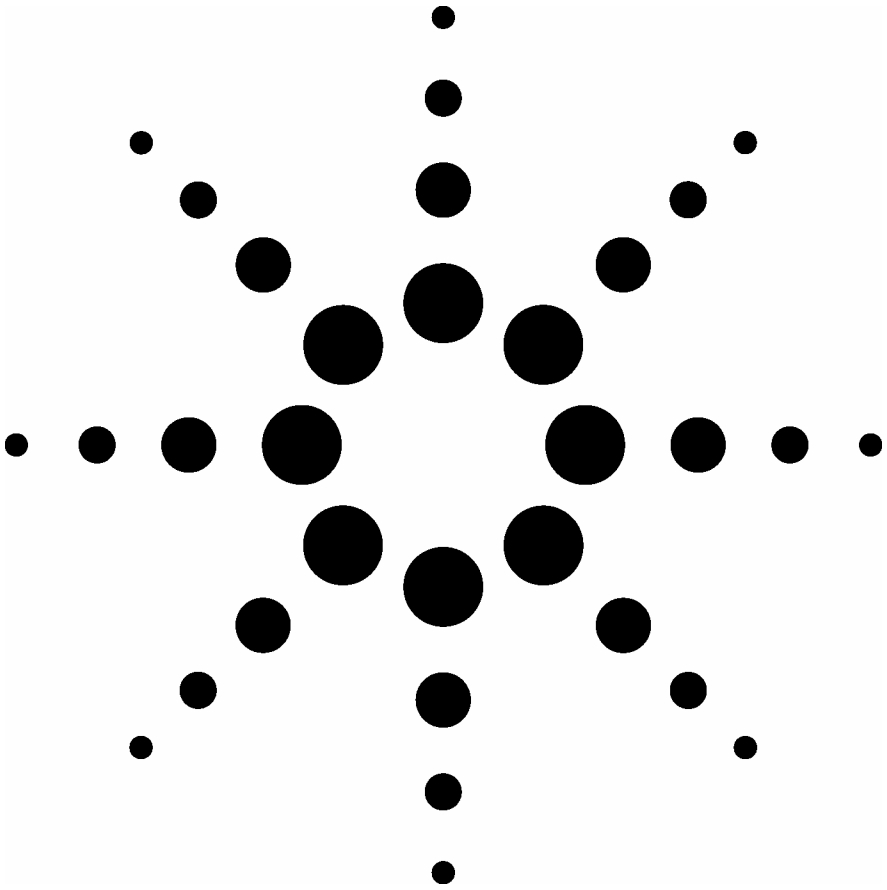
Implementing a Supportable VEE Architecture

White Paper



PC (Mac) MacDonald
Agilent Technologies

Agilent Technologies

Implementing A Supportable VEE Architecture

PC (Mac) MacDonald
Systems Engineer
Application Engineering Organization

Rev. B.00

How Can I Build A Supportable VEE Program?

There are many ways to construct a VEE program that is simultaneously:

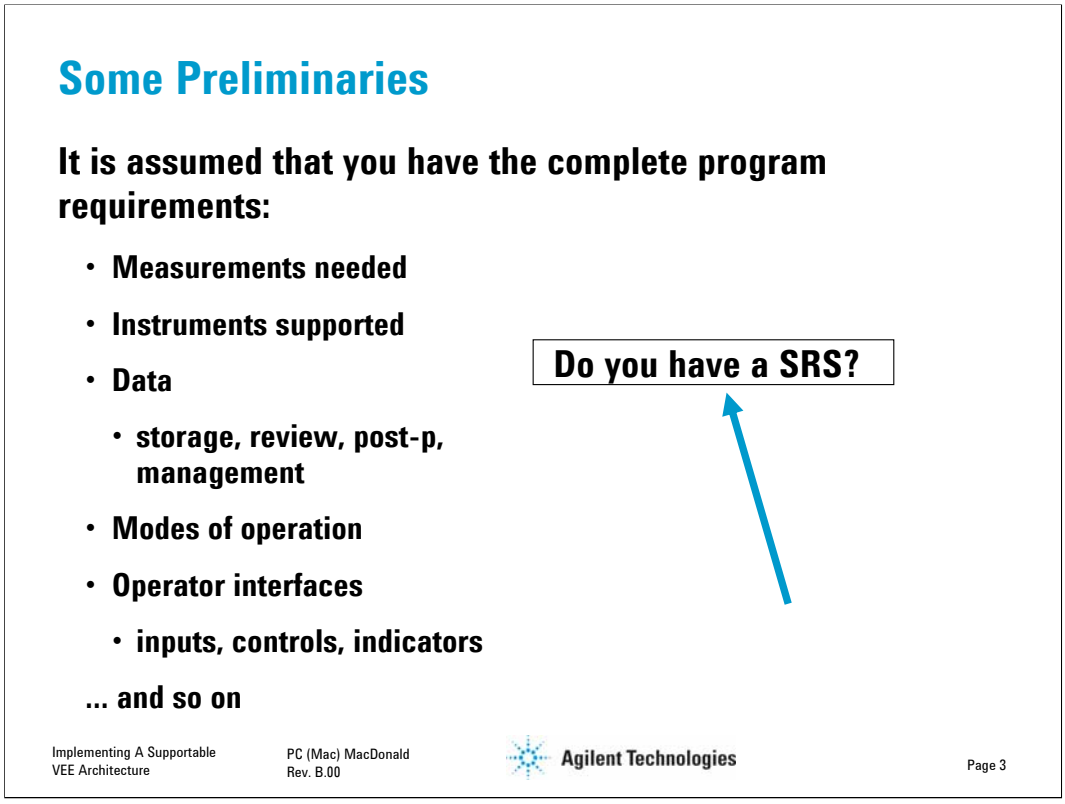
- **Supportable**
- **Expandable**
- **Reasonably straightforward to debug**

•There are plenty of poor, wanting, and even frightening ways to build software

•There is no single best approach, magic pill, or "silver bullet"

•This presentation illustrates the general way I approach constructing a VEE program

  •I will do this by walking through the planning and architecture of a fairly small, yet sophisticated program



Rules of Engagement

Until you have a firm understanding of *exactly* why the program is needed and what it is expected to do, you can not expect to complete the programming task efficiently. In fact, until you have put that stake in the ground that defines in writing what you are designing, you will continue to suffer from *creeping elegance, moving design criteria, and other generally elusive & arbitrary requirements.* It is imperative to have consensus among customers, users and management that the software requirements are properly defined. *This is particularly true if you are an 'in-house' resource.*

• If you don't start with firm and accepted requirements, your work will always be late, never be right, you'll work lots of evenings and weekends, and your peers will be continually blaming you for shoddy work because they probably won't even grasp that they played a part in the problem.

• It takes a great deal of effort to produce a good specification. However, people who refuse to work any other way find that they get the software project done in about half the time it takes persons with less discipline. They spend up to half of their software development project time defining requirements and developing architectures before a single line of software gets written. They spend lots less time debugging than their less organized peers.

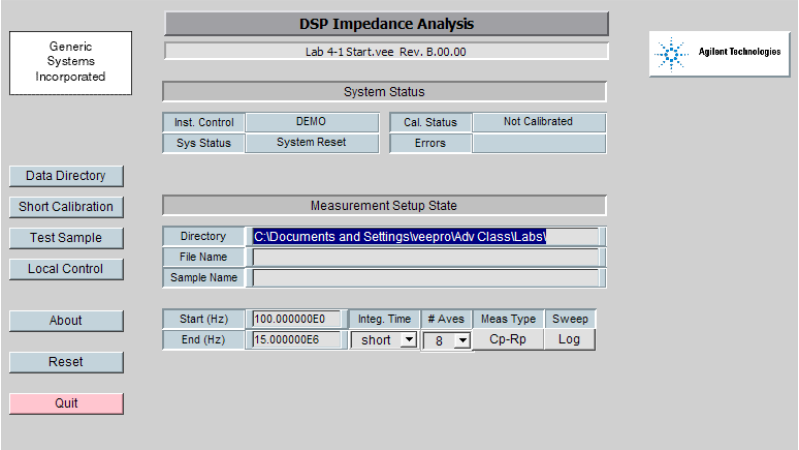
• Once your execution task has begun, **don't even agree to consider** changes without schedule relief unless you have budgeted time in your project for those tasks. Make sure to get buy in on this before you begin programming. If changes are imposed and demanded, don't be shy about putting in writing the impact this will have on schedule. This will initially be difficult for most sponsors to digest, but over time they will come to respect you for refusing the impossible and its associated failures.

• Most of us don't like to plan. Many of us find it painful to negotiate. Those of us that can manage both have an edge.

Architectural Topics We'll Address

Program Substructure

Program Structure



This is the main view of the program we are about to discuss

Designing a Program Substructure

A solid program requires a highly organized way to manage all this stuff:

- **What MODE am I in?**
- **Which SETUP did I use?**
- **How much time is left in the test?**
- **Did I remember to calibrate?**
- **Is test STATUS OK?**
- **What the DUT temperature?**
- **What specimen is in the chamber?**
- **How many averages are left?**

- Even the simplest programs require keeping track of a great many things.

Organizing Program Information

Most program substructure can be broken down into three basic categories:

- **Setup**
- **Status**
- **Data**

Larger Programs can benefit from further segregation:

- **Control**
- **Calibration**
- **Etc.**

- These are the groups that seem to work for me. Your experience may vary.

Before You Begin Programming

Organize the substructure categories!

Setup:

Setup Structure			
GLOBAL VARIABLE 'Setup'			
field names	*type/field*	*dim/type/field*	*notes/qualifiers*
.aveCount	int32		number of averages, init=8
.dirName	string		init="C:\"
.dutName	string		specimen under test name; init=""
.fileName	enum		short/medium/long; init=short
.intTime	enum		1 to ? in powers of 2; init = 8
.freqStart	float64		init=100, Hz
.freqStop	float64		init=15e6, Hz

• The program using this Setup structure manages the state of only one instrument.

• I prefer Excel for this planning task, but practically anything will work.

• Notice how we have not only defined what the variable names are prior to beginning the programming task, but we've also defined:

  • variable type

  • scalar or array (and all that subsequent baggage)

  • allowable range of values

  • units

  • initial values

  • a description if it isn't obvious from the variable name

• Working these parameters out ahead of time provides a reference during software development, and avoids making decisions in the heat of a complex logic development task. I often develop this information during the requirements interviews... this is a great way to uncover all manner of issues because it forces the end user to truly engage the questions and process.

• If the number of instruments are sufficient, and especially if there are more than one of each type, some of the structural elements might be arrays or structures in and of themselves – that is, a structure of structures and arrays.

  • You'll never get a structure which is this complex to work if you try to design it on the fly.

Before You Begin Programming

Organize the substructure categories!

Status:

Status Structure			
GLOBAL VARIABLE 'Status'			
field names	***type/field***	***dim/type/field***	***notes/qualifiers***
.calibrated	uint8		been short cal yet? Init=0
.control	string		"PROGRAM" || "LOCAL"; init="LOCAL"
.counter	int32		clear table counter; init=0
.err	uint8		if error occurred = 1; init=0
.errNo	int32		last error number; init = 99999
.errMsg	string		last error message; init = ""
.flexGridInitialized	uint8		appears in program but probably not used
.measuring	uint8		if measuring = 1; init = 0
.message	string		current operator message; init=""

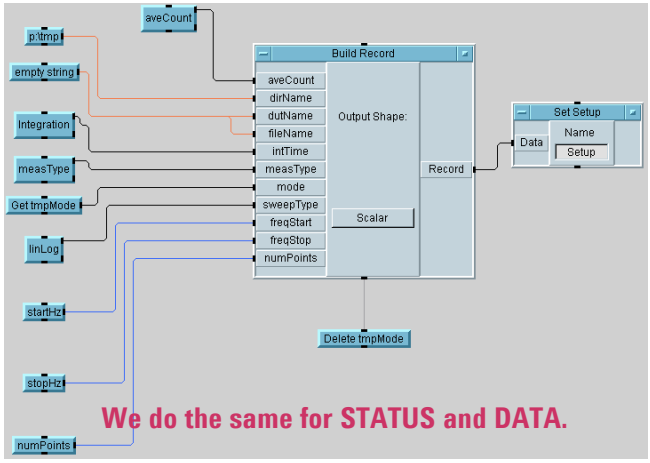
- Here's the STATUS structure
- We'll skip the Control structure

When You Begin Programming

Convert your substructure categories into VEE Records!

Benefits of using records:

- **Like types of information are logically grouped**
- **When records are implemented as VEE Variables, minimal wiring is necessary among VEE objects**
- **Easy formula access with "." notation**



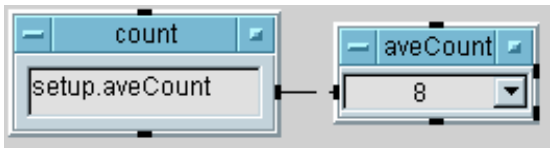
We now have a VEE global variables named SETUP, STATUS and CONTROL that are available anyplace in the program. A purist might tell you this is an extraordinarily bad thing because when there are unexpected results, you don't know where the variable was changed. I agree to a certain level.

However, in a graphical programming language using this approach minimizes the complexity of wired connections burrowing down multiple levels of functions calls, along with the associated task of creating, naming and managing terminals.

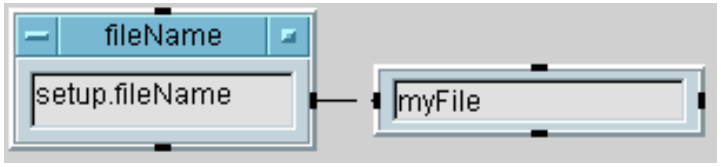
Care must still be taken when altering global variables.

Accessing Our Records

Objects on the left are FORMULAS with the RESULTS terminal hidden; on the right are CONSTANT input objects with a DEFAULT VALUE terminal added...



The Record Variables are easily accessed anyplace in the program



Labeling the objects saves lots of time now & @debug!

• There are other ways to access variable record the data in VEE records, but formulas with DOT NOTATION is the most straightforward.

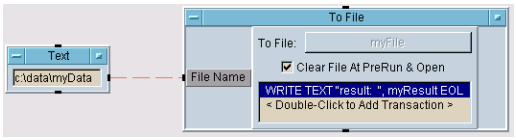
• Labeling the boxes like this does take a little time, but it saves hours of head banging later.

• Putting a name on the object is also helpful when you put a group simultaneously onto a panel – right clicking the object will tell show which parameter it is associated with.

Designing A Supportable Program Structure

Mac's Rules of Graphical Programming with VEE

1) **Never allow your program objects to occupy more than one screen**
2) **Implement carefully designed data, status and control records**
3) **Use user functions and user objects to accomplish 1)**
4) **Do not unnecessarily constrain objects, but be sure to constrain objects that require it**
5) **VEE terminal arrangements cause programs to naturally flow down and to the right. Don't fight it!**



• When program objects or functions occupy more than one screen, you haven't broken down the tasks and operations to a sufficient level of granularity. Experienced textural language programmers also try to limit functions to a page for the same reasons. Either way, it just makes the function RADICALLY easier to visualize, understand and debug.

  • I once had a customer ask me to modify a program with the main run loop that spanned 200 VEE screens (!). I looked at the code for two days and still couldn't begin to understand it. It was so complex and scattered that it's a miracle they got it to work in the first place. We started over and built the run loop in 1/3 of a screen using the VEE Sequencer Object. This thread is shown later in this presentation.

• Use userObjects to group specific functionalities if you think you'll probably not use the function in more than one place in your program (you can always change it to a userFunction later.)

• Use userFunctions if

  • you will use the functionality in more than one place in the program

  • you need to programmatically choose which function to call

  • you need to update the contents of a function panel at will

  • it has a panel – an operator interface of any type

• Be sure to name your objects and functions in a very consistent and logical way.

• ALWAYS be sure to constrain objects with only an ASYNCHRONOUS terminal on the left (input) side.

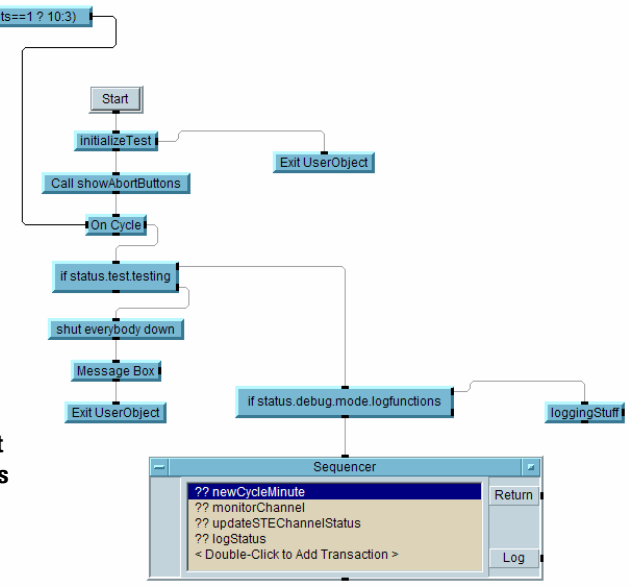
  • these are easy to recognize -- they have a dashed line connecting them to the upstream object

Designing A Serviceable Program Structure

Mac's Rules of Graphical Programming With VEE

6) **Use showPanel() and hidePanel() to control when, where and how large you want your function panels to show**

7) **Use the Sequencer object in conjunction with status variables to control complicated execution requirements**



• Implementing the object property SHOW ON EXECUTE is OK for beginners or for quick and dirty stuff. ShowPanel and HidePanel allow the programmer to exert a great degree of control on the user interface.

  • SHOW ON EXECUTE doesn't belong in a serious program because it severely limits the programmer's control over the user interface panel

Some Thoughts On the Sequencer

• For some reason VEE users either generally loath or are just frightened senseless over the Sequencer

  • Perhaps it's because we used to pummel our training class students with the Sequencer Lab in the Intro to VEE class (this has since been significantly improved!)

• The Sequencer is the single most powerful object of all the VEE intrinsics

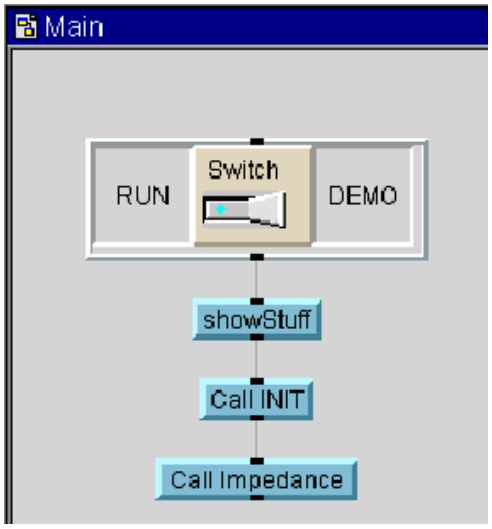
  • Learn to use the Sequencer along with a solid data and control substructure, and you'll find you will NEVER need to make complex execution decisions with if/then objects and spider webs of constraint connections

  • The thread above is the one that replaced 200 pages of spider webs for a run loop

• **The Sequencer is your friend!**

A Typical Program MAIN

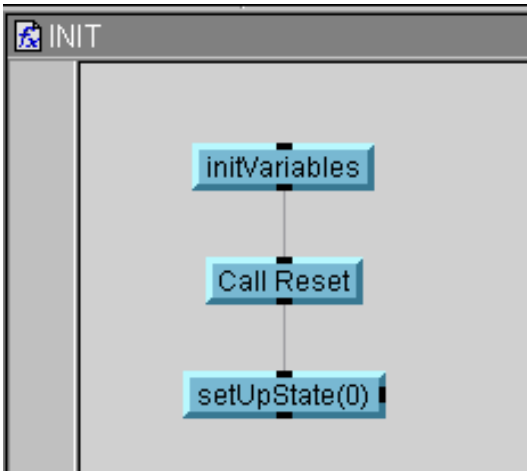
Is this simple enough for a MAIN?



- Run / Demo control is a user object with a panel

INIT Function Detail View

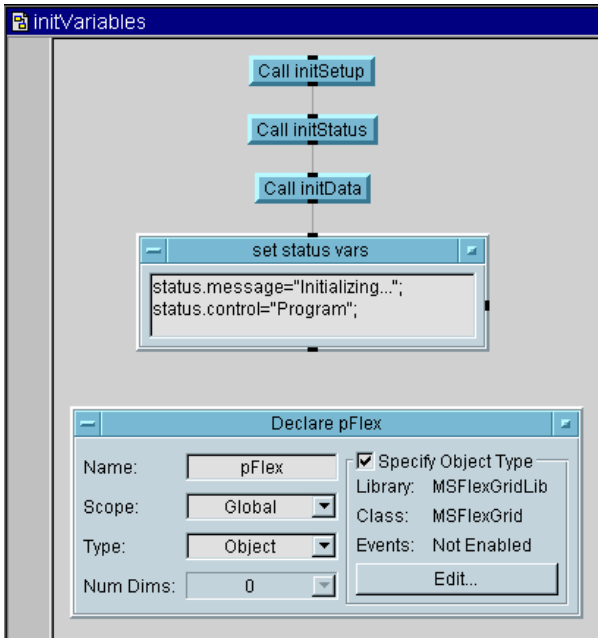
Is this function too simple?



- There is no programming wisdom that forces a function to be complicated
  - The function need only serve a clear purpose

userObject initVariables Detail View

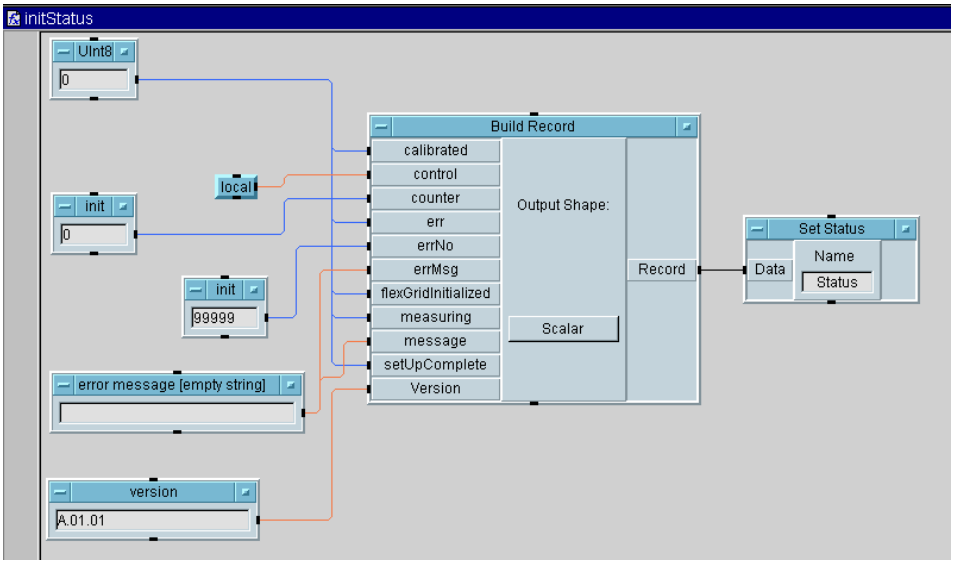
- **This program uses the MS ActiveX control msFlexGrid so a variable must be declared to access and control its functionality**



Initialize your variables and records at the beginning of your program.

Function initStatus Detail View

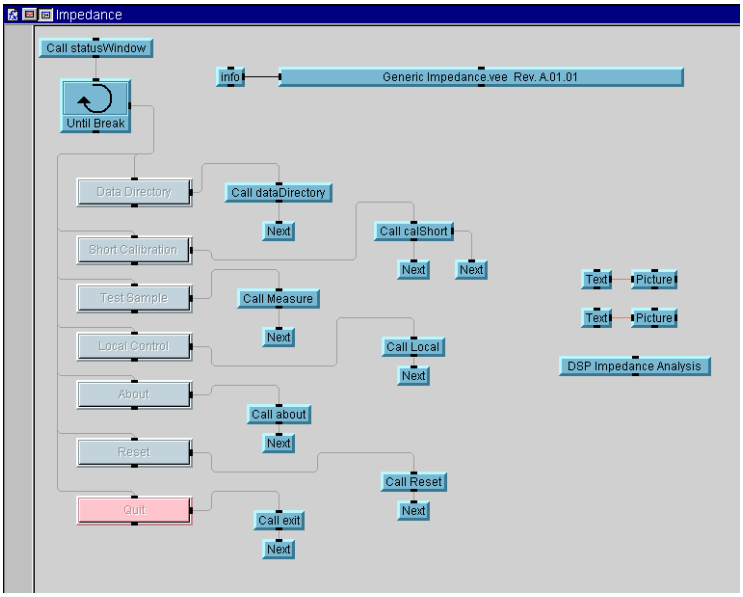
- **initSetup appeared around slide 9**



- Note how the terminal names are in alphabetical order
  - This wasn't an accident – it just makes the RECORD object easier to troubleshoot.
  - Sometimes I'll put a terminal like VERSION at the top if it makes life easier.
  - In any case, order or grouping has no bearing on execution or memory efficiency.
- Experienced programmers will normally initialize DATA variables to '-9999' or something that is clearly bogus
  - It helps to spot problems very quickly
- We'll skip inspection of the initData module

Function Impedance Detail View

- **Note use of UNTIL BREAK / OK construct**
- **This is <u>NOT</u> a *RUNAWAY LOOP***

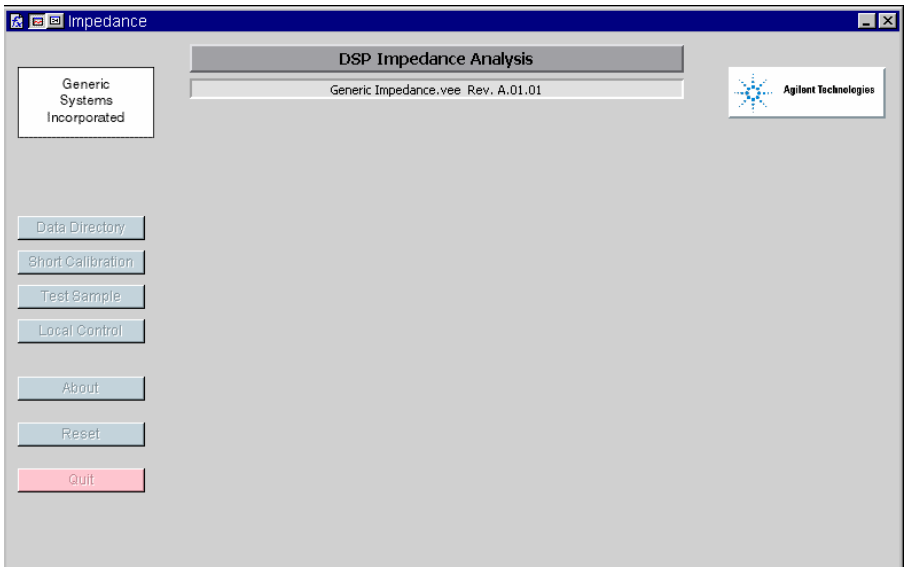


- Function Impedance is the heart of this program
  - It's the first panel the operator sees when the program starts
  - It is where the program is exited
  - All functionality returns to Impedance upon completion of a task

- The UNTIL BREAK / OK construct used in Impedance is a very efficient way to operate your program
  - This construct uses ZERO PERCENT of CPU while it is waiting for an operator to select the next step
  - This makes both the program and computer responsive to the operator
  - VEE Toggle objects with a RESET control pin behave much differently

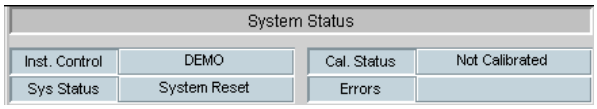
- NEVER use an open or unconstrained UNTIL BREAK construct to build a runaway loop unless you have a specific reason to do so!
  - It will use 100% of the CPU cycles for the fastest computer you can find
  - It will cause the program and the entire computer to be unresponsive because the computer is servicing a runaway loop
  - Adding more CPU power just wastes cycles faster
  - This is the #2 problem I find in customer software (right after functions that take up 200 pages {no kidding....}]
  - Use events, interrupts, ON CYCLE objects, or SRQs instead

Function Impedance Panel View

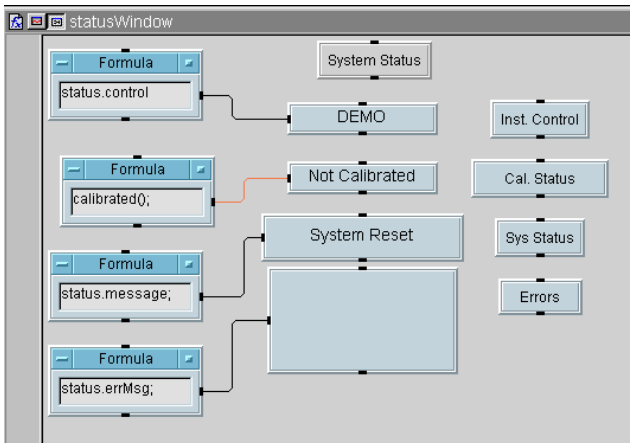


- This program uses panels dynamically placed upon panels
  - In this program the Impedance function panel is what I would call the MAIN or ROOT panel because it's what all action and control returns to
  - The STATUS and SETUP STATE functionalities shown on page 4 are separate panels placed on IMPEDANCE at run time

Function statusWindow Panel & Detail Views

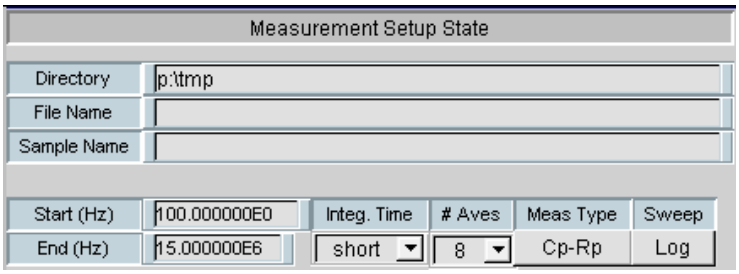


- **Why no inputs or controls?**



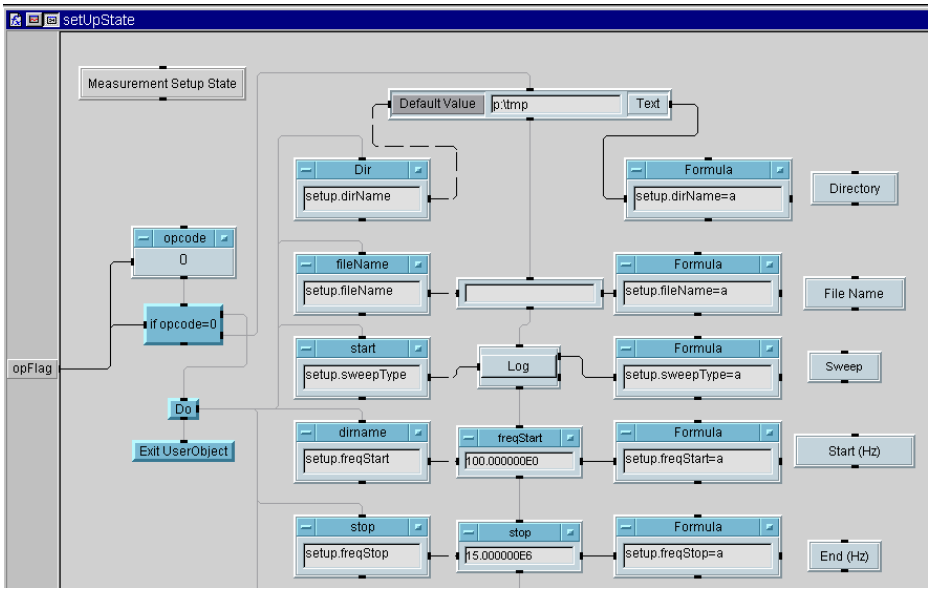
- This is the first of two sub panels shown on the main view
- The function has no controls – it consists solely of indicators
    - It is shown via the showPanel() function
- Calling this function (it has no arguments) merely updates the status fields,
    - This makes it a simple task to update the panel information whenever it is required
    - If this status view did not exist as a separate function, it would be very difficult to update it at will
- The reason for a status panel?
    - Measurements can take a long time.
    - Studies have shown that a typical computer user will wait about 5 seconds for something to happen before they start pounding keys, racing the mouse, or generally get impatient
    - You can expect this time interval to shrink as computers and networks get faster while our expectations become greater
    - A constantly updating status screen assures the impatient that, indeed, the request is being processed

Function setupState Panel View



- This is the second sub panel on the main view
- All operator inputs are found on this panel

Function setUpState Detail View (partial)

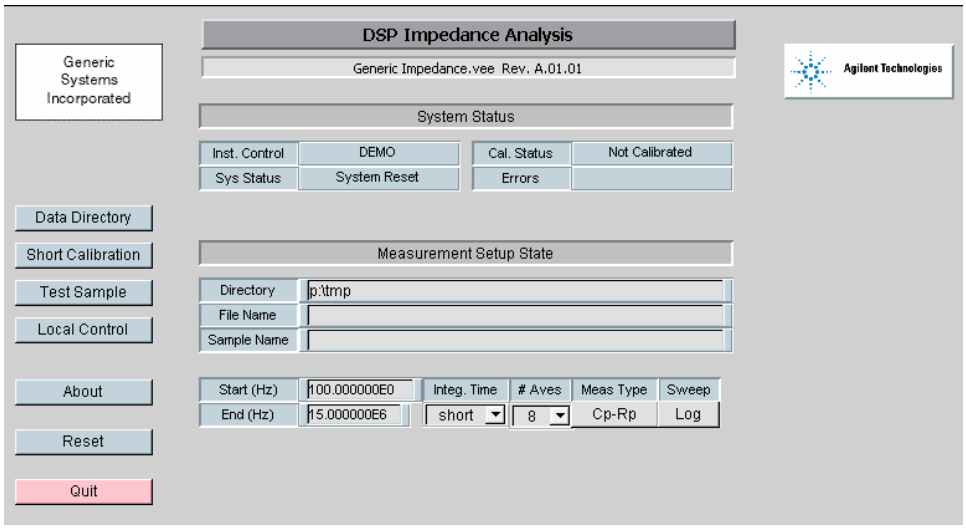


Details of function setUpState:

- The Two Corollaries to Mac's Rules of Graphical Programming Rule (1) are:
  a) If you are going to use more than one screen, have a really good reason for it
  b) If you must use more than one screen, be sure to go DOWN the page! (because you can use the page up/down controls on your keyboard to navigate easily)
- Function setUpState has one argument, opFlag.
  - Entering the function with opFlag = 0 (as done in function Init) causes the record variables in the second column to be read into the DEFAULT VALUE pins of the various constants in the third column.
    - The information shows up immediately on the panel view, since it has already been placed with a ShowPanel call
    - No other action takes place, since the DEFAULT VALUE pins added to all the CONSTANT objects in column three are VEE ASYNCHRONOUS INPUTS, the objects update, but do not operate
    - Note the dashed line connecting column two objects to column three objects -- this indicates a connection to ASYNCHRONOUS INPUTS
    - Thus the result of calling with opFlag = 0 is that the user interface (panel) is prepopulated with the default setup parameters.
  - After the operator is satisfied with the parameters, the function is called with opFlag = 1, and the third and fourth columns execute, setting the record variables in the fourth column to their new values

This technique is a useful way to get data from a default or recalled state, modify it, and stuff it back into the program variables.

Program Runtime View – Initial Screen



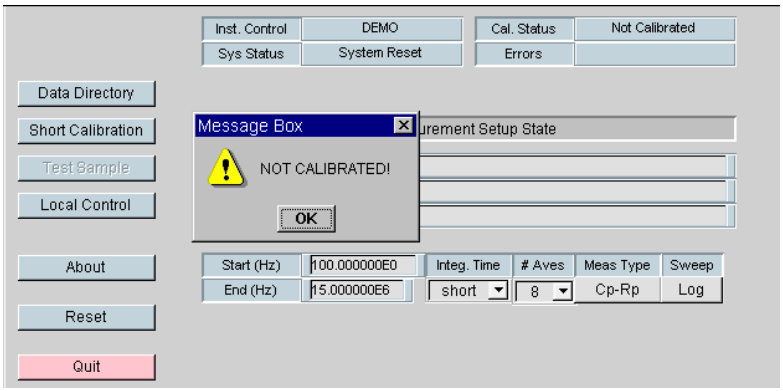
Some user interface philosophies I embrace:

- Color should be used sparingly, and always have a specific purpose in any user interface
- Studies have shown that radical colors disturb operators
    - Pastels are generally preferred; it's OK to use RED or GREEN, etc., but use it sparingly, and for a meaningful purpose!
- In this program
    - Blue pastel coloring is used for operational cues
    - Brighter pastels are used for warning indicators or operations that might otherwise cause the program to stop doing what it is doing
- Note that the navigational controls are all located in one place
    - If there were auxiliary controls they would be on the right, if possible.
    - The bottom of this panel has been clipped off to show more detail – thus the lack of balance.
- Note also that
    - Operator inputs are colored light gray
    - Operator input labels are in blue sunken boxes
    - Status information appears in blue flat boxes
    - Input groups are labeled with medium gray sunken boxes
    - The only color that is not completely quiet is the QUIT control – by design

- Note also that one would usually proceed from Data Directory down the page to Quit for normal operation -- operators find this to be a fairly intuitive arrangement
- There are three different panels here, so it is necessary to select Lock Panel Position in the function PROPERTIES menus to keep the Impedance panel from occluding the other two function panels

Program Operation

After the operator selects a target file for the data to be written to, he must calibrate prior to executing a test.



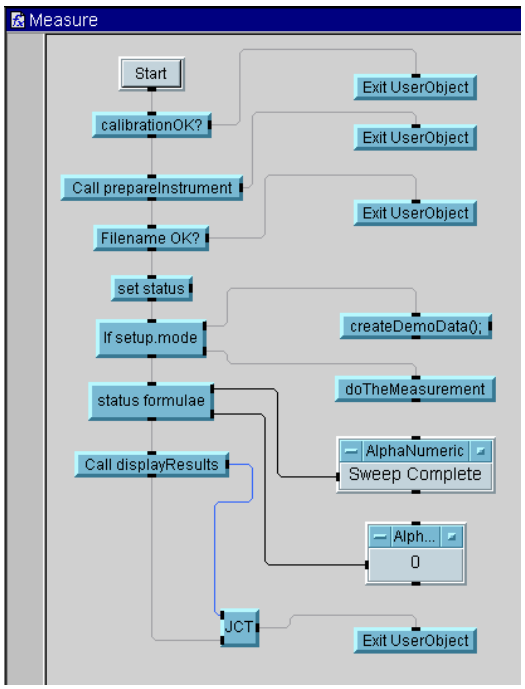
Pressing the TEST SAMPLE control on the IMPEDANCE calls function MEASURE.

The first thing that MEASURE does is check the calibration status.

- Calibration status is checked via variable status.calibrated
    - The operator is forced to calibrate prior to running test
    - Control is returned to function Impedance if the operator tries to test without calibrating

After calibrating, pressing TEST SAMPLE again causes function Measure to execute

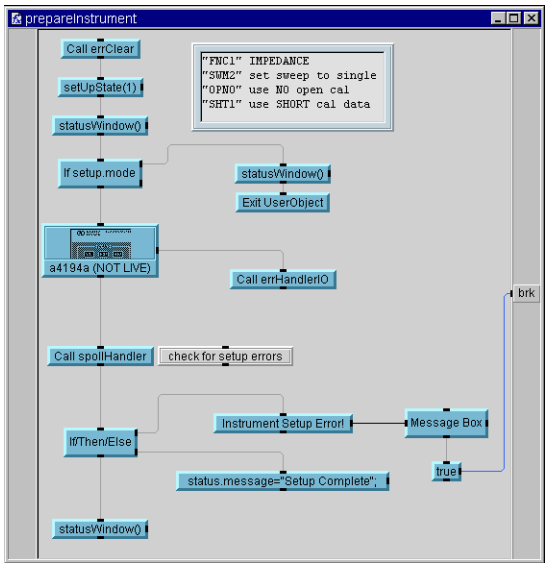
Function Measure



Function Measure

- Has no panel
- checks calibration status
- sets up instrument with function prepareInstrument
- checks file name status
- calls object doTheMeasurement
- calls function displayResults

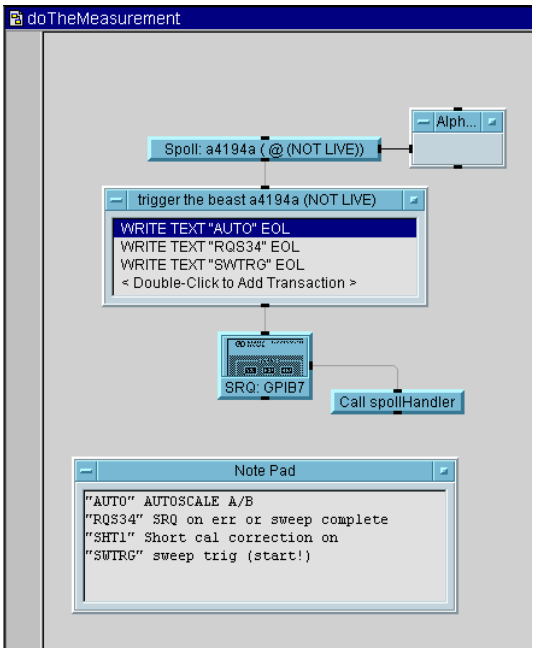
Function prepareInstrument



Function prepareInstrument

- The function begins with a call to errClear
  - This is a general module used elsewhere
  - clears any existing instrument errors with an SPOLL
  - Re-sets the instrument to interrupt (SRQ) on error
    - The SRQ interrupt feature is not actually used in this version
- The function calls setUpState(1)
  - Reads operator supplied measurement options shown on slide 21
- The function updates statusWindow
- The function sends the ASCII set up commands to the instrument in the a4194a object
  - The output pin on the right side of the a4194a object is the ERROR pin
  - On ERROR condition from I/O box, calls errHandlerIO
    - Most likely cause of error here would be timeout or instrument power off
- The function calls spollHandler to see if the ERROR bit is set in the instrument
  - Most likely cause of error here would be a bad command or combination thereof
- statusWindow is refreshed

Object doTheMeasurement



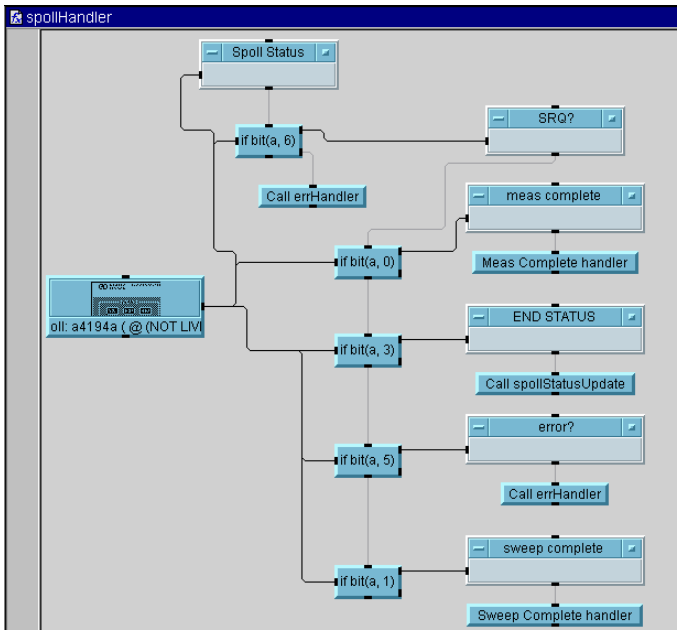
UserObject doTheMeasurement

- performs an SPOLL to clear the status register bit 6 (SRQ)
- triggers the measurement
- Waits for interrupt on measurement done
- Calls function spollHandler

- Using an interface event (SRQ) to trigger the measurement complete actions is very efficient
  - You can have multiple instruments performing simultaneous tasks
    - As each task completes, the instruments pull SRQ
      - service routine determines which instrument and action to take
  - There is no need to use a WAIT or DELAY function in **ANY PROGRAM** except for debug purposes – always use SRQs or delayed poling with an ON CYCLE object.
  - Allows operator interface to be active during measurement
    - Permits operator to abort a long measurement
    - Allows countdown or status update while measurement is active
    - Makes life more fulfilling for the operator (ahem...)
- You usually don't get this kind of functionality with a canned driver
  - That's why Direct I/O in VEE is so powerful

Notice how a NOTE PAD object is used to keep track of the proper commands? This is a valuable aid when returning to enhance or debug the software after even a couple of days have passed.

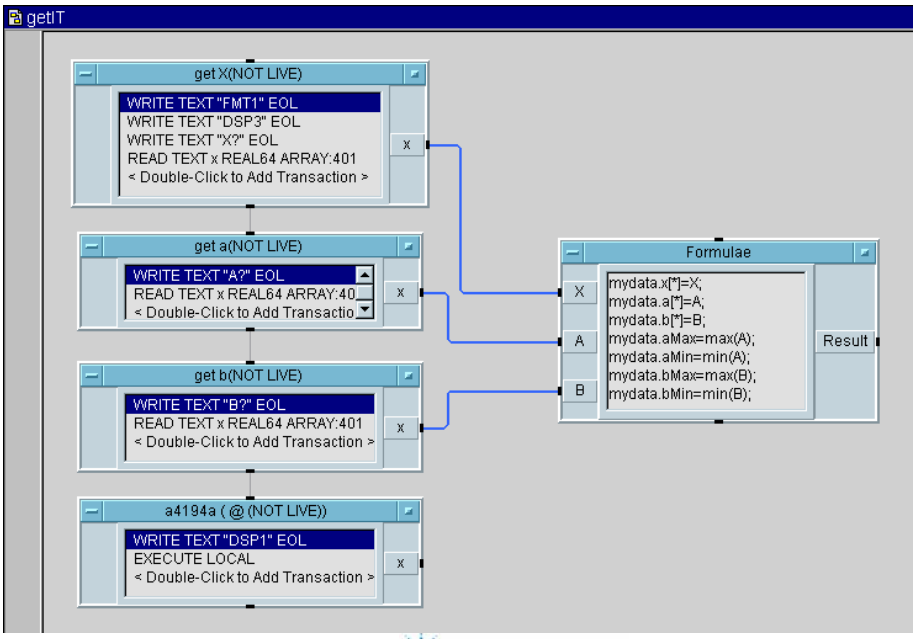
Function spollHandler



Function spollHandler

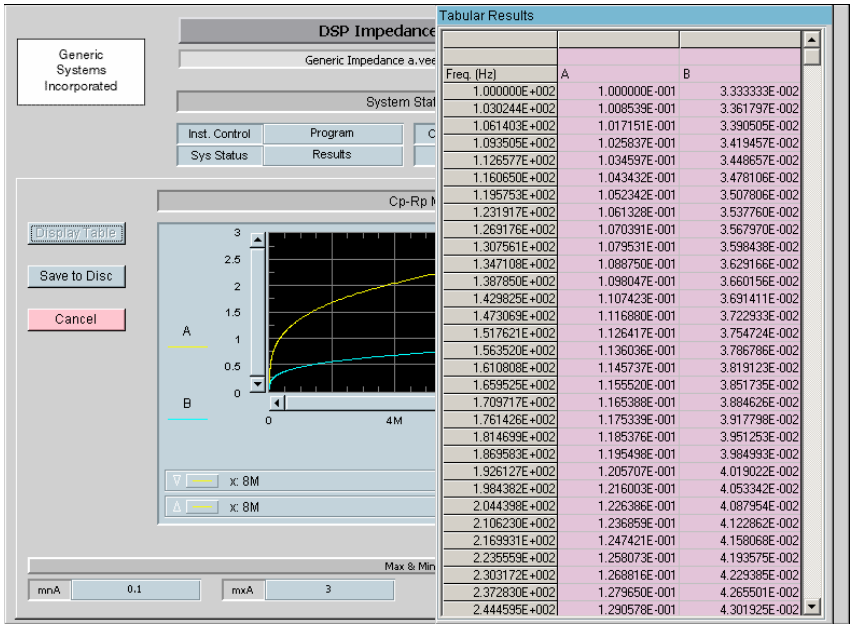
- If more than one instrument is present, program must determine which instrument called for service
- Finds out why interrupt occurred
    - makes sure data is ok and no errors
    - returns to Impedance with messages if errors are caught
- Eventually calls object getIt, which reads the data back to the program
- Calls functions to plot, show table, save data, etc.

Object getIt



UserObject getIt reads data directly into structure variable myData using formulas and dot notation.

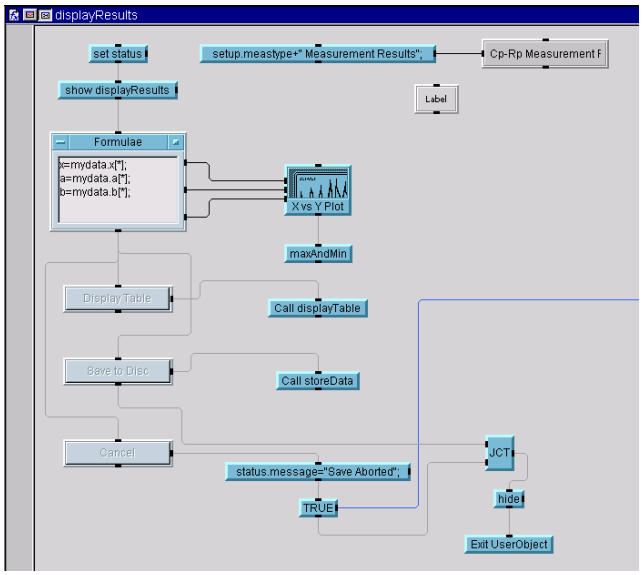
Function displayResults - Panel View



Function displayResults – Panel View

- We've now returned to function Measure
- Measure calls function displayResults
  - Displays data in Table form if desired via MSFLEXGRID ActiveX control
  - Saves Data to disc if desired in tab separated variable format (TSV)
    - TSV data is simple to import into Excel
    - Direct placement of data into worksheet in VEE 7.5 new Excel objects, or via ActiveX Automation is straightforward, but requires Excel on host computer
      - The VEE to Excel object technique is taught in the Intro VEE training class
      - ActiveX Automation and .NET control techniques are taught in the Advanced VEE training class
  - Returns to Impedance

Function displayResults – Detail View



- This structure type should be looking pretty familiar by now!
- This ends our dissection of the program structure

A Couple of Common Sense Suggestions

Back Up Strategy

Mac's 4 Rules of Backup

Do it

- **Frequently**
- **Often**
- **Regularly**
- **To more than one physical location**

• I REALLY LOATH having to do any task twice

  • I have the scars to show that I truly came upon this wisdom the hard way

VEE crashes during development. So do LabView and Visual Studio, because it's all software running in Windows. Go figure. Even Rocky Mountain Basic had its bad days. If you've not experienced this phenomenon, you probably don't do much programming

• It is thus essential to have some dependable system of keeping track of what's what if you are going to squirrel backups away in lots of places

  • For my program name I generally use a short, project related prefix that never changes, followed by

  • an integer that increments every day I work on the program, followed by

  • a letter that indicates each time I make a logic leap that could break something that is already working.

• I have anywhere between 1 and a couple dozen daily incremental backups that never leave my development system

• I back up the end of every day to a system in another state, or, if in a hurry, I zip my program and email it to myself at work and at home

• This kind of paranoia has saved my bacon on countless occasions

• How much is your time worth? Your reputation? Dependable people don't lose stuff!

A Couple of Common Sense Suggestions

Revision Control

Never let a document or software module out of your control without embedding some form of unambiguous revision control & date information:

- **Functions or libs – embedded in name & internally**
- **Programs**
  - **in status structure (shown on initial view)**
  - **Internally**
  - **possibly in name**
- **Documents – in the properties, on the front & every page**

When I am discussing or comparing a program or document with a customer, I always ensure that, literally, we are on the same page.

In Conclusion – Implementing a Supportable VEE Architecture

- Have firm, detailed & unambiguous requirements before you begin *any* software development.
- Take time to plan a detailed program substructure & architecture; stop to update your underlying planning documents as the program changes, while the details are still fresh.
- Have a disciplined approach to building, releasing and supporting your software – those outlined in this presentation will do, but there are many other approaches.

Look for Application Notes And Other Information On Our Website

LXI – LAN Extensions for Instruments:
www.agilent.com/find/lxi

Agilent VEE Pro:
www.agilent.com/find/VEE

Agilent Instrument Drivers:
www.agilent.com/find/drivers

Agilent I/O Technology, Agilent Connection Suite 14.1:
www.agilent.com/find/io

Test System Developer Guides:
www.agilent.com/find/system

(Look for "Test System Developers Guides")

About The Author

PC (Mac) MacDonald, a Test Automation Systems Engineer at Agilent Technologies, holds an engineering degree from Purdue University.

Mac has been automating test applications on various program development platforms for decades, and, among other things, is currently an instructor for Agilent's VEE Pro & PNA/COM/Visual Basic Customer Education classes.

You can contact Mac via e-mail at

pcmacd@agilent.com | pcmacd@gmail.com

or by telephone at 714/935-5425

Printed in USA, March 2006
5989-4914EN

www.agilent.com

Agilent Technologies' Test and Measurement Support, Services, and Assistance
Agilent Technologies aims to maximize the value you receive, while minimizing your risk and problems. We strive to ensure that you get the test and measurement capabilities you paid for and obtain the support you need. Our extensive support resources and services can help you choose the right Agilent products for your applications and apply them successfully. Every instrument and system we sell has a global warranty. Two concepts underlie Agilent's overall support policy: "Our Promise" and "Your Advantage."

Our Promise
Our Promise means your Agilent test and measurement equipment will meet its advertised performance and functionality. When you are choosing new equipment, we will help you with product information, including realistic performance specifications and practical recommendations from experienced test engineers. When you receive your new Agilent equipment, we can help verify that it works properly and help with initial product operation.

Your Advantage
Your Advantage means that Agilent offers a wide range of additional expert test and measurement services, which you can purchase according to your unique technical and business needs. Solve problems efficiently and gain a competitive edge by contracting with us for calibration, extra-cost upgrades, out-of-warranty repairs, and on-site education and training, as well as design, system integration, project management, and other professional engineering services. Experienced Agilent engineers and technicians worldwide can help you maximize your productivity, optimize the return on investment of your Agilent instruments and systems, and obtain dependable measurement accuracy for the life of those products.

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

Agilent Direct

www.agilent.com/find/agilentdirect
Quickly choose and use your test equipment solutions with confidence.

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:

www.agilent.com/find/contactus

Phone or Fax
United States:
(tel) 800 829 4444
(fax) 800 829 4433

Canada:
(tel) 877 894 4414
(fax) 800 746 4866

China:
(tel) 800 810 0189
(fax) 800 820 2816

Europe:
(tel) 31 20 547 2111

Japan:
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

Korea:
(tel) (080) 769 0800
(fax) (080) 769 0900

Latin America:
(tel) (305) 269 7500

Taiwan:
(tel) 0800 047 866
(fax) 0800 286 331

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6755 0042
Email: tm_ap@agilent.com

Printed in USA, March 8, 2006
5989-4914EN

Agilent Technologies