VEE 4.0 Introduction Notes (mar 97)

last update 28 aug 98 / greg goebel / public domain / vwv_4_0

* This document provides introduction notes on the VEE 4.0 release in the spring of 1997.

* Contents:

[1] OVERVIEW

* HP VEE 4.0 is an important release that provides a jump in functionality greater than any other previous version of VEE. Major new features in VEE 4.0 include:

- No more VEE-RunOnly: You can now create VEE programs and distribute them along with a run-time DLL to as many users as you like with no further licensing requirements. The distribution CD contains a subdirectory named "runtime" with four disk-image subdirectories ... you can copy these four subdirectories to four floppies to create your own run-time distribution floppies.

- Better performance: VEE has been internally redesigned to provide better performance than on previous versions. Since this new internal organization has slightly different behavior than the old scheme that could potentially cause anomalous operation of some old programs (Beta testing has shown such problems to be minimal), a "compatibility mode" is available to allow running such programs at their traditional speed.

- New user interface: The VEE user interface has been almost completely redesigned to provide a multiple-window scheme for handling the main program and subprograms independently. A tree-oriented browsing system allows a user to select subprograms at will at any level of nesting. A large number of other new editing and debugging features make program development -- particularly of large programs.

  Other tools include a Find utility to allow searching a program for a particular item; a Profiler to allow examination of the execution load of various components of a program; a Call Stack to track the execution of programs through UserFunctions; and an Instrument Manager that consolidates all I/O configuration in a neat and consistent fashion.

  Minor improvements include the consolidation of all math functions into a common dialogue;

better debugging facilities; wiring that changes color depending on the data type it handles; object alignment tools; and many other small but convenient changes.

- An assortment of bug fixes and other enhancements have also been provided ... important changes include Local Variables; an enhanced Formula box that provides multiple outputs, multiple comma-separated formulas, and the ability to assign value to array elements; new system information functions to allow reading environment variables and so on; and a few other minor functions.

Note that VEE's list price has been increased by about 25% to make up for unlimited Run-only capability ... even with this, however, VEE is a better value than the competition.

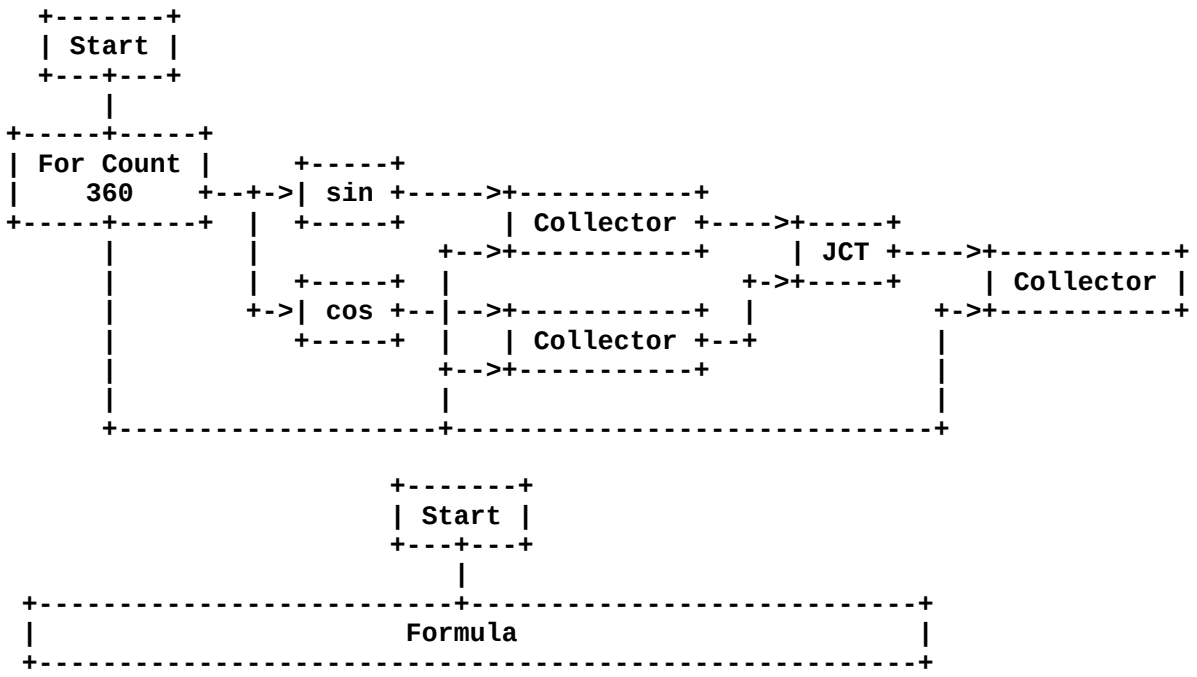
The rest of this document explains the details of the new VEE improvements.

[2] PERFORMANCE IMPROVEMENTS

* VEE 4.0 provides substantial performance increases over VEE 3.2X by virtue of supporting "compiled" operation ... which isn't necessarily what it sounds like: it doesn't mean that VEE now creates a stand-alone executable program -- it means that the elements of the VEE program are interpreted into an intermediate form ("p-code") that can be interpreted in a fast fashion. As far as stand-alone programs go, VEE 4.0 still requires a run-only environment (though you now get this for free and can distribute it as you please).

This new "compiled" mode does come at a price, in that some operations that were allowed on VEE 3.2X are now forbidden and give you an error message (more on this below). However, VEE 4.0 also supports a "VEE 3 Compatibility" mode to allow you to run old programs that have problems in this regard until they can be tweaked accordingly.

Having both compiled and compatibility mode does make it easier to demonstrate VEE 4.0's improvements in performance, however ... consider the examples below:



```
| [ sin(ramp( 360, 0, 359))  cos(ramp( 360, 0, 359 )) ] |
+-------------------------------------------------------+
```

These two examples generate an array containing all the values of sine and cosine for each degree from 0 to 360; they were originally devised to show the difference in VEE speed for iterative instead of array operations, but they also serve well to show off VEE 4.0's performance. Experiments with the two programs give the relative performance:

	iterative	array
compatibility mode:	1	22
compiled mode:	8	215

The figures are the increase in speed relative to the slowest operation (the iterative program under compatibility mode). These figures indicate a roughly order-of-magnitude increase in performance of these two sample programs.

Be warned that overall system performance isn't going to make a jump of this magnitude (don't worry -- you won't hear anything about "VEE being faster than C!") when you factor in display and file I/O ... you can likely expect an improvement in performance of about 3 or 4 times -- which brings VEE applications up to a performance level comparable to those of any other applications in the same domain on a PC.

In general, looping constructs and formulas will show the greatest speed increases; I/O-bound operations like file-I/O or graphics will show the least.

* OK ... so what does compilation cost you? First, crossed loops don't work any more -- VEE 4.0 flags them as an error and won't run them. Second, any time you have feedback in a VEE program or function, you *must* do it through a junction box.

* VEE 4.0's compiled mode also offers one other important benefit: In VEE 3.2X, if you executed a UserFunction, no other thread in VEE would execute in parallel with it ... that is, its operation was "atomic". VEE 4.0 now allows such timeslicing (in compiled mode only).

This leads to a related question: what happens if you call the *same* UserFunction in different threads under VEE 4.0? In earlier versions of VEE, UserFunctions were not "reentrant": you couldn't call a UserFunction from the same UserFunction.

Under VEE 4.0, UserFunctions are *still* not reentrant. This means that if you call a UserFunction in one thread and then try to call it again from a parallel thread, the second call will "block" (not execute) until the first call is complete.

However, if you have parallel paths in the same program they will not operate in an alternating fashion any longer.

[3] NEW DEVELOPMENT ENVIRONMENT -- MENUS

* A good way to start is to go through the menu selections and see what has changed.

* The "File" menu incorporates a few small changes:

- A "Save Secured Runtime Version" entry that allows you to save your VEE program in a secured ".vxe" format that can be executed in runtime mode; this file will be saved separately from the original file, ensuring you don't secure your source by mistake. The old "Secure" entry is gone.
- Modifications to the "Edit Default Preferences" dialogue box to select between compiled/VEE-3-compatibility modes, spacing for object alignment in the development environment, and enhancements to print capability.
- The "Print All" entry has been changed to "Print Program", with a few small enhancements in print options.
- Programs previously accessed by VEE now have their names queued up at the bottom of the menu to allow them to be retrieved (a common feature in such contemporary applications as Excel and the like, but new to VEE).

* The "Edit" menu has a similar level of modifications:

- "Clone" is gone ... "Copy" & "Paste" do the job just as well.
- "Select Objects" and "Move Objects" are gone as well; a new "Select All" and other features replace this functionality.
- The new "Find" entry brings up a dialogue that allows you to search for program elements.

* VEE 4.0 adds a new "View" menu:

```
  View  Debug  Flow
-+-------------------+-
 |  Variables...     |  Display list of variables for examination.
 |  Last Error       |  Display last error message that occurred.
 +-------------------+
 |  Main      Ctrl+M |  Select main program panel in MDI interface.
 |  Execution Window |
 +-------------------+
 | *Program Explorer |  Show/hide Program Explorer tree.
 |  Call Stack       |  Bring up UserFunction stack tracer.
 |  Profiler         |  Bring up program execution profiler.
 +-------------------+
 | *Toolbar          |  Show/hide toolbar at top of environment.
 | *Status Bar       |  Show/hide status bar at bottom of environment.
 +-------------------+
```

The "Debug" menu has been modified slightly:

- "Run" and "Resume" have become a single entry, "Run/Resume".
- The "Step" entry has been expanded to "Step Into", "Step Over", and "Step Out" to support the more flexible stepping capabilities in VEE 4.0. "Step Into" allows you to single-step through each object in the program; "Step Over" allows you to skip over UserFunctions and UserObjects while stepping through a program; and "Step Out Of" allows you to jump out of

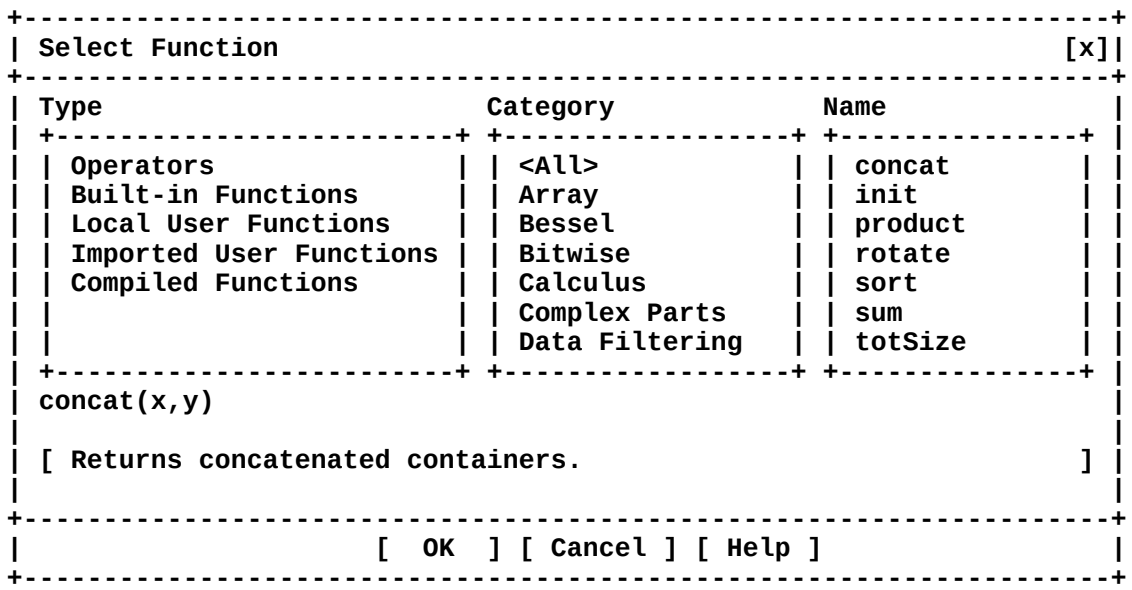
stepping in a UserObject or UserFunction and return to the calling context.

- "View Globals" has been deleted (it is now essentially part of the "View" menu).
- An entry for "Object Probe" allows you to determine what objects are wired to a specific object.

The "Flow" menu remains the same. The "Device" menu has been modified slightly to reflect the fact that the "Math" and "Advanced Math" menus have been deleted to support a much more convenient way of handling functions:

- The main change has been to add the "Formula" entry and a "Math & Functions" entry to bring up a dialogue box that allows access to all VEE functions (see below).
- "Regression" has been added and the random number functions eliminated from the menu.
- Cascade menus for "Call" and "Panel" have been eliminated, with some functions incorporated into the menu itself and others added to the "Math & Functions" dialogue.

The "Math & Functions" dialogue box provides a convenient way of consolidating all the earlier clutter of the math operations in VEE (as well as integrates handling of UserFunctions in the program):



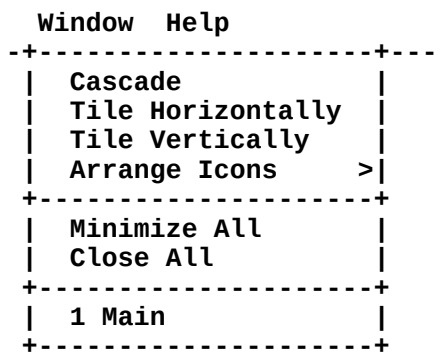
This item is available as the "fx" button on the Toolbar.

* The I/O Menu deletes the separate VEE driver and Plug&Play Driver configuration entries and uses the Instrument Manager instead (more on this later).

* The Data Menu is only slightly changed -- the "Globals" entry now becomes the "Variables" entry (since VEE 4.0 now has Local Variables as well), and the "System Information" functions are incorporated into the Math&Functions dialogue.

* As noted, both the Math and Advanced Math menus have been deleted. The Displays menu

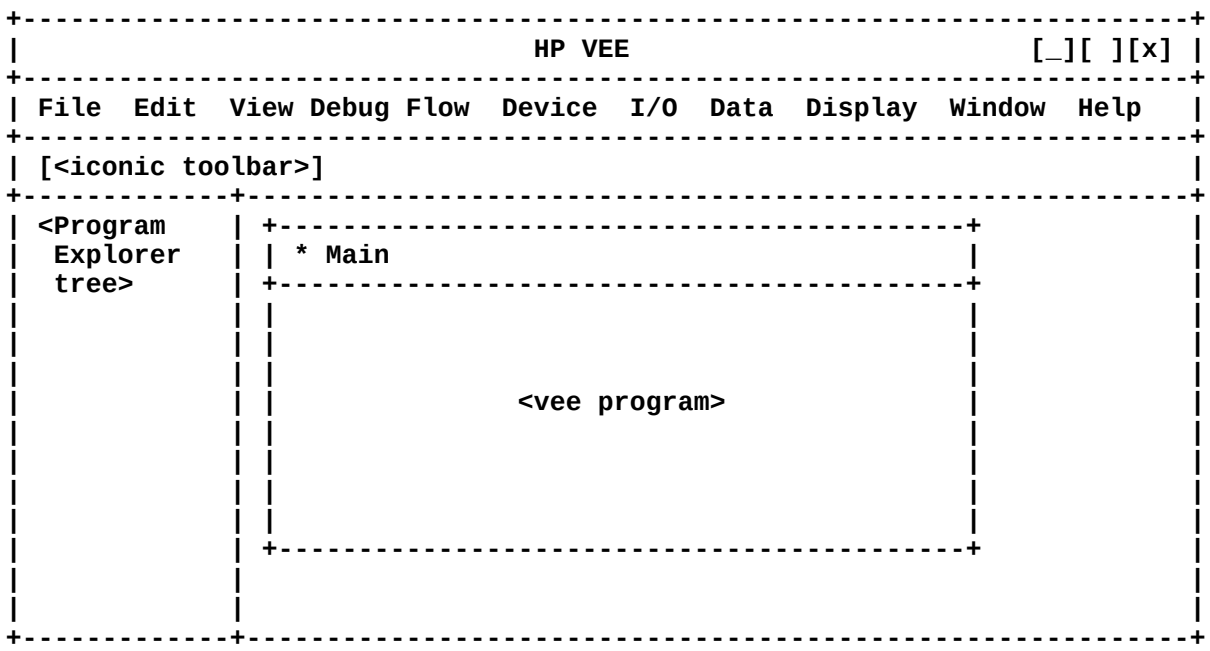
remains the same, as does Help ... but (to support the new multiple-window interface) a new Window menu has been added:



This should be familiar to anyone who used the similar menus on Win3 or Excel or whatever.

[4] NEW DEVELOPMENT ENVIRONMENT -- MDI INTERFACE, EXPLORER

* The changes in the VEE 4.0 development environment are immediately visible; the new environment looks like this:



The obvious changes are the "Program Explorer" tree at left (which looks something like Windows Explorer but allows you to traverse through the functions and features of your VEE application rather than files and directories), and the fact that the editing workspace itself appears as a "Main" panel.

In VEE 4.0, all UserObjects and UserFunctions are now accessed as separate windows, using the so-called "multiple document interface" (MDI) scheme; they can be "iconized" and will show up as

text icons at the bottom of the work area. The separate window scheme makes editing of UserFunctions and UserObjects a much simpler operation.

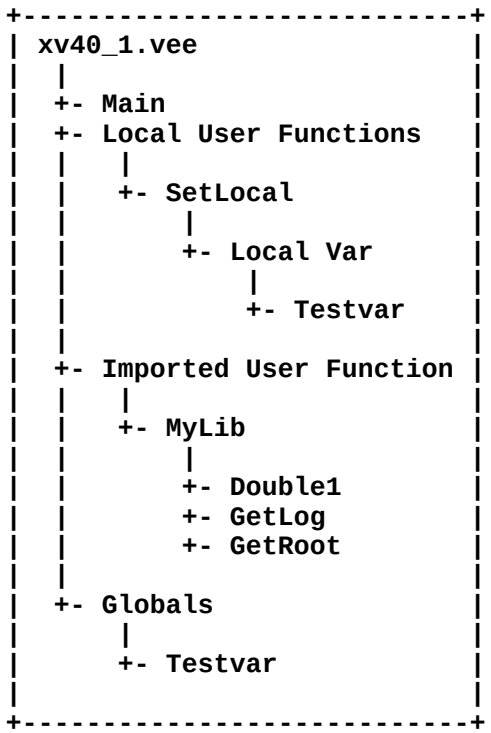
You can cycle between windows in the MDI user interface with the Ctrl+Tab key combination, and cycle backwards with the Ctrl+Shift+Tab combination. The new "View" menu allows you to keep all the windows and icons in the workspace organized -- allowing you to minimize or close them, arrange the windows in a tiled or cascaded fashion, or select one of the windows from the menu. Note that when you save your program, the current state of the workspace -- that is, all the windows and their positions -- is saved so it can be restored when you return to VEE.

If your program has a Panel View, when you run the program the Panel View will also appear as a separate window -- the "Execution Window". This means that you no longer have separate VEE operational modes ... it looks the same at runtime as it does during development.

This means that if you *don't* want users to have access to the VEE development enviroment, you must run the program in a VEE RunOnly mode. This actually doesn't require much effort; if you perform a "Save Secured RunTime Version" on a program, it is saved as a ".vxe" file; executing the .vxe file from Windows Explorer will run it in its own execution-time window. (The .vxe file is stored separately from the original .vee file, so as an added benefit you can't secure your source file by mistake ... which is good because you *can't* get it back if you do.)

By default, installing VEE will set up an association between the VEE RunOnly bits and the .VXE file extension, so simply selecting a .VXE file will run that program. This makes VEE programs look very much like executable programs when the computer is properly configured.

* Program Explorer allows you to access and probe the elements of the program in a structured fashion; here's an example demonstrating the contents of Program Explorer in a sample application:



This example shows "Main" (the core VEE program, which is a constant); User Functions in the programs and the variables associated with them; User Functions from User Libraries and their variables; and Global variables associated with the entire program.

In general, the Program Explorer list the following resources in its tree, designated by distinct icons:

- The VEE Main program (at the top of the tree).
- Local UserFunctions.
- Imported UserFunction libraries and their constituent UserFunctions.
- Imported Compiled libraries and their constituent compiled functions.
- Imported Remote UserFunction libraries & their constituent UserFunctions.
- Declared and undeclared global variables.
- Local variables (under their appropriate context).
- Library variables (under their appropriate context).

Double-clicking on an entry in the Program Explorer brings the corresponding window into the workspace (for a UserFunction or the like) or the value (for a variable). Clicking on the secondary mouse button on an entry brings up a menu that allows you to examine the selection in more detail.

For a, say, UserFunction, you get a menu like:

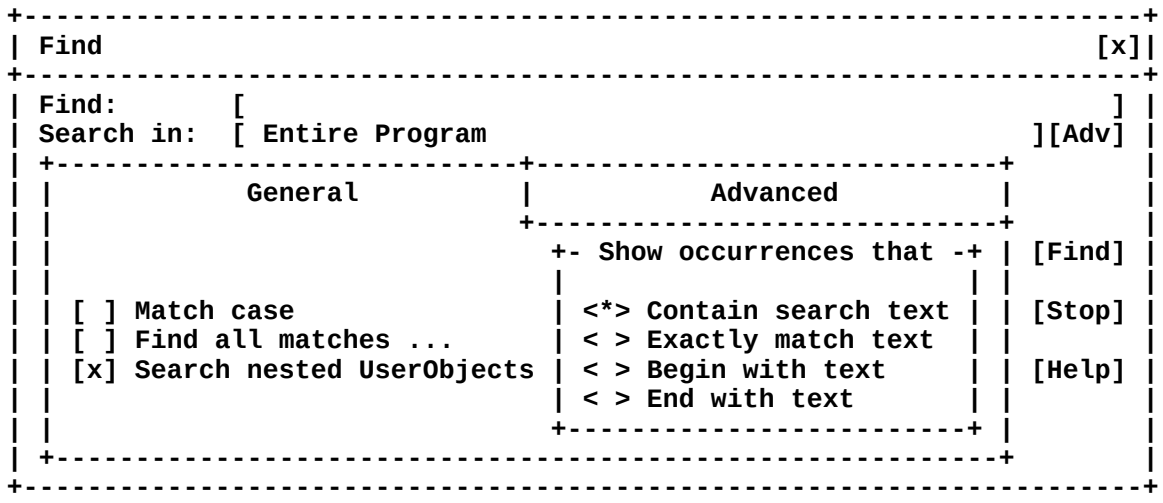
```
-+---------------+-
 | Open          |  Bring up window to inspect UserFunction.
 | Find...       |  Bring up Find panel to search through UserFunction.
 | Print...      |  Print program documentation.
 | Document...   |  Save program documentation.
 +---------------+
 | Generate Call |  Create a Call Function box with UserFunction.
 | Calls         |  Get a list of what the UserFunction calls.
 | Called By     |  Get a list of what calls the UserFunction.
 +---------------+
 | Cut           |  Remove the UserFunction from the Explorer tree.
 +---------------+
 | Description   |  Get Description box for UserFunction.
 | Properties    |  Get Properties tabfolder for UserFunction.
 +---------------+
```

For a variable you get a simple menu:

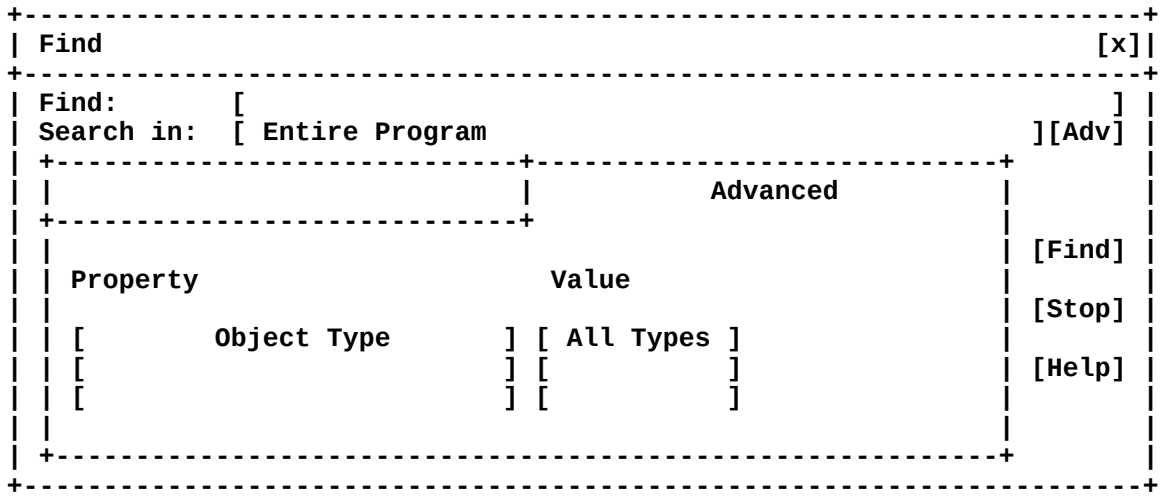
```
-+---------------+--
 | View          |  View value and type of variable.
 | References    |  List program elements that refer to the variable.
 +---------------+
```

[5] NEW DEVELOPMENT ENVIRONMENT -- FIND / PROFILER

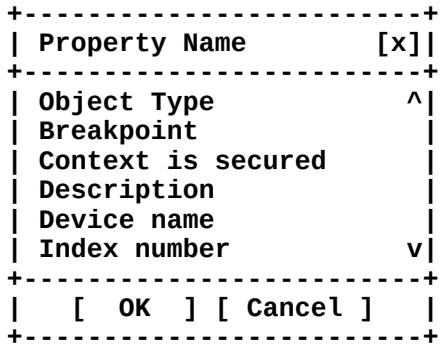
* The new Find utility allows you to search for text items through the entire program and then jump to the context in which it occurs; the utility has the appearance:



The "Advanced" menu for the Find allows you to constrain the search:

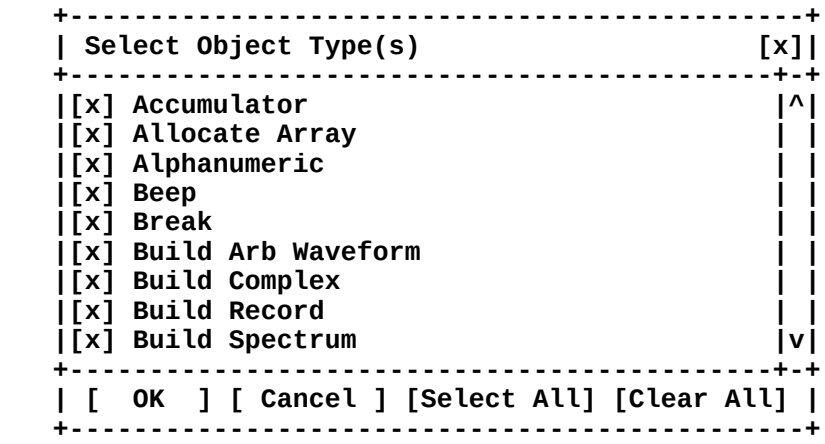


You can specify three constraints on the search; click on one of the "Property" fields and you gt a list of property types to perform the search with:

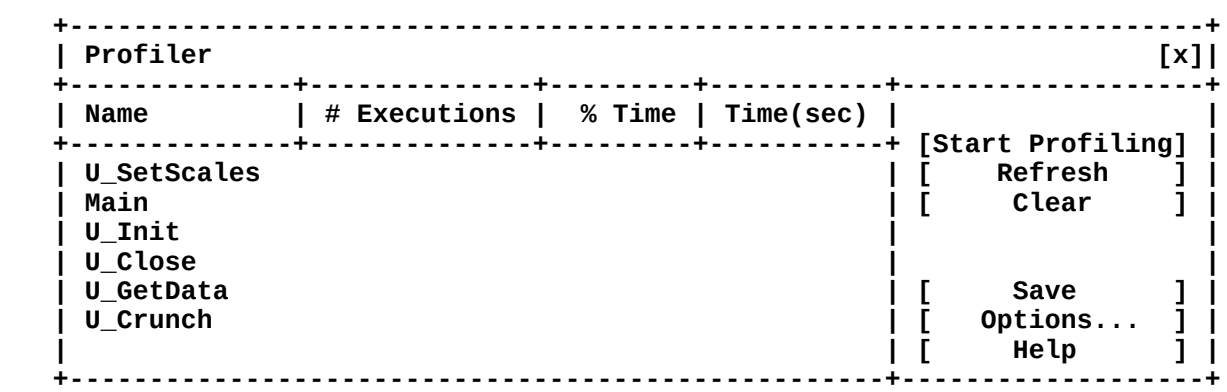


There are far too many properties to examine here ... note that most of the properties have "Value"

fields, some of which are little more than "yes/no" flags or text-entry fields, but some are more complicated -- such as that for "Object Type", which gives the full list of objects that can be searched for:



* The Profiler gives a list of User Objects and User Functions in a program to allow you to get the proportion of execution time spent in each:

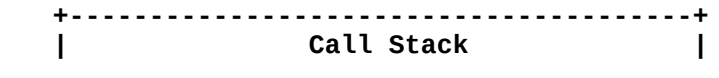


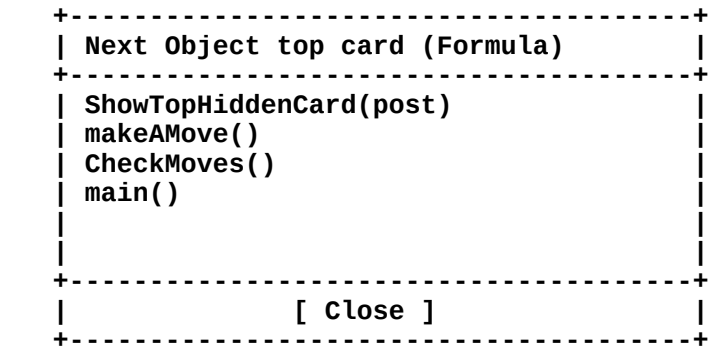
Note that this tool only works in Compiled mode, not in Compatibility mode. You can sort the data by categories and save it to a file; you select the category by clicking on the title above the data column, and reverse the sort order on that field by clicking on it again.

[6] NEW DEVELOPMENT ENVIRONMENT -- DEBUGGING & PRINTING FEATURES

* VEE 4.0 provides a large set of debugging enhancements, such as a "Show Last Error" menu entry; Error boxes also allow you to directly bring up a window giving the context in which the error occurred, while the Error box remains up to give the details of the error.

A Call Stack window allows you to track the level of nesting in a UserFunction and trace the path through which it occurred:





Note that this does not work if the program is initiated with a Start button. Multiple stepping options are also available.

* If you select "Print Program" from the File menu, you can perform a printout of the entire VEE program ... with considerably more information than was available with VEE 3.X:

- A header sheet with general file information.
- A full listing of the Program Explorer tree.
- Main & UserObject/Function Views, referenced by grid coordinates and followed by a referenced text detail list.
- An index (table of contents, basically) of the materials in the printout.

[7] NEW DEVELOPMENT ENVIRONMENT -- MISCELLANEOUS CHANGES

* There are also a number of small changes in the VEE programming environment.

First, now (in compiled mode) VEE assigns different colors to wiring depending on the data type (and you can change the colors through the system Default-Preferences tab dialogue). The default color assignments are as follows:

- Blue: Numeric -- integer, real, or complex type.
- Orange: String type
- Gray: Nil value (usually from sequence-out pins).
- Black: Structured types (like Records) or unknown types.

If the data on the line is an array type, the line will be two pixels wide instead of 1. Ensuring that specific data types flow through lines is important, since if VEE knows what data type will flow through the lines the programs will run faster.

* Second, VEE 4.0 provides a nice pair of new features called "LineTips" and "TermTips"; if you simply position the mouse cursor over a wire, a tiny window pops up to tell you what kind of data the wire is carrying, and if you position the mouse cursor over a pin on an iconized object, a tiny window pops up to give you the pin name.

* Third, Object Probe allows you to highlight all objects wired to an object by holding down the Shift key and pressing the main mouse button; you can also hold down Ctrl+Shift, then press on the main mouse button to highlight objects connected to the output.

* Fourth, you can stretch or move objects and connect wiring outside the current context window. Doing this is a little subtle ... for example, if you want to wire two objects together through a window boundary, you click on the output pin of the first object to get a moving wire, then move the mouse cursor to the edge of the window until the wire disappears.

This means you are in a "scroll region"; if you have trouble finding it, just watch the status bar at the bottom of the VEE development environment and it will tell you when you are in a scroll region. Then, after a momentary delay, the scroll will occur and you will be able to fix the wire on the second object's input pin.

Similar comments apply to moving and stretching objects off-window. This technique takes a little practice to master.

* Fifth, you can connect an output pin to multiple input pins without having to click on the output pin for each one; click on the output pin with the main mouse button and then let up to get a moving wire, then use the *secondary* mouse button to connect it to an input pin. You can keep on making connections using the secondary mouse button as many times as you like; the moving wire won't go away until you click on the primary mouse button again.

* Sixth, you can wire an input terminal back to an existing wire (nice feature).

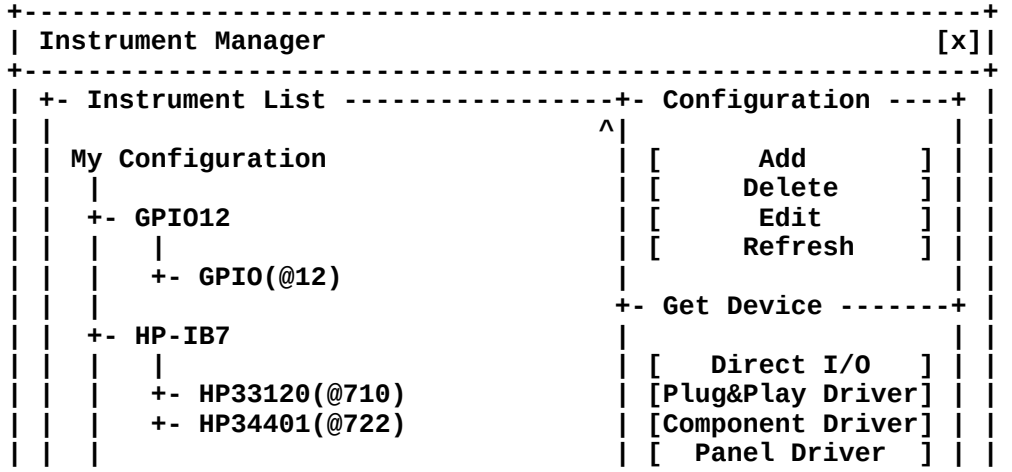
* Seventh, you can clone objects by simply holding down the Control key and doing a click-and-drag.

* Other improvements include the ability to align objects (to the right, left, top, bottom, or centered) in the development environment; and cut, copy, or paste objects between different VEE programs, using the standard Windows accelerator keys to perform these tasks.

[8] INSTRUMENT MANAGER

* The VEE 4.0 Instrument manager consolidates the various I/O configuration entries in earlier versions of VEE, providing a more convenient interface to Plug&Play Drivers, classic VEE drivers, and Direct I/O.

The Instrument Manager has a tree-structured format, somewhat similar to that of Program Explorer, but divided up by interface.



```
| |  +- Serial9                      v|                    | |
| +-----------------------------------+--------------------+ |
|        [ Save Config ] [   Cancel  ] [    Help    ]        |
+-----------------------------------------------------------+
```

Device configuration for each device in the tree is much cleaner than before; for example, for an RS-232 device you get the dialogue:

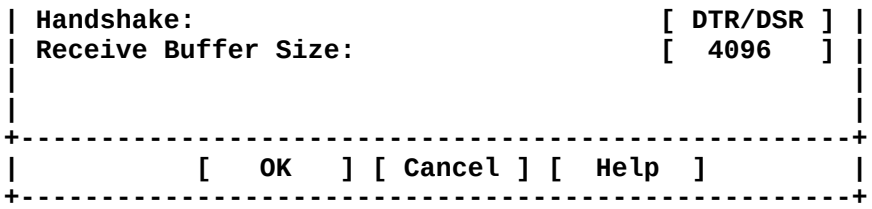
```
+----------------------------------+
|       Device Configuration       |
+----------------------------------+
|                                  |
| Name:              [ 34401     ] |
| Interface:         [  Serial  v] |
| Address (eg 9)     [     9     ] |
| Gateway:           [ This host ] |
|                                  |
|    [ Advanced I/O Config... ]    |
|                                  |
| [   OK   ] [ Cancel ] [  Help  ] |
+----------------------------------+
```

The Advanced I/O Config button leads to a TabFolder dialogue to allow you to establish general settings:

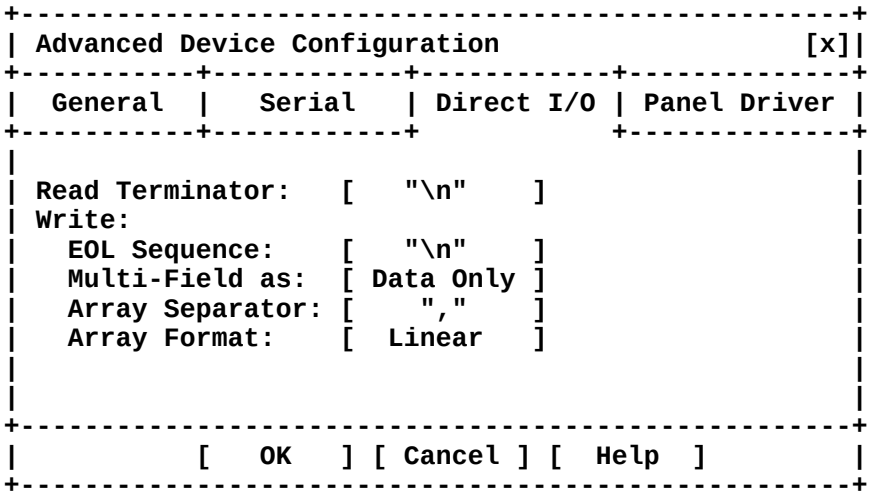
```
+-------------------------------------------------+
| Advanced Device Configuration                [x]|
+-----------+------------+------------+-----------+
|  General  |   Serial   | Direct I/O | Panel Driver |
|           +------------+------------+-----------+
|                                                 |
|  Timeout (sec):                    [   10    ] |
|  Live Mode:                        [   ON    ] |
|  Byte Ordering:                    [   MSB   ] |
|  Description (optional):           [         ] |
|                                                 |
|                                                 |
|                                                 |
|                                                 |
+-------------------------------------------------+
|           [   OK   ] [ Cancel ] [  Help  ]      |
+-------------------------------------------------+
```

-- specify serial parameters:

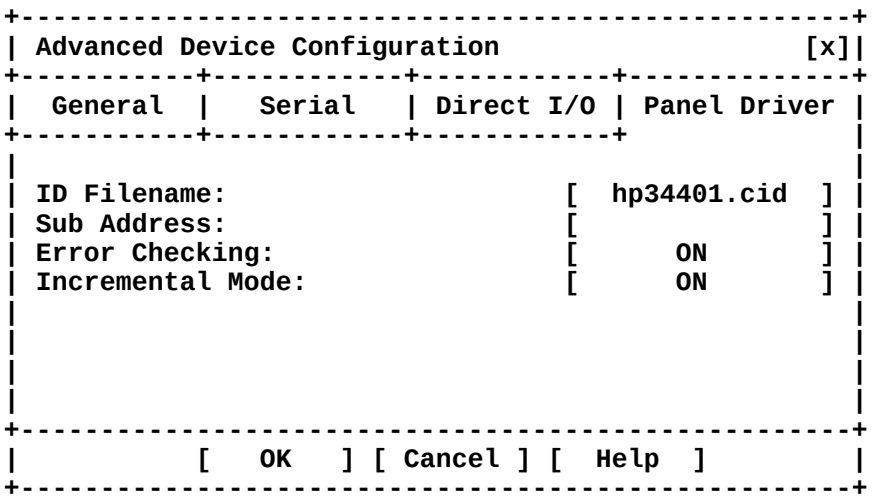
```
+-------------------------------------------------+
| Advanced Device Configuration                [x]|
+-----------+------------+------------+-----------+
|  General  |   Serial   | Direct I/O | Panel Driver |
+-----------+            +------------+-----------+
|                                                 |
| Baud Rate:                         [  9600   ] |
| Character Size:                    [    8   v] |
| Stop Bits:                         [    1    ] |
| Parity:                            [  None  v] |
```



-- typical Direct I/O configuration details:



-- and classic VEE driver configuration:



* The Instrument Manager also takes over the functions of the old separate Instrument Finder utility; you just select the level in the tree you want to scan and click on the "Refresh" button to do the scan. At the top level the scan will confirm interfaces and scan HPIB and VXI for devices; you can select an individual interface and do the scan to that as well.

All the instruments at the lowest level of the Instrument Manager are tagged by an icon ... when you bring up VEE, the icon will be the front panel of an instrument -- but if you do a scan and the

instrument is recognized, the icon turns into a PC in front of the instrument front panel to acknowledge the connection.

If the instrument has not been configured before, the entry for it in the tree will be of the form:

```
newDevice (@ 710)
```

-- and the instrument front-panel icon will have a "?" on it. If you then select that individual instrument and do a "Refresh", you get a dialogue of the form:

```
+---------------------------------+
| Identify Instrument          [x]|
+---------------------------------+
|                                 |
|     OK to Send "*IDN?" to       |
|       newDevice (@ 710)?        |
|                                 |
+---------------------------------+
|      [   OK   ] [ Cancel ]      |
+---------------------------------+
```

If the device is IEEE 488.2 compatible (that is, it understands "*IDN?") and you click on the OK button, then it goes out and identifies the instrument and changes the entry to:

```
newDevice (hp33120a @ 710)
```

You can then edit the entry to change the device name if you like to, say:

```
fgen (hp33120a @ 710)
```

Once configured, a device stays configured, though if it is off or not present a scan will turn the icon back to an instrument front panel. If you exit VEE and then reload it, all the icons will be back to instrument front panels until you do a scan again.

[9] VEE LANGUAGE CHANGES: LOCAL VARIABLES / NEW FORMULA BOX

* While the VEE language itself has not been greatly extended in VEE 4.0, there have been a few useful enhancements.

* First, you can now declare variables to be global; local; or local to a UserFunction library. This is done with the Declare Variable object:

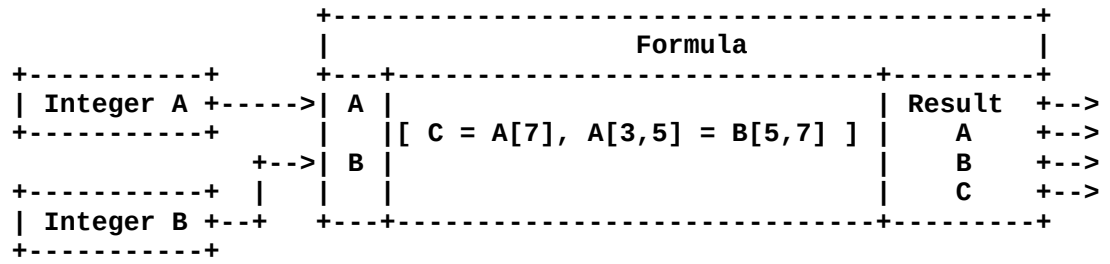
```
+----------------------------+
|       Declare Variable     |
+----------------------------+
| Name:     [    global1    ] |
| Scope:    [    Global  ][v] |
| Type:     [     Real   ][v] |
| Num Dims: [      0     ][v] |
+----------------------------+
```

The fields are straightforward ... the "Scope" field can be selected as:

- Global.
- Local To Context (which would be Main, a UserFunction, or UserObject).
- Local To Library.

If you do not declare a variable, it defaults to Global. Local variables are "static"; they retain their values between invocations of the UserFunctions that contain them.

* Second, the Formula box has been considerably enhanced: not only can you use array elements on both sides of operations, but you can include multiple formulas in the same formula box and (necessarily) set up multiple output pins.



You can set arrays, segments of arrays, record fields, and arrays of records with this scheme ... though the data structures on both sides of the equation must match.

[10] VEE OCX FOR OLE

* The 3.2 version of VEE for Windows introduced remote procedure call (RPC) capabilities for VEE ... that is, the ability of one copy of VEE to invoke UserFunctions from a library loaded on another copy of VEE operating on a different PC, linked over the Internet. An applications program interface (API) was also designed to allow a C program to perform RPCs into VEE UserFunctions on a remote PC; this was known as "Callable VEE".

VEE 4.0 extends the VEE RPC technology by adding an OLE Custom Control, or OCX (that allows you to access VEE UserFunctions from Visual BASIC, Excel, or any other OLE 2.0-compliant application.

The OCX file is installed with VEE and placed into your Windows system directory; it is named "call.ocx". If you want to install it on a PC that doesn't have VEE installed, you will need to install it manually and check it into the Windows registry (details are provided with the online help file, "call.hlp", which is installed in the main VEE installation directory).

The PC making use of the OCX must have a TCP/IP connection to the remote host where the VEE running the RPCs will reside (this can be the same PC). The remote system must have VEE 3.2 or above; if the remote system is a PC, it must also have the "veesm.exe" (VEE Service Manager) application running out of the Windows Startup group (VEE on HP-UX will be automatically configured for RPCs and does not use "veesm" as such).

In general, the OCX is incorporated into an OLE-compliant application (such as a program that is written in Visual BASIC) and configured (either by the user or the program) to specify a remote

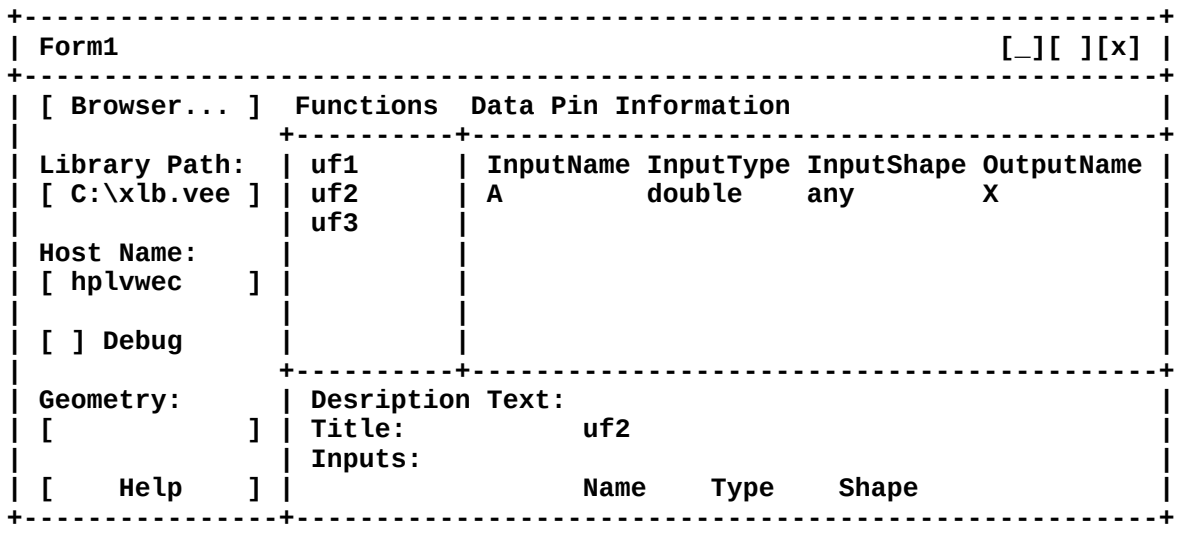
computer and various operational parameters; when initiated, the OCX goes out over the network to run VEE on the remote computer, load a UserFunction library into that VEE, execute UserFunctions with the parameters sent, and read back the results into the OCX and its host application.

* The specific operation of the OCX is easy to understand if you have ever programmed in Visual BASIC; it is essentially a generalized version of a Visual BASIC Custom Control (VBX ... OCX is essentially the next-generation version of the older VBX technology), and like Visual BASIC controls, provides a visual "widget" or "object" that includes:

- Properties: Essentially control and status variables relevant to the appearance and operation of the OCX.
- Methods: Essentially the command set of the OCX that cause it to execute the operations it was designed to perform.
- Events: Essentially interrupts provided back to the calling application relevant to the operations the OCX performs.

Other concerns are the values returned by the calls to the OCX (error codes and the like) and the data types used with the remote VEE UserFunctions.

* The OCX has the visual appearance:



The visual interface allows you to specify:

- The path to the library on the remote system.
- The Internet address of the remote system.
- A Debug flag to set single-stepping in functions on the remote system.
- The window geometry (size and placement) of VEE on the remote system.
- A list box giving the UserFunctions in the remote library.
- A list of data pin descriptions for the selected UserFunction.

The "Browse" button allows you to bring up a browser to locate the library on the remote system.

The properties associated with the OCX follow below. Examples are provided; note that "Vocx" is an arbitrary name for an instance of the OCX, and the variable names ("VeeHost", "VeeGeometry", and so on) are arbitrary as well.

```
HostName:  Specifies remote workstation.

   Vocx.HostName = "hplvewec"
   VeeHost = Vocx.HostName

GeometrySpec:  Specifies dimensions and placement of remote VEE window.

   Vocx.GeometrySpec = "640x480+20+20"
   VeeGeometry = Vocx.GeometrySpec

TimeOut:  Specifies timeout interval.

   Vocx.TimeOut = 60
   VeeTimeout = Vocx.TimeOut

Debug:  Sets debug mode on the remote system.

   Vocx.Debug = 1                    ' Set in debugging mode.
   Vocx.Debug = 0                    ' Set in debugging mode.
   DbMode = Vocx.Debug
```

The most important methods associated with the OCX include follow below. Note that execpt for the method names themselves, all the variable names ("RemLib", NLibs", and so on) are arbitrary:

```
LoadVeeLibrary:  Specify library to load.

   RemLib = "\veelibs\xlb.vee"
   RetCode = Vocx.LoadVeeLibrary( RemLib )

UnloadVeeLibrary:  Unload library from remote VEE.

   RemLib = "\veelibs\xlb.vee"
   RetCode = Vocx.UnloadVeeLibrary( RemLib )

CallFunction:  Calls remote UserFunction.

   RemLib = "\veelibs\xlb.vee"
   RemFn = "uf1"
   Args(1) = 42
   Args(2) = 666
   RetCode = Vocx.LoadVeeLibrary( RemLib, RemFn, Args )

RetrieveFunctionResults:  Returns results of calling UserFunction.

   RemLib = "\veelibs\xlb.vee"
   RemFn = "uf1"
   RetCode = Vocx.RetrieveFunctionResults( RemLib, RemFn, FnResults )
```

There are a large set of other methods, mostly intended to allow a program to interrogate the remote VEE system and automatically determine what UserFunctions are resident in the remote VEE and how they are configured. Very few users will be interested in this capability, and so you are referred to the online help for more details.

```
NumLibraries:     Identify number of libraries loaded on remote system.
NumFunctions:     Get number of functions in remote library.
NumArguments:     Returns number of arguments (input pins) for function.
NumReturns:       Returns number of results (output pins) for function.
EnumLoadedLibs:   Returns array of names of remote libraries.
EnumFunctions:    Returns array of function names in remote libraries.
EnumArguments:    Returns array of function argument (input pin) names.
EnumReturns:      Returns array of function return (output pin) names.
NumActualReturns: Returns number of results from a function call.
EnumReturnTypes:  Provides data on type and shape of return data.
AboutBox:         Tells the OCX to show its copyright dialogue.
```

OCX events (and the values they return) include:

```
HostNameChanged: Occurs when OCX object HostName field is changed.

   Private Sub Vocx_HostNameChanged( ByVal NewHostName As String )
   '  NewHostName gives new hostname entered by user.
   End Sub

LibraryAdded:  Occurs when new library is specified on OCX object.

   Private Sub Vocx_LibraryAdded( ByVal NewLibName As String )
   ' NewLibName gives new library name entered by user.
   End Sub

NewGeometry:  Occurs when user changes geometry field on OCX object.

   Private Sub Vocx_GeometryChanged( ByVal NewGeometry As String )
   ' NewGeometry gives new geometry in string format.
   End Sub

DebugChanged:  Occurs when user changes Debug flag on OCX object.

   Private Sub Vocx_DebugChanged( ByVal NewDebugState As Boolean )
   ' NewDebugState gives state of debug flag.
   End Sub

FunctionSelected:  Occurs when user selects new UserFunction from OCX.

   Private Sub Vocx_FunctionSelected( ByVal LibraryPath As String,
                                      ByVal FunctionName As String )
   ' LibraryPath gives path to VEE library.
   ' FunctionName give name of function selected.
   End Sub

RPCError:  Occurs when RPC error occurs on RPC.

   Private Sub Vocx_RPCError( ByVal ErrorNumber As Integer,
                              ByVal ErrorString As String )
   ' ErrorNumber and ErrorString give information.
   End Sub
```

* The OCX supports the following elementary data types:

- Long and short integers.
- Single and double-precision floating-point values.

- OLE strings.
- 1-dimensional arrays of all the above.

VEE composite data types are mapped into the OCX data types as follows:

- Coord: A 1-dimensional array of double-precision floating-point numbers.
- Complex: A 1-dimensional array of two double-precision floating-point numbers, with the real value as the first element and the imaginary value as the second element. Complex arrays are 1-dimensional arrays double-precision floating-point numbers with the same real/imaginary alternating scheme.
- Pcomplex: A 1-dimensional array of two double-precision floating-point numbers, with the magnitude as the first element and the phase angle as the second element. Pcomplex arrays are 1-dimensional arrays of double-precision floating-point numbers with the same magnitude/phase alternating scheme.
- Waveform: A 1-dimensional array of double-precision floating-point numbers.
- Spectrum: A polar complex array, as defined above.

[11] MISCELLANEOUS ENHANCEMENTS

* Other minor changes include:

- Enum constants have an additional data output that returns the ordinal number of the choice the user made.
- The Knob and Slider objects have added small up-and-down arrows to provide high-resolution (vernier) control inputs.
- The Meter object has control inputs to control the sub-ranges.
- You can now use Direct I/O to read the status of DSR and CTS lines on RS-232.
- You can bring out a pin on Direct I/O objects to allow them to change their LAN address (when using LAN I/O can't); you can also dynamically change the device name in a DIO object.
- You can use the secondary mouse button to "zoom" in on XY Displays objects.
- Interface Operations now has an EXECUTE PASS CONTROL transaction to allow a program to pass control; Interface Event now has an Active Controller event to allow the program to know when control has been passed back.
- There are three new system-information functions:
    - getEnv(<environment_variable>): Read value of environment variable.
    - getHostName(): Get network name of current host.
    - installDir(): Get name of VEE install directory.
- The VEE program source file format has been modified to ensure that it changes as little as

possible when additions are made.

- Customers have long asked us to provide the ability to read the values of markers from XY Trace objects, and for "normalized" FFT values; we were not able to build these improvements into VEE 4.0, but there are new example programs that provide UserFunctions to perform these tasks.

[12] FINAL COMMENTS & CAUTIONS

* Please remember the following items when dealing with VEE 4.0:

- The term "compiler" is misleading ... VEE 4.0 does not generate executable code, it simply compiles programs into "p-codes" for faster execution.
- Since VEE-Runonly now comes with the product along with an unlimited runtime license, VEE's inability to compile executable code should not be a serious sales obstacle.
- Overall performance will only be increased by a factor of 50% or so ... this makes it competitive with other applications in its field.
- The first time you run a program after it is brought into the VEE 4.0 environment, it will go through a compilation phase, meaning that it will take a little longer to execute the first time around.
- Use of compiled mode to get better speed may lead to some minor upward-compatibility problems ... program modifications may be required.
- VEE 4.0 programs are not backward-compatible to older versions of VEE.

[<>]