using a "backdrop" (a graphic that hides the windows desktop when using PRAXIS), turn it off by using the Main Form menu commands "Config/Preferences/Backdrop" to remove the check mark to the left of the menu word "Use". Backdrops can inadvertently hide needed forms of the Script Designer, so they are best disabled when designing scripts

Start the PRAXIS Script Designer by using the “Scripts, Design a Script” menu of PRAXIS’ main form. The Script Designer application should appear.

Let's minimize the Praxis application, temporarily, to keep things simple (minimizing keeps an application running, but not using up large areas of the screen). Find the button on your Windows Taskbar (usually at the bottom of the screen) that is labeled "Liberty Praxis". Right-click on the button, then left-click on the word "Minimize" that is on the resulting pop-up menu. The forms of the Praxis application should disappear from the screen (but can be brought back by clicking on the taskbar button that remains).

At the top of one of the Script Designer forms is the title "Liberty Script Designer". This is the Script Designer's main form. It is best to maximize the Script Designer main form, making it stretch across the top of your screen. If you right click on the form's title bar, and if the form is not already maximized, you will see a pop-up menu option you can use to maximize the form.

The Script Designer has a number of components for your use, presented on tabs of the Script Designer form. These components are nearly identical to some found in Delphi, and similar to many found in Visual Basic environments. Components are essentially items you put on forms so the user of your Script can control or view things. The Script Designer also has a number of menu commands for managing, editing and searching code or form files, and for viewing different forms or tools.

Step 2: Create the Script Use the “File, New Praxis Script” menu of the Script Designer, and a form will appear to help you start your script.

Select the folder for the operating mode your PRAXIS script is being designed for use within:

- “VCLScripts” for use with an AudPod (full PRAXIS mode)
- “Demo Scripts” for use without an AudPod (Free, Demo mode)
- Any other folder you want (though such a script will not be as easily found in PRAXIS’ Script Launcher).

Give a name for your the new script folder (which will be the same as the name of your script). For this one, use a name like “LCMeter Script”.

In the part of the form that reads “Create the new Script by”, click in the circle labeled “Create from specified parameters”.

For the language, choose “DelphiScript” (if using this guide literally) or “VBScript” if you are using VBScript (but you’ll need to translate the code given below).

Under “Script Template”, choose “minimal”. Then click “Create Script” and several new windows will appear on your screen.

- There will be a form titled "Form1"; an editor box with a tab titled "unit1"; and an "Object Inspector".
- **Form1** (you can rename it if you like, or create more forms) is where you design your user interface and place your controls.
- The **Unit1** edit window is where you will write the programming code that responds to certain user inputs or other events. The unit template that was automatically generated includes some event handlers to allow the code to respond to PRAXIS events and to help you preserve some user’s settings between uses of your script, via an initialization (INI) file.
- The **Object Inspector** allows you to customize or adjust the "properties" (color, captions, styles) of the forms or controls to make them act the way you want them to.
- You can use the **F11** button to make the Object Inspector visible (after bringing focus to the component or form which you wish to configure it for.
- Use the **F12** button to toggle between a Form and its related Unit..

You can resize these windows as needed to improve visibility of any that you are working with.

Step 3: Set Up the User Interface. Here, you define and arrange the controls for the users of your script program. Find Form1 (or click within the Unit editor and then click F12 to make it appear). Resize the form so that it covers perhaps an eighth of your screen by dragging the edge of the form with your mouse (you can readjust it later, if you need to). You can also drag it by its title bar to other locations on the screen, if you wish.

4A: A Frequency TrackBar. We need a way for the user to select the measurement frequency at which he wants to measure the inductance or capacitance. Pure inductance or capacitance should have a constant value at all frequencies, but pure inductors or capacitors don't exist. So we will make our script very flexible.

In PRAXIS's Free Mode, impedances can only be measured at frequencies below 2kHz. So we need to restrict the users' choice to a range such as 100Hz to 2kHz. A simple way to do this without having to fuss with checking user input for validity is to give the user a "TrackBar" to control this value. Of course, if you want to change this script later to cover wider frequency ranges for use exclusively with an AudPod, you can do so,

Click the "Win32" tab on the "Liberty Script Editor" form, then click on the "TrackBar" component shown there (the names of the components will display when you pass your mouse over their images in these tabs). Click on the TrackBar component once, then click inside your Form1 to drop a TrackBar on it. It will be named "TrackBar1", as displayed in the Object Inspector. Stretch TrackBar1 to the size you want it to be by grabbing and dragging one one of the small square "handles" that appear around it. Click and hold your mouse button down within the TrackBar to drag it to your desired location on the form.

If you accidentally drop too many of these on the form, or if you decide you don't really want a component you've dropped on the form, click within it once to highlight it, then use your Delete key to remove it. If you accidentally double-click on a component such as the TrackBar, you may find that the editor pops up with some new code entered in it -- don't be concerned for now, just click again inside your Form1 and continue.

Click once on your TrackBar1 to highlight it (you'll see the little handles around its edges again). This will cause TrackBar1's properties to appear in the Object Inspector (click F11 if the Object Inspector isn't visible or use the "View" menu of the Script Designer main form to select the Object Inspector). Go to the Object Inspector and find the Property called "Max" and edit its contents to 2000. Change the "Min" property to 100. Set the "Position" property to 1000. These select the control to start at 1kHz, and allow adjustment from 100Hz to 2kHz. Set the "TickStyle" property to "tsNone". Later you can try changing some other properties on you own, but for now, these will do.

You need a frequency readout for the TrackBar. We'll use a label for that. In the Script Designer's main form, click on the "Standard" tab, then click once on the "Label" component (it looks like the letter "A"). Then click on your Form1 and drag your mouse to define the area where "Label1" should appear, and let go of the mouse. The Object Inspector will now allow you to set properties of this label (use F11 if you need to bring the Object Inspector back into view). Change its "Caption" property to the text "1000 Hz" (you can edit it within the Object Inspector or click on the small button at the right of the property to bring up a text editor for it). You can drag the Label and/or resize it as you desire. It should be placed below the TrackBar in this example.

4B: A Button. We need a way for the user to tell Praxis that he is ready to make a measurement. We'll use a button for that. You can find it on the "Standard" tab, and select it, size, and place it like you did the TrackBar and the Label. Use the Object Inspector to change its Caption property to the text "Measure LC".

4C: Readout Labels for Results. This LC meter will provide both the series inductor (or capacitor) value and the series resistance (loss factor) of the device

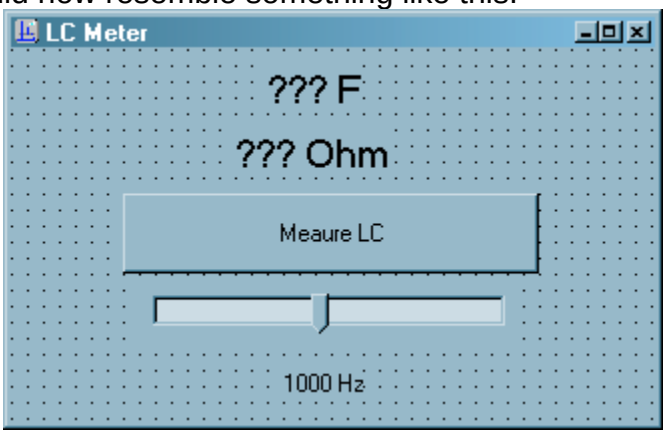
being measured. So we will need two result Labels, one for LC and one for R. Place two more Labels on the form for this purpose.

Use the Object Inspector to change the "Name" properties of these two new Labels. Click on each label to set the Object Inspector to control the desired label. Name one label "LClabel" and the other "Rlabel". This will make your program code easier to understand later. Set the Caption property of "LClabel" to the text "??? F", and the Caption property of "Rlabel" to "??? Ohm". Change the Font Size properties of these labels to perhaps "14", so that they are larger.

4D: Change Form Properties. Properties of Form1 can also be changed using the Object Inspector. Click on Form1's area (somewhere where another component is **not**) and use the Object Inspector to change the Caption property to the text "LC Meter". There are many other properties of this form you can try out later, but for now, leave it at that.

Step 5: Save your work-in-progress. Use the "File, Save All" menu of the Script Designer main form (and the Save button of the dialog form if it appears). It is a good idea to do this periodically, to protect yourself from losing work in case of computer or software problems. Remember that a Script is a very powerful programming tool -- it can be perfectly capable of crashing itself or your computer and operating system if you make coding mistakes, so don't go too long without saving your work.

Your Form1 should now resemble something like this:



Step 6: Write Code. Code writing is often the most time-consuming task in Script Development. And if you haven't programmed very much before, this section may require the most effort to follow. So take your time. If you need further clarification or understanding, or want to get into more depth, there are many books available on Delphi programming (after which DelphiScript is patterned). The script program being created in this example is quite simple, only scratching the surface of what can be done.

6A: Some DelphiScript Program Concepts

Praxis scripts are "event driven" programs. This means that the code you write is executed in response to user inputs or operating system conditions. For instance, we will write a program "procedure" (a block of programming instructions) that will be executed every time the user clicks on the Button on your form. We will write another procedure that will be executed when the user moves the TrackBar. There will be another procedure that is executed once, in response to the user starting the Script. These type procedures are called "Event Handlers".

There can be other procedures or functions that are used more generally. For instance, if you often need to update the LClabel and Rlabel captions, you can put code to do that into a general procedure, which can then be called by name from several Event Handler procedures, or from other general procedures or functions.

"Functions" are like procedures except when functions are executed, they result in a value that can be analyzed. Procedures just get executed, but don't return any result values.

Procedures or functions in Scripts should usually be declared and written so that they are specifically associated with the form that they are used with. In other words, they should be implemented as so-called "methods" of your specific "form object". The Form1 "object" is an "instance" of a TForm1 "type" -- that means that you could, if you wanted, have a bunch of similar forms on screen, all based on the TForm1 design, just like you now have a bunch of Labels on your Form1, all based on the Label component you pulled off the Script Designer "Standard" tab.

"Methods", when being called from other are usually identified by the name of the object with which they are associated. Some examples:

"Form1.DoSomething"
"Praxis.Start".

When labeling the implementation code, though, a method is identified by the type of its associated object. The type names usually start with a "T". Example:

"Procedure Tform1.DoSomething;"

It is possible to have procedures and functions that are *not* methods (i.e., that stand alone and are not associated with objects). However, we can't generally recommend using these in scripts, as there are sometimes oddities in the Scripter's operation that can confuse execution of such code.

Form Objects (or other objects) can also have variables or "fields" (places to store values) associated with them. These are declared in the body of the object Type declaration, as will be demonstrated shortly.

6B. Modify the TForm1 type declaration. Because we will be making all of our procedures and variables be associated with the Form1 object, we need to "declare" them in the TForm1 type declaration. This is the part of the code where you tell the computer how your Form1 object code is organized.

If you can't see the Unit code editor for "Unit1.pas", click on your Form1 window and then press your [F12] key You can switch back and forth between the Form1 image and its code editor, as needed, by pressing the [F12] key.

In the code editor, you should see a section that looks like this:

```
TForm1 = class(TForm)
  TrackBar1: TTrackBar;
  Label1: TLabel;
  Button1: TButton;
  LClabel: TLabel;
  RLabel: TLabel;
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure Button3Click(Sender: TObject);
 private
  { Private declarations }
  procedure HowToOpenWindowsApplets; //an example
  somevalue:double;
 public
  //****Handlers for Events that can be caused by PRAXIS' operation:
  // (add code, as needed, to their bodies, in the Implementation section)
  procedure Initscript(Sender: TObject);
  {Other Public declarations }
 end;
```

That is the "Type declaration" section for TForm1, used for our "Form1". You can see where all the components we created are declared to be part of it. Below that, you'll see the following, identifying Form1 to be an "instance" of TForm1:

```
Form1: TForm1; //NOTE: this identifier ("Form1", here) is NOT visible from
          //other scripts!
```

Event Handler methods for components on the Form can be automatically declared and outlined, as will be shown later. But we will be needing some general purpose methods and variables, so let's declare them now. (You can also add these kinds of things one at a time, later as you develop your script. But to keep it simple in this example, we will do them now).

We'll make floating-point ("double") variables to hold the magnitude and phase values that our script will read from Praxis impedance plots. So, under the line that says {Other public declarations}, add this line:

```
Magnitude, Phase: double;
```

We'll declare a procedure that will be responsible for updating our Readout Labels. So, add this line under the one just added:

```
Procedure UpdateReadouts;
```

We will also use a certain Event Handler procedure that should be a part of all Praxis scripts, called "InitScript". InitScript is called by Praxis immediately when the Script is first started, and can be used to intialize script variables or Praxis system values before the user uses any of the script's controls. Notice that there is already a declaration for InitScript provided in the Script Designer template:

```
Procedure InitScript(Sender:TObject);
```

The "Sender:TObject" parameter used by InitScript is also used in all other event handlers to tell the script code where the call to that handler originated (this value can be useful if more than one control uses the same script handler).

We need one more Praxis-activated Event Handler that was not provided by the Script Designer. It was not included because we created the script as “minimal” in step 2 above. (If we had selected it to ”Include PRAXIS Event handlers”, all it and other event handlers would have been included in the template, but it would have complicated the code in this simple example). The new Even Handler is needed so we can execute code in response to Praxis' making a new measurement (when it AutoStops). So type the following line after the InitScript declaration:

```
Procedure AcqAvgSetDone(sender:TObject);
```

We need to remember to update or implement these three procedures later!

6C. Write the Event handlers. We will have event handlers for clicking on the button, for changing the TrackBar position, and for "InitScript" (which gets called when the script is first started).

First, let's handle the TrackBar change handler, which has the simplest code. Press [F12] to make TForm1's image show, then click on the TrackBar1 components. Make the Object Inspector visible, if necessary, by using the View menu of the Script Designer main form. Then click on the "Events" tab of the Object Inspector, and double-click in the area to the right of the words "OnChange".

You'll find yourself suddenly back in the code editor, which now has a new procedure declared as "TrackBar1Change(Sender: TObject)". Your cursor will

be down in the skeleton of a procedure near the end of the text in the code editor, which is properly arranged for writing the implementation code for TrackBar1Change. Notice that the procedure is here referred to as "TForm1.TrackBar1Change".

The TrackBar1Change handler will do two things. First, it will make Label1's caption display show the new value of TrackBar1's position. Then it will call (execute) our yet-to-be-written "UpdateReadouts" procedure. So, change the procedure code so it looks like the following:

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  Label1.Caption := EngFloatToStr(TrackBar1.Position)+'Hz';
  UpdateReadouts;
end;
```

We could have also written "UpdateReadouts" as "Form1.UpdateReadouts", but that wasn't necessary, since that procedure is part of the same object as the TrackBar1Change procedure. Notice also that Delphi assigns variables and properties using ":=" (colon+equals) rather than "=" (equals). The simple equal sign "=" is used for different purposes. Also, the semicolons (";") are very important in Delphi -- they signify the end of each instruction.

Next, we'll moidify our InitScript. The format needed for its skeleton is similar to that in the TrackBar1Change procedure. InitScript will choose the Stimulus and Acquisition types for Praxis and set up some configuration parameters. Look through the Unit1 code until you find the InitScript code template, which should look like this:

```
procedure TForm1.Initscript(Sender: TObject);
//Here, initialize values for the script (don't do that in TForm1.Create)
var Ini: TIniFile;
begin
  //Any first settings that are specific to Praxis:

  //Ini File handling for user's settings:
  Ini := TIniFile.Create(Praxis.ScriptPath+'ScriptIni.ini');
  Try
    //The code below will position the script's main
    //  form on the screen to where it was last moved by the user.
    //  Use the Ini file to also restore any other values you want to keep
    //  between uses of your script.
    Form1.Top:=Ini.ReadInteger('FormPosition','Top',0);
    Form1.Left:=Ini.ReadInteger('FormPosition','Left',0);
    //Read back any other Ini filed values that you store in TForm1.FormClose:
    somevalue:=Ini.ReadFloat('IniSectionName','MyFloatingPointIniValueName',
      1.2345); //1.2345, here, would be the default value for the first use
  Finally
    Ini.Free;
  End;
end;
```

Just before the last line (“end;”), enter the code lines as given below:

```
If Praxis.DemoMode then // do the following if there's no AudPod:
 begin
  If (not Praxis.HaveDemoProbeMixerConfig) then
  begin
    ShowMessage('You need to first run the "SoundCard Probe Calibration" script.');
    Close;  //End the script
  end;
  If Praxis.DemoRtaMode then Praxis.DemoRtaMode := False; //use Probe settings
 end;
 //Initialize variables and Praxis system settings:
 Magnitude : =0;  Phase :=0;
 Praxis.Stimulus := stiChirp;
 Praxis.ChirpLinearSweep := False;
 Praxis.ChirpStartFreq := 0.1;
 Praxis.ChirpStopFreq := 2000;
 Praxis.ChirpTimeLength := 200E-3;
 Praxis.Acquisition := acqImpedance;
 Praxis.AutoStop := True;
 Praxis.AverageLimit := 1;
 Praxis.AverageType := atCoh;
 If not Praxis.DemoMode then //with the Audpod:
   Praxis.InputSelect[1] := inProbe1;
Praxis.UseAverage:=True; //because PRAXIS always turns this OFF when a script starts.
```

Any text following double slashes ("//") on a line is a comment, to help explain what the code is supposed to do. Begin and End should surround all code in a procedure, or and any other sections of code that are to be blocked together. You can learn more about most of the "Praxis" properties used below from the Praxis Manual or Help files.

The code for the AcqAvgSetDone procedure is next, to implement the Event Handler that we declared earlier. No code template for this was automatically generated, so we need to enter this ourselves. It can go in any order with the other procedures or functions, but we’ll put it near the end of all our code in this editor, just before the last line that reads “end.” (the one that has a period after it, not a semicolon). The code we’ll add merely calls the UpdateReadouts procedure when the impedance curve has been remeasured. Copy this in:

```
Procedure  TForm1.AcqAvgSetDone(sender:TObject);
begin
  UpdateReadouts;
end;
```

We also need to add code to respond to Button1 being clicked. Get to Button1's Object Inspector, and use the Events tab there. Double-click in the space to the right of the words "OnClick". The event handler will be declared and your cursor will appear in the code editor ready to edit the new procedure

"TForm1.Button1Click(Sender: TObject)". This just starts Praxis' acquisition engine. Here's how the handler should look when it's done:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Praxis.Start;
end;
```

6D. Write other Procedure Code. In this example, we have deferred much of the work to the UpdateReadouts procedure. That procedure is responsible to use Praxis' data trace marker to read the impedance value at the desired frequency. It then needs to convert the magnitude/angle data into resistance and reactance values, and convert the reactance to the appropriate capacitance or inductance value. Finally, the procedure updates the readouts with the calculated values.

The code is given below, and can be entered at the end of the other code, just before the final line (“end.”). The "//" comments given in the code describe how it is supposed to work (you don't need to include them in your code, if you don’t want to):

```
Procedure TForm1.UpdateReadouts;
var resistance, reactance, frequency: double;
begin
  If (Praxis.PrimaryPlot.PlotDescription = 'Impedance Data') then
    // that's "=", not ":=" above! Makes sure that the plot is applicable.
  begin
    Praxis.PrimaryPlot.UseMarker[1] := True; //use marker 1 to read data
    Praxis.PrimaryPlot.MarkerFrequency[1] := TrackBar1.Position;
      //set the marker to the commanded frequency
    Praxis.PrimaryPlot.MarkerNext; Praxis.PrimaryPlot.MarkerPrevious;
      //the above two instructions jiggle the marker onto an exact data point
    Frequency := Praxis.PrimaryPlot.MarkerFrequency[1];
    Magnitude := Praxis.PrimaryPlot.Marker[1,1]; //read the marker, mag
    Phase := Praxis.PrimaryPlot.Marker[2,1];     //read the marker, phase
    resistance := Magnitude*cos(Phase*pi/180); //cos acts on radian angle
    reactance := Magnitude*sin(Phase*pi/180);  //sine acts on radian angle
    Rlabel.Caption := EngFloatToStr(resistance) + ' Ohms';
    If (reactance>0) then //an inductive impedance
      LClabel.Caption := EngFloatToStr(reactance/(2*pi*Frequency)) + 'H'
    else if reactance<(-1e-12) then //capacitive.
      LClabel.Caption := EngFloatToStr(-1/(reactance*2*pi*Frequency)) + 'F'
    else LClabel.Caption := '';  //Avoid possible divide by zero error
  end;
end;
```

Step 7: Test the Script. Because you have been entering tested code, you have a good chance that the script will work correctly for you first time. But there's also a good chance that typographical errors were made. So, before you try running your script… save it again! (Use the "File, Save All" menu).

To run this script successfully, you'll need to either

- have a calibrated AudPod connected, or
- have your system set up without an AudPod, and with the Free Demo Mode "Probe Cal" script successfully completed.

You'll also need to set up your system to measure impedance. You can find connection diagrams for this from the Measurement Guides (for AudPod), or in the "demo Thiele Small Measurements" script for Free Mode (connect your capacitor or inductor in place of the loudspeaker in the diagram).

When you are ready, click on the "Liberty Praxis" button at the bottom of your screen to Restore it back to view. Click on the "RunScripts" menu of Praxis' "Main Form" to open the Script Launcher. Look down the list of scripts available under "Select Scripts" until you find your "LC Meter" script and click on that. You'll notice that there is not description or picture shown for it, since you haven't made one for it (yet). Click on the "Launch Script" button (note: you can also launch a script you are developing from the Script Designer, using the Run menu or the small “Run” button).

One of two things may happen. Your script may start, or you may get an error message. If an error message occurs, note any line numbers that are reported in the error message. If the system offers to "continue execution", click "Cancel" and close any other error messages that appear.

A reported line number can be used to find an error in your code. If you got an error, go back to the Script Designer and get to the code editor. In the lower left corner of the code editor window are shown two numbers that indicate the line number and character number of your cursor in the editor. Move your cursor to locate the same line number that was reported in the error message, and look in that line, the one before it, and the one after it for a mistake (for some reason, the reported line is usually off by 1). Your mistake may also be an open single-quote (') or a missing semicolon (;) on an earlier line.

When you get the program started without error messages, be sure to go to Praxis' "(acq)" configuration form and type in the known value of the reference resistor you are using in your test setup. As usual, the best value of reference resistor to use is one close to the value in impedance that you expect to be measuring. For most components used in passive loudspeaker crossovers, 8 to 10 ohms is a good value. You may need to adjust the input level sliders for proper operation with the AudPod (such adjustments were not included in this example script, to keep it simple).

Then click on the "Measure LC" button and see how it works.

Note: If you make a serious error (such as getting your script stuck into an endless loop so that it cannot be closed) during more adventurous experiments, you may need to use Ctl-Alt-Del to close down both Praxis and the Script. If you

can get PRAXIS' attention, but not the script's, you can close down the script using the "Scripts, Abort Running Script" submenu on PRAXIS's main Form, but this may cause side effects and should be avoided when possible.

Step 8: Polishing Up. As you have seen, you do not need to provide a bitmap picture or description file in your scripts directory to be able to run it. But if you are making the script available to others, these would be good to add. Just use the options under the Script Designer's "Tools" editor to bring up the appropriate editor and save each afterwards.

You can probably think of many ways to embellish or improve on this script. A help menu could be added to provide some brief user instruction. A control could be added so the user can enter the Reference Resistor value right on the script form. All sorts of appearance improvements can be made (illustrations, colors, better centering, etc). You could change it so that the measurement is made continuously.

And, of course, you can begin to write entirely different scripts for other purposes, now that you know how to get into and out of the Script Editor.