Measurement Automation with SigLab

Measurement automation can prove to be a daunting task with outdated software paradigms and "rack and stack" technology. Combining MATLAB Graphical User Interface (GUI) tools with SigLab® measurement functions produces a Windows-based automated measurement solution that is powerful and easy to implement.

Overview

The Steps to Automation

The three basic steps to creating an automatic measurement and test solution are:

1. Define measurements required.
2. Define measurement limits.
3. Write a MATLAB M-file that automatically makes the measurements and compares the results to the limits.

Features such as operator log in/out, device-under-test serial number entry, test report generation, and measurement archival are often required. Since MATLAB is a general programming language (albeit optimized for numerical analysis and visualization), these features are easy to implement.

Figure 1 outlines the overall automation strategy. A virtual instrument is used to set up the proper measurement parameters. This information along with measurement data is stored in File 1. A limits generation utility is used in conjunction with the measurement data in File 1 to create test limits stored in File 2. The measurement parameters and limits are used by the auto test M-file during the execution of the test.

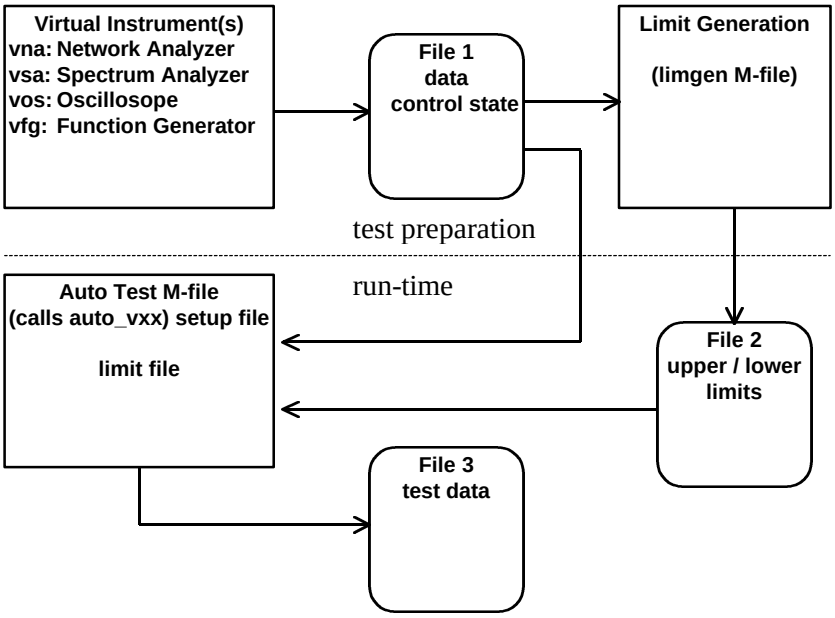


Figure 1 - Automation Strategy

The Strategy

One of the most difficult tasks in automating measurements using traditional instrumentation is setting up the instrument state. Cryptic strings for the IEEE-488 interface are difficult to modify, and tedious to create. The SigLab automation strategy (see Figure 1) completely bypasses this error prone step by using the virtual instruments (VIs) to define the measurement state. Since the VIs save both control state and measurement data (File 1), the VIs are the perfect tools to define and refine the measurement parameters. The limit generation utility uses the data acquired by a VI to establish the pass/fail limit criteria (File 2).

Once the measurements have been setup and the limits generated, the auto test M-file uses the measurement definitions and limits to implement the test.

Measurement Definition

The VIs Allow Easy Measurement Setup

Optimizing a measurement when many measurement parameters are involved can be a difficult and time consuming process. Using a VI overcomes this difficulty by providing a graphical means of measurement control. Virtual instruments are provided covering the main classes of measurements performed on dynamic systems:

Oscilloscope	vos.m	vos_auto.m
Spectrum Analyzer	vsa.m	vsa_auto.m
Network Analyzer	vna.m	vna_auto.m
Function Generator	vfg.m	vfg_auto.m

Figure 2 shows the Network Analyzer VI with a frequency response measurement of a dynamic system (an electrical filter). To make this measurement, the user must specify approximately 14 different parameters. Specifying the automated test setup using a graphical interface is much more convenient than embedding the measurement parameters in software.

Optimizing the transfer function measurement parameters is easy with the vna GUI based virtual instrument.

When the measurement has been made, the user specifies the file to which both control state and data are saved. The contents of this file are used to set limits and define the measurement setup for subsequent automation.

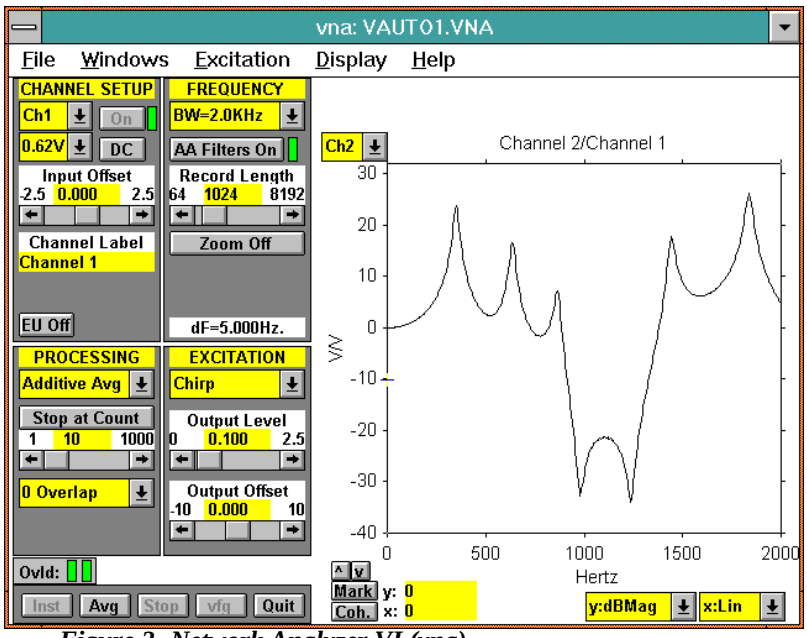


Figure 2- Network Analyzer VI (vna)

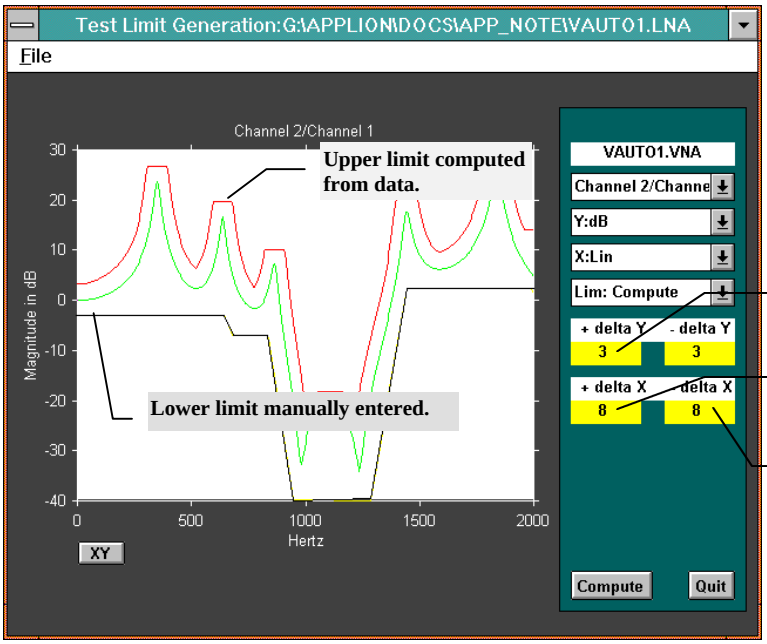


Figure 3 shows the limit generation utility with transfer function data from the network analyzer VI.

The upper limit was computed using the following 3 parameters:

Add 3 dB to measurement.

Slide data to right 8 measurement intervals.

Slide data to left 8 measurement intervals.

The lower limit was initally computed (using -delta Y) and then manually edited.

Figure 3 - Limit Generation Utility

The user designates the file to store the state of the VI and the measurement data. In this case, the file is **vauto1.vna**. The limit generation program uses the measurement data in the file to set test limits. The auto test M-file uses the state information in the file to duplicate the measurement parameters of the VI.

Limit Generation

Limgen.m

The **limgen** utility can be used to generate test limits. There are two primary approaches that can be used to generate the limits:

1. Compute from measured data.
2. Enter graphically from test specs.

In both cases, the initial starting point is a VI file which contains a set of measurement data. This file could contain a measurement of a typical device, or by using MATLAB, the user may have combined the measurements from a large number of devices to build up a composite measurement.

Figure 3 shows the transfer function measurement with upper and lower limits. The upper limit is set by the user entered error bands. The "+delta Y" value is added to the data while the data set is translated to the right and left by the amounts specified in the "+/- delta X" fields. These allow for error bounds to account for errors in magnitude (y) and frequency (x).

The lower limit was computed using the "- delta Y" parameter and subsequently edited by using the mouse. The editing capability allows the test limits to be graphically entered. Simply compute the limits from the data and then edit them.

Although not shown in Figure 3, phase limits may also be specified for the transfer function data.

Computed Limits in MATLAB

Cases often arise where graphical limits are not appropriate. For these situations, the limits can be computed directly in MATLAB and applied to the test data.

Automated Measurements

We will now shift our focus from the tools used in test preparation to those used during the run-time phase.

Using vxx_auto.m Functions

Each of the four primary virtual instruments has an automated counterpart which makes identical measurements, but lacks the graphical user interface.

The VIs and their automated counterparts are as follows:

Oscilloscope	vos.m	vos_auto.m
Spectrum Analyzer	vsa.m	vsa_auto.m
Network Analyzer	vna.m	vna_auto.m
Function Generator	vfg.m	vfg_auto.m

Measurement data is not produced by the function generator, but the generator is useful as a source of excitation when using the oscilloscope or spectrum analyzer VIs.

The syntax of the automatic measurement functions is shown in Figure 4. The 'init' action must be used once to download code (if necessary) and initialize some internal variables. The next call to the function will perform a measurement. If the 'meas2out' action is specified, the measurement will be made with the parameters specified in Setup_File (a **vna** file in this case), and returned in the four output variables Out1 through Out4. If the 'meas2file' action is invoked, all of the above happens and a file with the name "Output_File" is also written. This file has exactly the same format as a regularvna file.

To maximize throughput, the previous measurement setup is used if the Mode parameter is specified as 'repeat'. In this case, the measurement setup parameters are not sent to SigLab, since they are assumed to be the same as in the previous measurement. Repeat mode is useful for creating adjustment loops.

Control via siglab.dll

In most cases, the four automatic measurement programs will prove to be efficient and sufficient to create powerful automatic test routines. Occasionally a lower level of hardware control is necessary. This control can be attained by using the siglab.dll MEX interface routine directly.

Automation Examples

Minimal User Interface – a Simple Test

The following example will demonstrate:

- SigLab initialization.
- Measurement of a transfer function.
- Comparison against magnitude limits.
- Report pass/fail status.

```
function [Out1,Out2,Out3]=vna_auto(Action,Setup_File,Mode,Output_File);
       Actions
       'init'  must initialize hardware before measurement
       'meas2out' returns:
              Network measurement (XferDat)              in Out1
              frequency vector (Fvec)                    in Out2
              Coherence                                  in Out3
              channel status (ChanStat)                  in Out4
       'meas2file'  returns new
              XferDat CohDat and vi_timestamp to Output_File as with above variables.
       Mode
              'new'          use setup information in Setup_File
              'repeat'       repeat measurement using the setup defined by the previous
                             'new' command
```

Figure 4 - Automated Measurement Routine Syntax (vxx_auto.m)

The program is shown in Figure 5 with comments in the left column. It requires two lines of code to make the measurement defined by the setup information in the **vauto1.vna** file. Two more lines accomplish the upper and lower limit comparisons. The comparison results are then reported to the operator. Those who have struggled with IEEE-488 commands and Basic (or similar low level languages) will appreciate the efficiency of this approach.

A Graphical User Interface

The next example will demonstrate the steps required to add a simple user interface to the previous test.

The example will demonstrate the following:

- Create a window.
- Create an axis for plotting.
- Provide a user Run button.
- Perform a network measurement.
- Plot the measurement results.
- Plot the limits.
- Report pass/fail information.

The user interface is shown in Figure 6. The upper and lower limits surround the measured frequency response of the device under test.

The code to implement the automated measurements with the graphical interface is shown in Figure 7. To understand the code fully one should be familiar with MATLAB; however, even the uninitiated can follow the general structure.

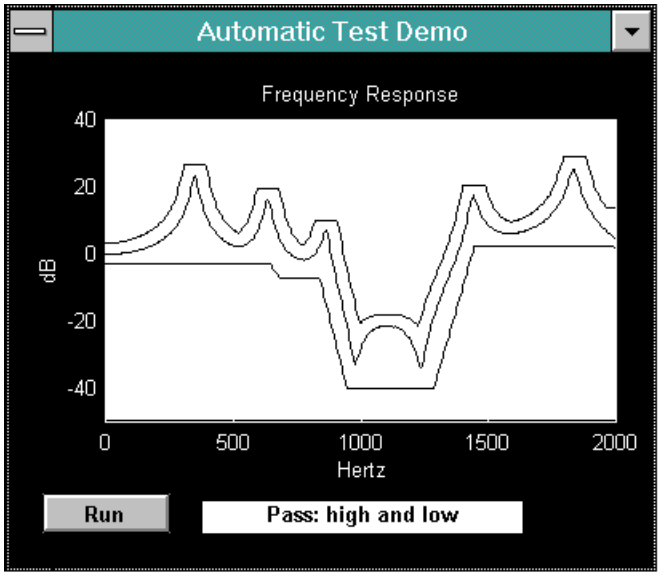
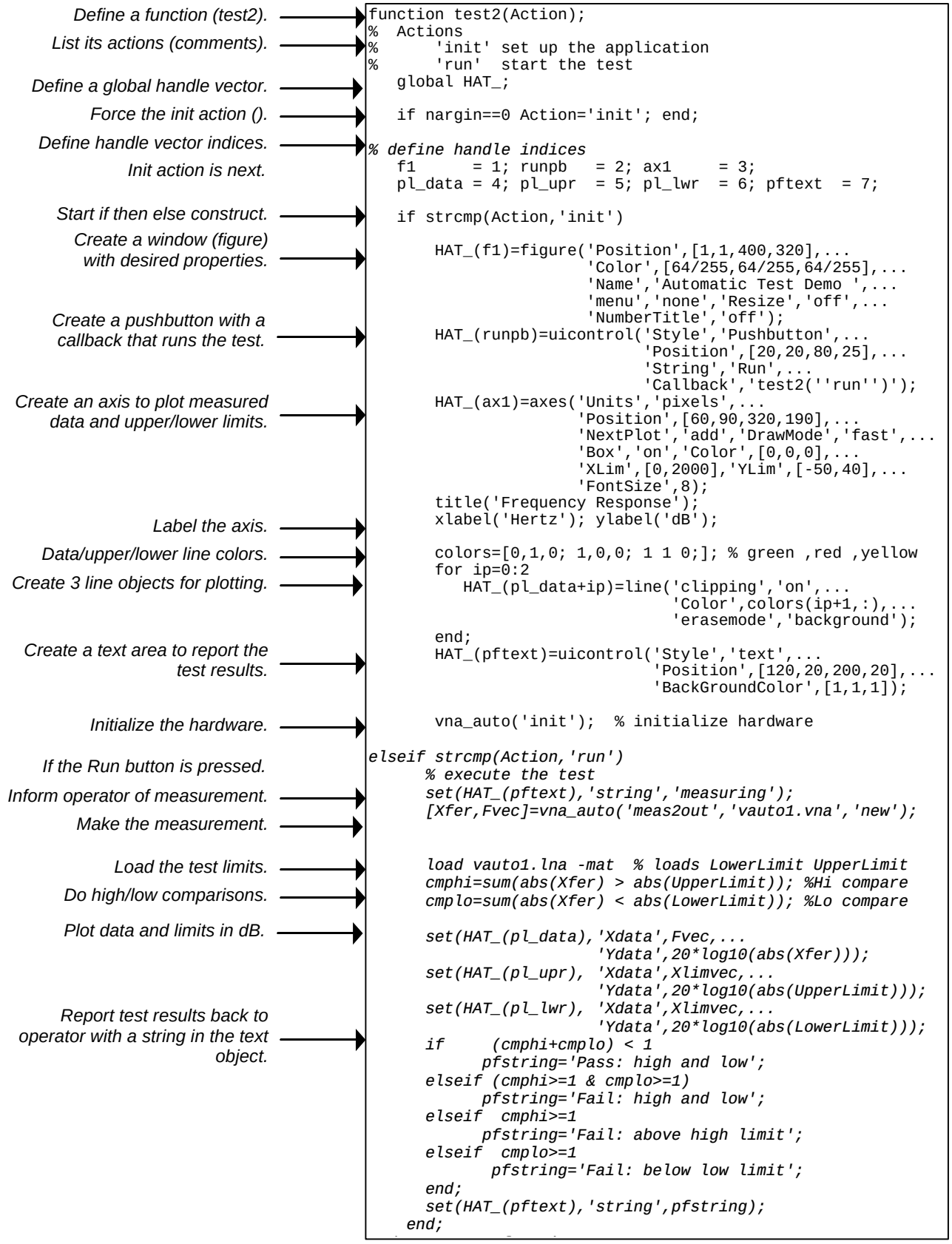


Figure 6 - Sample user interface

Clear command window. →
Initialize the hardware. →
Make measurement. →
Load the test limits. →
Compare against high. Compare against low. →
Report pass/fail status using the comparison results and the disp() function to return text to the command window. →

```
% test1.m: simple automatic test demo script m-file
  clc;


  vna_auto('init');

  [Xfer,Fvec] = vna_auto('meas2out','vauto1.vna','new');


  load vauto1.lna -mat

  cmphi=sum(abs(Xfer) > abs(UpperLimit));
  cmplo=sum(abs(Xfer) < abs(LowerLimit));



  if     (cmphi+cmplo < 1      disp('Pass: high and low');
  elseif (cmphi>=1 & cmplo>=1) disp('Fail: high and low');
  elseif  cmphi>=1             disp('Fail: above high limit');
  elseif  cmplo>=1             disp('Fail: below low limit');
  end;
```

Figure 5 - An Automated Test Example




```
function test2(Action);
%  Actions
%      'init' set up the application
%      'run'  start the test
   global HAT_;

   if nargin==0 Action='init'; end;

% define handle indices
   f1      = 1; runpb   = 2; ax1     = 3;
   pl_data = 4; pl_upr  = 5; pl_lwr  = 6; pftext  = 7;

   if strcmp(Action,'init')

       HAT_(f1)=figure('Position',[1,1,400,320],...
                       'Color',[64/255,64/255,64/255],...
                       'Name','Automatic Test Demo ',...
                       'menu','none','Resize','off',...
                       'NumberTitle','off');
       HAT_(runpb)=uicontrol('Style','Pushbutton',...
                             'Position',[20,20,80,25],...
                             'String','Run',...
                             'Callback','test2(''run'')');
       HAT_(ax1)=axes('Units','pixels',...
                      'Position',[60,90,320,190],...
                      'NextPlot','add','DrawMode','fast',...
                      'Box','on','Color',[0,0,0],...
                      'XLim',[0,2000],'YLim',[-50,40],...
                      'FontSize',8);
       title('Frequency Response');
       xlabel('Hertz'); ylabel('dB');

       colors=[0,1,0; 1,0,0; 1 1 0;]; % green ,red ,yellow
       for ip=0:2
          HAT_(pl_data+ip)=line('clipping','on',...
                                'Color',colors(ip+1,:),...
                                'erasemode','background');
       end;
       HAT_(pftext)=uicontrol('Style','text',...
                              'Position',[120,20,200,20],...
                              'BackGroundColor',[1,1,1]);

       vna_auto('init');  % initialize hardware

elseif strcmp(Action,'run')
      % execute the test
      set(HAT_(pftext),'string','measuring');
      [Xfer,Fvec]=vna_auto('meas2out','vauto1.vna','new');


      load vauto1.lna -mat  % loads LowerLimit UpperLimit
      cmphi=sum(abs(Xfer) > abs(UpperLimit)); %Hi compare
      cmplo=sum(abs(Xfer) < abs(LowerLimit)); %Lo compare

      set(HAT_(pl_data),'Xdata',Fvec,...
                        'Ydata',20*log10(abs(Xfer)));
      set(HAT_(pl_upr), 'Xdata',Xlimvec,...
                        'Ydata',20*log10(abs(UpperLimit)));
      set(HAT_(pl_lwr), 'Xdata',Xlimvec,...
                        'Ydata',20*log10(abs(LowerLimit)));
      if     (cmphi+cmplo) < 1
            pfstring='Pass: high and low';
      elseif (cmphi>=1 & cmplo>=1)
            pfstring='Fail: high and low';
      elseif  cmphi>=1
            pfstring='Fail: above high limit';
      elseif  cmplo>=1
             pfstring='Fail: below low limit';
      end;
      set(HAT_(pftext),'string',pfstring);
   end;
```

Figure 7 - An Automated Test with a Graphical User Interface

Interfacing to Auxiliary Devices

IEEE-488

Professor Godfrey at Stanford University has created some M-files to interface with instruments on the IEEE-488 bus.

The entire interface is written in M-files, so it is relatively easy to work with. It requires a GPIB card from CEC Corp. and their basic (CECHP.EXE) software. There is a README file on the ftp site that provides more information.

To get the software:
ftp isl.stanford.edu
login; anonymous
cd pub/godfrey/reports
get readme

if you want the whole thing:
binary
get gpib.zip

If you want your name on a mailing list about the software, send e-mail to: godfrey@isl.stanford.edu

Serial Port

Another useful means of controlling devices other than SigLab is by way of a serial port. Spectral Dynamics has a simple serial routine for string oriented I/O that may be called within the MATLAB environment.

Conclusion

The MATLAB graphical user interface tools and the SigLab system may be combined to form a first class Windows based automatic test and measurement system. Since MATLAB has all the features of a general purpose programming language, it is easy to communicate with auxiliary devices.

If desired, the MATLAB/SigLab combination can also be used from within other programming environments. This methodology does not use the MATLAB GUI tools, but treats the MATLAB/SigLab combination as a measurement engine.

For more information contact:
Spectral Dynamics
1010 Timothy Drive
San Jose, CA 95133-1042
Phone: (408) 918-2577
Fax: (408) 918-2580
Email: siglabsupport@sd-star.com
www.spectraldynamics.com

Printed in U.S.A.