How to Use HP VEE

Notice

Printing History

Edition 1 - January 1995

About this Manual

This manual is your "task-oriented" guide to using HP VEE. Before you use this manual, you should learn the basics of HP VEE by reading *Getting Started with HP VEE* and completing the exercises in that manual. You should also be familiar with your computer and its operating system. Refer to the documentation that came with your computer for further information.

Once you have learned the basics, use this manual to quickly look up how to do a particular task with HP VEE. You needn't read this manual from beginning to end — just use it as needed. Chapter 1 covers some "HP VEE Concepts" to get you started. The rest of this manual (Chapters 2 through 9) consists of short task sections with titles like "To Create an Array Constant" and "To Display a Waveform." Related task sections are organized into larger sections and chapters to help you quickly find the task that you want to perform.

Using the Manual Examples:

The program files for several of the examples in this manual are provided in your *manual examples directory*. This directory is found as follows:

- *HP VEE for Windows:*

  `C:\VEE\EXAMPLES\MANUAL`

- *HP VEE for UNIX:*

  `/usr/lib/veetest/examples/manual/`

You can open an example program from the HP VEE online help system. Refer to "To Open an Example Program File" in Chapter 2 for further information.

Conventions Used in this Manual

This manual uses the following typographical conventions:

Example	**Represents**
HP VEE Reference	Italicized words are used for book titles and for emphasis.
`File`	Computer font represents text you will see on the screen, including menu names, features, buttons, or text you have to enter.
`dir` *filename*	In this context, the word in computer font represents text you type exactly as shown, and the italicized word represents an argument that you must replace with an actual value.
`File` $\Longrightarrow$ `Open`	The "$\Longrightarrow$" is used in a shorthand notation to show the location of HP VEE features in the menu. For example, "`File` $\Longrightarrow$ `Open`" means to select the `File` menu and then select `Open`.
`Zoom Out | In 2x | In 5x`	Choices in computer font, separated with a bar (|), indicate that you should choose one of the options.
$\boxed{\text{Return}}$	The keycap font graphically represents a key on the PC keyboard.
Press $\boxed{\text{Ctrl}}$+$\boxed{\text{O}}$	Represents a combination of keys on the PC keyboard that you should press at the same time.
Dialog Box	Bold font indicates the first instance of a word defined in the glossary.

Contents

Tables

1

HP VEE Concepts

HP VEE Concepts

The general concept of graphical programming, and how HP VEE programs work, is covered in *Getting Started with HP VEE*. This chapter presents some key HP VEE concepts in additional detail.

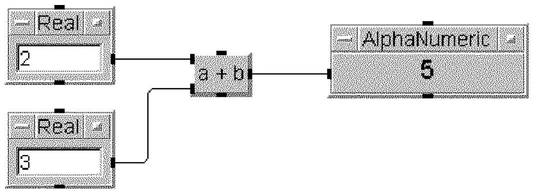
Understanding Propagation

Propagation is the general flow of execution through your HP VEE program. The propagation guidelines define the order in which HP VEE objects operate. In general, propagation is determined by **data flow**, that is the flow of data from object to object within an HP VEE program. Let's begin with a closer look at how objects operate.

How Objects Operate

An HP VEE object operates by accepting the data on its input pins, processing that data, and then returning the resulting data on its output pins. An HP VEE object will not operate until *all* of its data input pins are activated with data on them. (There is one exception. The **JCT** object has asynchronous data inputs, and will operate when *one* of its data input pins is activated with data.)

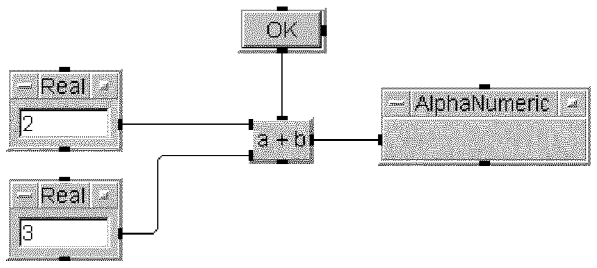
Let's look at a simple example. In the program below, the **a+b** object will not operate until there is data on both of its data input pins. Thus, both of the **Real** constant objects must operate first (in no particular order).



When the **a+b** object operates, it adds the data and activates its output pin with the resulting data. The **AlphaNumeric** object does not operate until *its* data input pin has data, so it operates last, displaying the result.

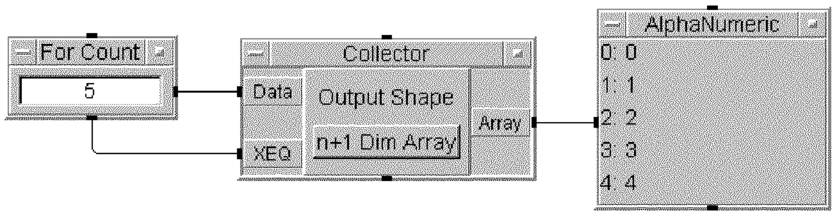
As you can see, data flow has determined the order of operation of the objects in the above program. That is, data flow determines the propagation order.

In addition to the data pins, the sequence pins can be used to determine when an object operates. In the following program, a **Confirm OK** object has been added to the previous example:



The sequence output pin of the **Confirm OK** object is connected to the sequence input pin of the **a+b** object. Sequence input pins do not have to be connected. That is, if a sequence input pin is not connected, it is ignored by the object. However, if a sequence input pin *is* connected, the object will not operate until it has been activated. So, in the above example the **a+b** object won't operate until you press (click on) the **OK** button.

There is one more type of input that affects when an object operates — the **XEQ** input. The **XEQ** input is like a data input pin, except that it is asynchronous. The **XEQ** input, if present, must be connected. However, the object can start operating before the **XEQ** input is activated. Here is an example:



```
0: 0
1: 1
2: 2
3: 3
4: 4
```

The **For Count** object repeats five times, outputting data to the **Data** input terminal on the **Collector**. The **XEQ** terminal is asynchronous, so it need not be active for the **Collector** to begin operation. However, the **Collector** won't finish operating, and won't output any data, until the **XEQ** terminal is activated. Thus, the `Collector` collects five values into an array, which it outputs when the `XEQ` terminal is activated by the sequence output pin of the `For Count` object.

NOTE

Remember that you can use `Edit Properties` from the object menu to turn on `Show Terminals`, as described in *Getting Started with HP VEE*. With `Show Terminals` turned on, the data input and output pins become "terminals", showing their names.

Basic Propagation Order

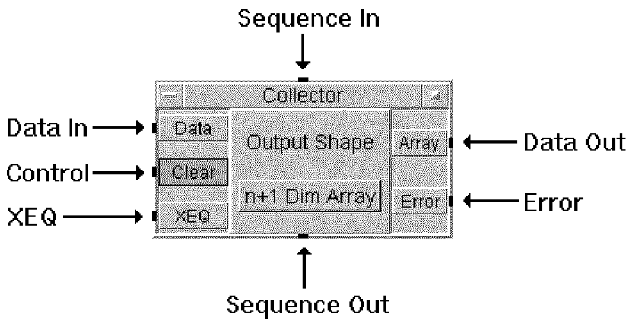
In the previous section we have seen how data flow affects the flow of execution, or propagation in an HP VEE program. Now we can state the basic propagation order.

When you press `Run` (or `Start`), the objects in your program operate in the following order:

1. All `Start` objects operate first, if they are present. (Note that you need not include `Start` objects in your program unless a data feedback loop is present.)
2. Objects that have no data input pins, and which have no sequence input pins connected, operate next.
3. Other objects operate in the order determined by data flow, as described in the previous section. In other words, objects with data input pins operate only when data is present on all data inputs, except as noted in the previous section.
4. The order of propagation can be modified by connecting sequence pins, which are described in the next section.

Pins and Propagation

In the previous two sections we've seen the basic propagation order, and how data input and output pins, sequence input and output pins, and **XEQ** inputs affect the operation of an HP VEE program. Now let's summarize all of the types of pins, and their effect on propagation.



- **Data pins** input or output a data container (refer to "Understanding Data Containers" for further information).
  - □ An object will not operate until all of its data input pins are activated. (Except the **JCT** object, which has asynchronous data inputs. The **JCT** object operates when any data input pin is activated.)
  - □ After an object operates, its data output pins are activated (if no error conditions have occurred).
- **Control pins** (optional) are asynchronous inputs that affect the state of the object but have no effect on propagation.

  Common control pins include **Clear**, **Reset**, and **Default Value**. Control pins are connected with dashed lines to indicate that propagation is not affected.
- **XEQ pins** are asynchronous input pins that force an object to operate (even if a data input pin has not yet been activated). An **XEQ** pin must be used on the **Collector** or **Set Values** object to tell the object when all data has been input.

You can add an `XEQ` input pin to a `UserObject` to force it to operate before the data and sequence input pins have been activated. However, it is not normally necessary to add an `XEQ` pin.

`XEQ` pins are activated by the presence of a data container, but the data in the container is ignored.

- **Sequence pins** are used only to specify the order of execution.
  - ▫ An object operates only after all data input pins and sequence input pins (if connected) are activated.
  - ▫ A sequence output pin activates after all the data output pins have activated and data flow has propagated as far as possible.

  You generally do not need to use sequence pins to obtain correct propagation. In fact it is usually best *not* to use them — let data flow determine the order of execution instead.

  `Show Terminals` has no effect on a sequence pin — there is no terminal label to show.

  A sequence input pin is activated by the presence of a data container, but the data in the container is ignored. A sequence output pin outputs an empty (nil) container when it activates.

- **Error pins** (optional). You can add an `Error` pin to trap an error condition generated by the object. The `Error` pin outputs the appropriate error number if an error condition occurs.

  If an error occurs, the `Error` pin is activated *instead* of any data output pins. Only the error pin and the sequence output pin (if connected) are activated.

NOTE

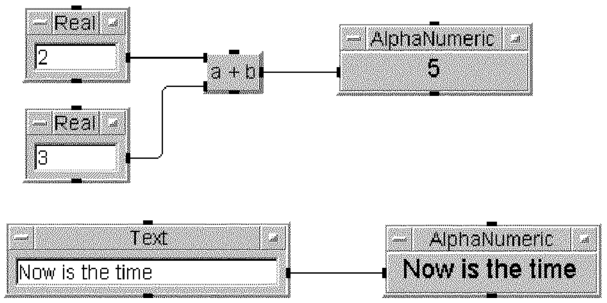
If you leave any data input pins or `XEQ` pins (if present) unconnected, an error will occur when you run your program.

You may leave data output pins, control pins, and `Error` pins unconnected. Sequence pins, more often than not, *should* be left unconnected.

Propagation of Threads and Subthreads

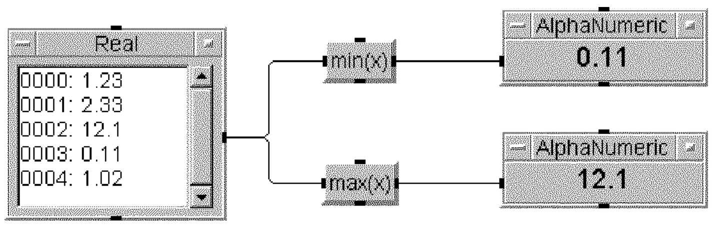
So far we've only considered propagation in a very simple HP VEE program—one that contains only one thread. Let's begin by defining what threads and subthreads are:

- **Threads**—Each independent set of connected objects in a HP VEE program is called a thread. A program can contain several threads. For example, the following program contains two parallel threads:



  The two parallel threads are not connected by data or sequence lines, so they are independent. However, they may be connected by control (dashed) lines.

- **Subthreads**—A branch of a thread is called a subthread. When two subthreads begin at the same data output pin of the same object, and there are no sequence or data lines between them, they are parallel subthreads. The following program shows two parallel subthreads branching from the data output pin of the `Real` constant object:



Parallel threads and subthreads operate round-robin style. Propagation through each thread or subthread proceeds according to the rules discussed previously. It is important to understand that multiple threads and subthreads operate in a parallel manner. That is, no single thread or subthread takes over and runs to completion before the others. However, there are two exceptions to this:

- If a thread contains an `Interface Event` or `Device Event` object, it takes over execution when an event is trapped. For example, if `Interface Event` detects an HP-IB `SRQ` message, the thread will continue to completion before any other thread can continue. Other threads are held off to allow the event to be serviced. For further information, refer to `Interface Event` and `Device Event` in the *HP VEE Reference* manual.
- If a thread has a `Start` object, and if you start the thread by pressing the `Start` button, that thread will run to completion before you can start any other threads.

Propagation Summary

The following is a summary of the propagation rules that HP VEE follows:

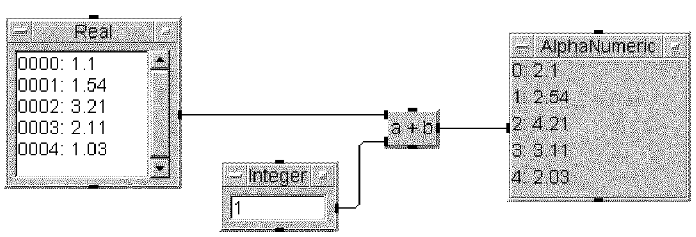
- Data flows through objects from left-to-right—sequence flows from top-to-bottom.
- All data and `XEQ` input pins must be connected.
- `Start` objects operate first. Objects with no data input pins or sequence input pin connections operate next.
- All data input pins must be activated before an object operates (except for the `JCT` object, which has asynchronous data inputs).
- If the sequence input pin is connected, it must be activated before an object can operate.
- Objects operate only once unless connected to a repeat object (for example, `For Count`), or unless forced to operate by an `XEQ` pin.
- Control pins are asynchronous and do not affect the operation of the object.
- When an error is generated from an object with an `Error` pin, the `Error` pin is activated instead of the data output pins. However, the sequence output pin *is* activated. (If there is no `Error` pin, an error message is displayed.)
- Parallel subthreads may operate in any order.
- Multiple threads operate round-robin style. (They time slice.)

Understanding Data Containers

As we've mentioned previously, the propagation of data through an HP VEE program consists of the movement of **data containers** from one object to another. The data container is the HP VEE internal data format. Every data container has both a **data type** (text, real, and so forth), and a **data shape** (scalar, one-dimensional array, and so forth).

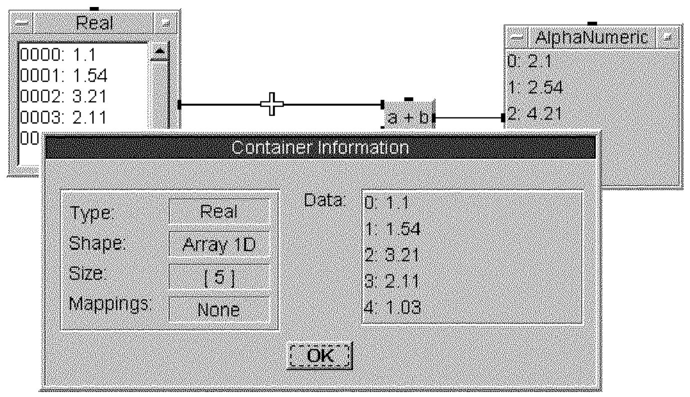
A data container may have only a single value in it, or it may have an array of several values. In either case, only one data container is output on a particular data output pin when an object operates. Let's look at an example.

In the following program, the **`Real`** constant object is configured as a one-dimensional array. The **`Integer`** constant object is configured as a scalar.



As you might expect, when the program runs the **`Real`** constant object outputs a data container that is a one-dimensional real array. The **`Integer`** constant object outputs a data container that is an integer scalar (the value `1`). So how does HP VEE add these two containers? There is no problem — HP VEE provides automatic data type conversion. HP VEE "promotes" the integer value `1` to become an equivalent real value (`1.0`). The **`a+b`** object then adds the real value `1.0` to every element in the one-dimensional real array, and outputs the resulting one-dimensional real array, as shown above.

If you are interested in the specific container that has been passed on any HP VEE data line, you can use **`Line Probe`** to look at that information. Press and hold the Shift key, move the mouse pointer near the desired line, and then click the left mouse button. The **`Container Information`** box appears. For example, the container passed on the data output line from the **`Real`** constant object of our example appears as follows.



```
0000: 1.1
0001: 1.54
0002: 3.21
0003: 2.11
00
```

```
0: 2.1
1: 2.54
2: 4.21
```

Type: Real
Shape: Array 1D
Size: [5]
Mappings: None

```
Data: 0: 1.1
      1: 1.54
      2: 3.21
      3: 2.11
      4: 1.03
```

In general, HP VEE converts data types automatically, and resolves data shapes if possible. You normally don't have to worry about how this is done. However, for technical information about this process, refer to Appendix A.

Let's look at the data types and data shapes that HP VEE supports.

Data Types

HP VEE provides 13 data types, but 3 of these types are used *only* in instrument I/O transactions. The following 10 data types are used for all HP VEE operations. That is, every HP VEE data container sent between HP VEE objects is of one of these 10 types.

NOTE

If an input terminal on an HP VEE object specifies `Any` (the default in many cases), it will accept containers of *any* HP VEE data type.

Composite data types (Waveform, Spectrum, and Coord) are associated with particular data shapes.

- `Int32` is a 32-bit two's complement integer (-2147483648 to 2147483647).
- `Real` (or `Real64`) is a 64-bit real that conforms to the IEEE 754 standard (approximately 16 significant decimal digits or ±1.7976931348623157E308).
- `PComplex` is a magnitude and a phase component in the form `(mag, @phase)`. Phase is in the currently active trigonometric units. For example, the PComplex number `4 at 30 degrees` is represented as `(4, @30)` when `Trig Mode` is set to `Degrees`. Each component is Real.
- `Complex` is a rectangular or Cartesian complex number. Each complex number has a real and an imaginary component in the form `(real,imag)`. Each component is Real. For example, the complex number `1 +2i` is represented as `(1,2)`.
- `Waveform` is a composite data type of time domain values that contains the Real values of evenly-spaced, linearly-mapped points and the total time span of the waveform. The data shape of a Waveform must be an Array 1D (a one-dimensional array).
- `Spectrum` is a composite data type of frequency domain values that contains the PComplex values of points and the minimum and maximum frequency values. Spectrum allows the domain data to be uniformly mapped as log or linear. The data shape of a Spectrum must be an Array 1D.
- `Coord` is a composite data type that contains at least two components in the form (`x`, `y`, ...). Each component is Real. The data shape of a Coord must be a Scalar or an Array 1D.
- `Enum` is a text string that has an associated integer value. The Enum data type is output by the objects found under `Data` ⟹ `Selection Control` (for example, the `Radio Buttons` object). You can access the integer value with the `ordinal(x)` function. The data shape of an Enum must be Scalar. Enum cannot be a required data input type.
- `Text` is a string of alphanumeric characters.
- `Record` is a data type composed of fields. Each field has a name and a container, which can be of any type (including Record) and any shape.

For further information on data types and data type conversions, refer to Appendix A.

Special Instrument I/O Data Types:

All integer values are stored and manipulated internally by HP VEE as the Int32 data type, and all real numbers are stored and manipulated as the Real (or Real64) data type. However, instruments generally support 16-bit integers or 8-bit bytes. Also, some instruments support a 32-bit real format. Therefore, HP VEE supports the following three data types, which are used *only* for I/O transactions involving instruments:

- `Byte` is an 8-bit two's complement byte (-128 to 127). (Byte is used in READ BINARY, WRITE BINARY, and WRITE BYTE instrument I/O transactions. The WRITE BYTE transaction is used for specialized character output to HP-IB instruments.)
- `Int16` is a 16-bit two's complement integer (-32768 to 32767).
- `Real32` is a 32-bit real that conforms to the IEEE 754 standard ($\pm$3.40282347E$\pm$38).

Data Shapes

Composite data types (Waveform, Spectrum, Record, and Coord) are associated with particular data shapes:

- The Waveform and Spectrum data types are always one-dimensional arrays.
- The Record and Coord data types can be either scalars or one-dimensional arrays. (They cannot be arrays of two or more dimensions.)

All other data types may be have either a Scalar or an Array data shape:

- `Scalar` is a single number such as `10` or `(32, @10)`.
- `Array` is an array with one to ten dimensions.

Arrays may be mapped. (A mapping is a set of continuous or discrete values that express the independent variables for an array.) Refer to Appendix A for information about mappings.

In many cases, an HP VEE object has data pins with an input data shape requirement of `Any`, meaning that the object accepts containers of more than one of the data shapes.

2

The Basics

The Basics

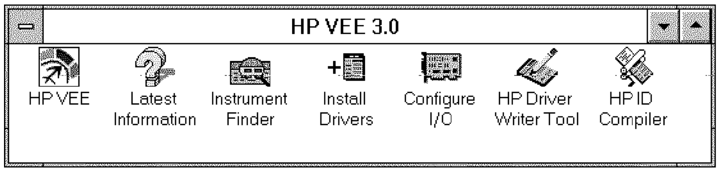
This chapter covers some general tasks that most HP VEE programmers do routinely.

Starting HP VEE

To Start HP VEE for Windows

Starting from the Group Window:

The easiest way to start HP VEE for Windows is to double-click on the `HP VEE` icon in the HP VEE group window:



In addition to the `HP VEE` icon, the group window contains icons for `Latest Information` and for five utility programs: `Instrument Finder`, `Install Drivers`, `Configure I/O`, `HP Driver Writer Tool`, and `HP ID Compiler`. You can start a utility program by double-clicking on its icon. Refer to the "HP VEE Utilities" appendix in *HP VEE Advanced Programming Techniques* for further information.

Starting from the Program Manager:

You can also start HP VEE for Windows as follows:

1. With MS Windows running, pull down the `File` menu from the `Program Manager` window and select `Run`.
2. Type `C:\VEE\VEE.EXE` and press Enter.

(If you have installed HP VEE for Windows in some other directory, substitute that directory for `C:\VEE`.)

To Start HP VEE on a UNIX Workstation

To start HP VEE on a UNIX workstation (HP-UX or SunOS):

1. Go to the shell prompt in a window under HP VUE, X11, or OpenWindows.
2. Type `veetest` and press (Return).

It doesn't matter what directory you are in because the HP VEE installation links `veetest` to the executable.

If you are using HP VUE, you can create a push button or other front panel control to start HP VEE. Refer to "Defining Front Panel Controls" in the *HP Visual User Environment 3.0 User's Guide* for further information.

There are several HP VEE utility programs that you can also start from the shell prompt. Refer to the "HP VEE Utilities" appendix in *HP VEE Advanced Programming Techniques* for further information.

You can start multiple copies of HP VEE running on the same workstation. However, if two copies of HP VEE attempt to access the same system resource (for example, an I/O interface) at the same time, unexpected results may occur.

To Start HP VEE with Command Line Options

HP VEE provides several command line options that you can use to start HP VEE in a special way. For example, you can specify an HP VEE program file to load when HP VEE starts. In fact, you can choose to load and run an HP VEE program on startup.

Table 2-1 lists the options that apply to both HP VEE for Windows and HP VEE for the UNIX workstation. Table 2-2 lists the options that apply only to the UNIX workstation. (Note that the command-line options are case-sensitive. For example, you cannot substitute `-R` for `-r`.)

To start HP VEE using the command line options:

- *HP VEE for Windows:*
  1. Pull down `File` ⟹ `Run` in the `Program Manager` window.
  2. Type `C:\VEE\VEE.EXE`, followed by the desired option or options, and press Enter.
- *HP VEE for UNIX:*
  1. Go to the shell prompt in a window under HP VUE, X11, or OpenWindows.
  2. Type `veetest`, followed by the desired option or options, and press Return.

Table 2-1. Command Line Options for HP VEE for Windows and UNIX

Option	Description
filename	HP VEE is started and the HP VEE program specified by *filename* is loaded into the HP VEE work area.
`-r` *filename*	The `-r` option starts HP VEE and runs the HP VEE program specified by *filename*. When the program finishes running, HP VEE exits immediately. However, if an error occurs, an error message is displayed before HP VEE exits. If you do not specify a file name, HP VEE ignores the `-r` option.
`-noerrdisp`	If this option is used in conjunction with `-r`, and the specified program does not run due to an error, HP VEE exits immediately without displaying an error message.
`-d` *directory*	The `-d` option starts HP VEE and uses the related files, such as instrument drivers, located in the specified directory instead of the default HP VEE installation directory.
`-iconic`	The `-iconic` option starts HP VEE as an icon instead of a window. Double-click on the icon to open it to a window.
`-notoolbar`	This option is used in conjunction with `-r` to hide the HP VEE toolbar while the program is running.
`-idmonitor`	This option starts a copy of HP VEE that contains only the `ID Monitor` and the `Bus I/O Monitor`.

Table 2-2. Command Line Options for HP VEE for UNIX Only

Option	Description
`-display` *Xservername*	Specifies the X Windows display server to use instead of the default X display. Using this option you can have HP VEE execute on one workstation, but use the keyboard and display of another workstation.
`-geometry` *width height xoffset yoffset*	Specifies an initial window geometry to be used instead of the default geometry. For example, `veetest -geometry 800x500+0-0` starts HP VEE in a window that is 800 pixels wide and 500 pixels tall, and that is placed in the lower left corner of the screen.
`-help`	Shows the HP VEE command line options. (Not to be confused with the HP VEE online help system, which is launched from within HP VEE.)
`-name` *name*	The `-name` option starts HP VEE and sets the application name of HP VEE to *name* instead of `veetest`. HP VEE uses *name* to specify X11 options in addition to the X11 options specified by the default application class (`Vee`). Refer to the "Configuring HP VEE" appendix in *HP VEE Advanced Programming Techniques* for further information.

Example: Starting HP VEE for Windows Using `-r` *filename*

To start HP VEE for Windows and automatically run the program `MYPROG.VEE`:

1. With MS Windows running, pull down the `File` menu from the `Program Manager` window and select `Run`.
2. Type `C:\VEE\VEE.EXE -r MYPROG.VEE` and press (Enter).

Example: Starting HP VEE for UNIX Using `-r` *filename*

To start HP VEE for Windows and automatically run the program `myprog`:

1. Go to the shell prompt in a window under HP VUE, X11, or OpenWindows.
2. Type `veetest -r myprog` and press (Return).

Finding Files

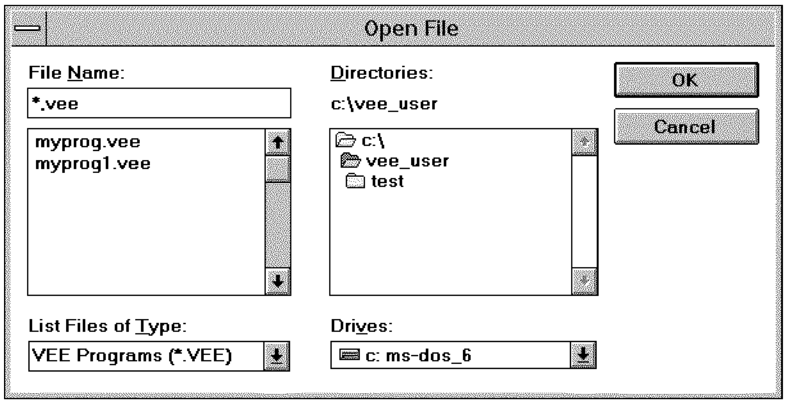
HP VEE provides convenient ways to find and open files that contain user programs, example programs, and library objects and programs.

To Open an HP VEE Program File

HP VEE for Windows:

To open an HP VEE program file from HP VEE for Windows:

1. Select `File` ⟹ `Open`. The `Open File` dialog box appears:



*.vee

c:\vee_user

myprog.vee
myprog1.vee

c:\
vee_user
test

VEE Programs (*.VEE)

c: ms-dos_6

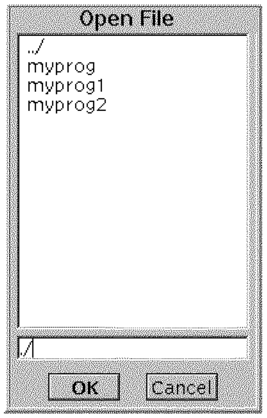
By default, the `Open File` dialog box looks for program files to open in the directory `C:\VEE_USER`. (By default, `File` ⟹ `Save` saves your programs in that directory.) The dialog box allows you to select a file name or change directories using the standard Windows dialog box conventions. For further information about Windows dialog boxes, refer to your Microsoft Windows documentation.

2. Click on the file name of the program file that you want to open. For example, to load the program in the file named **`myprog.vee`** you would click on **`myprog.vee`**, and then click on `OK`. Or enter the name of a file in the **`File Name`** type-in field and click on `OK`.

HP VEE for UNIX:

To open an HP VEE program file from HP VEE on a UNIX workstation:

1. Select `File` ⟹ `Open`. The `Open File` dialog box appears:



   By default, the `Open File` dialog box looks for program files to open in the directory in which you started HP VEE. For example, if you started HP VEE by executing **`veetest`** in the directory **`/users/jim/myprogs/`**, that is where the `Open File` dialog box will look for program files. (By default, `File` ⟹ `Save` saves your programs in that directory.)

2. Click on the file name of the program file that you want to open. For example, to load the program in the file named **`myprog`** you would click on **`myprog`**, and then click on `OK`. Or enter the name of a file in the type-in field and click on `OK`.

   If you want to change directories, click on a directory (for example, **`../`**) to change the path, or enter a new path in the type-in field.

To Open an Example Program File

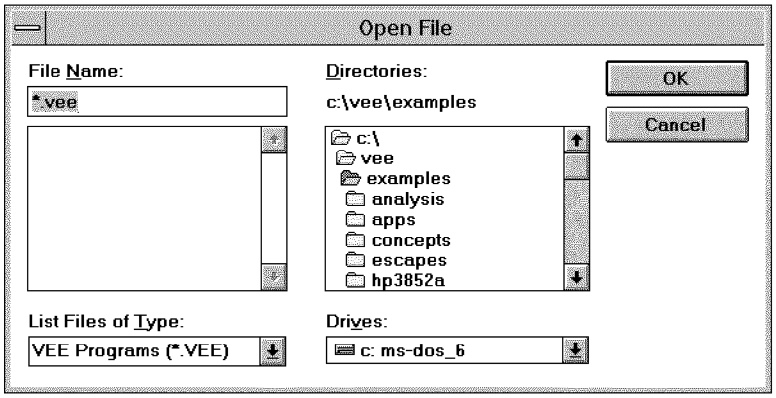
HP VEE for Windows:

Several example programs are provided with HP VEE. These examples are found in subdirectories under `C:\VEE\EXAMPLES`. For example, the examples from this manual are found in the subdirectory:

`C:\VEE\EXAMPLES\MANUAL`

You can open any program file, including an example, using **`File`** ⟹ **`Open`**. However, for your convenience you can open an example program file from **`Help`**:

1. Select **`Help`** ⟹ **`Open Example`**. The **`Open File`** dialog box appears, but is "targeted" on the examples directory:



File Name: *.vee
Directories: c:\vee\examples
c:\
vee
examples
analysis
apps
concepts
escapes
hp3852a
List Files of Type: VEE Programs (*.VEE)
Drives: c: ms-dos_6

2. Select a subdirectory, and click on the name of the example file you want to open. Click on `OK` to open the file.

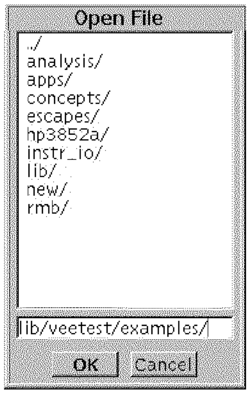
HP VEE for UNIX:

Several example programs are provided with HP VEE. These examples are found in subdirectories under **/usr/lib/veetest/examples/**. For example, the examples from this manual are found in the subdirectory:

/usr/lib/veetest/examples/manual/

You can open any program file, including an example, using **File** ⟹ **Open**. However, for your convenience you can open an example program file from **Help**:

1. Select **Help** ⟹ **Open Example**. The **Open File** dialog box appears, but is "targeted" on the examples directory:



2. Select a subdirectory, and click on the name of the example file you want to open. Click on **OK** to open the file.

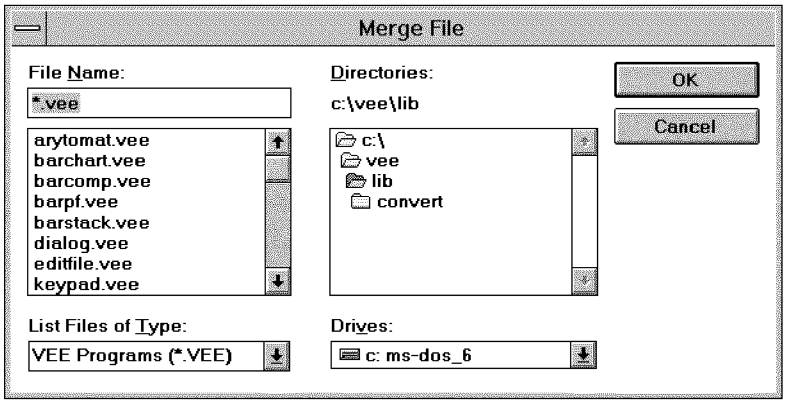
To Merge Objects and Programs from Library Files

HP VEE provides several library objects (pre-built `UserObjects`) and complete library programs that you can merge into your own HP VEE program. To do this, use `File` ⟹ `Merge` to merge the appropriate file into the work area.

HP VEE for Windows:

To merge a library object or program into the HP VEE for Windows work area:

1. Select `File` ⟹ `Merge`. The `Merge File` dialog box appears:



   By default, the `Merge File` dialog box looks for library objects and programs in the directory `C:\VEE\LIB`. You can also merge any HP VEE program file into the work area by changing directories. The dialog box allows you to select a file name or change directories using the standard Windows dialog box conventions. For further information about Windows dialog boxes, refer to your Microsoft Windows documentation.

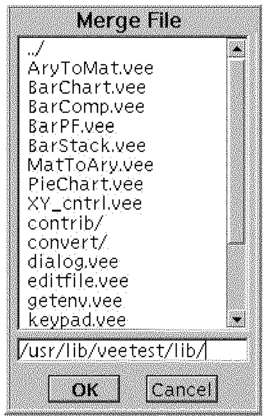
2. Click on the file name of the library file that you want to open. For example, to merge the library object named `keypad.vee`, click on `keypad.vee` and then click on `OK`. The `keypad` object (a `UserObject`) is

merged into your workspace. You can connect it into your program like any HP VEE object.

HP VEE for UNIX:

To merge an HP VEE library object or program file into the HP VEE work area:

1. Select `File` ⟹ `Merge`. The `Merge File` dialog box appears:



By default, the `Merge File` dialog box looks for files to open in the directory `/usr/lib/veetest/lib/`. However, you can change directories just as with the `File Open` dialog box. You can merge any HP VEE program file into the HP VEE work area.

2. Click on the file name of the library object or program that you want to merge. For example, to merge the `keypad.vee` object, click on `keypad.vee` and then click on `OK`. The keypad object (a `UserObject`) is merged into the work area and you can connect it into your program like any other HP VEE object.

   If you want to change directories, click on a directory (for example, `../`) to change the path, or enter a new path in the type-in field.

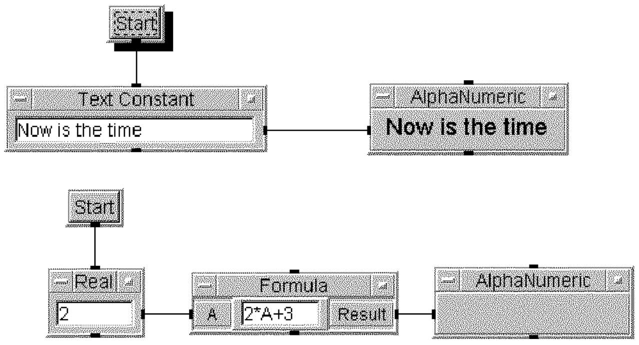
Controlling Program Flow

The fundamental concepts of HP VEE program flow are covered in Chapter 1. The following are some practical ways to control program flow.

To Start an HP VEE Program

- To start an entire HP VEE program, including all threads within the program, press (click on) the **Run** button in the HP VEE tool bar. All threads within the work area will start, whether or not they have start buttons.
- If you want to be able to start an individual thread in a program independently, add a `Start` button to the thread. When you press a start button, only the thread that contains the start button will start.
- If a thread employs *feedback*, the thread must contain a `Start` object to indicate where propagation is to begin. This is the only case where a `Start` object is required. (If you run a program that includes a thread with feedback, but no `Start` object, an error is reported.)

Let's look at an example. The program shown below includes two threads, each with a `Start` object.



If you want to run just the top thread, press its `Start` button. The thread runs, displaying "Now is the time" in the `AlphaNumeric` object, as shown. However, the bottom thread does not run.

If you press `Start` in the bottom thread, it runs, but the top thread does not.

If you press `Run`, both threads of the program will run. In each thread, propagation begins at the `Start` object.

To Branch within an HP VEE Program

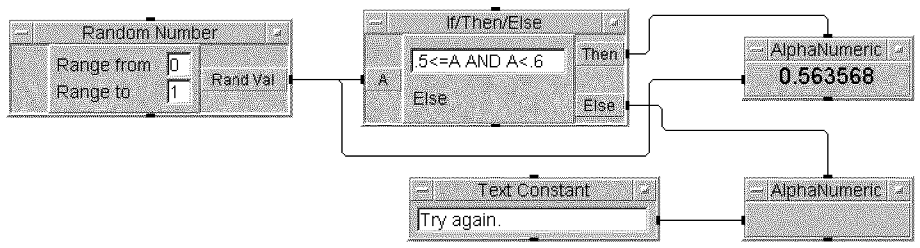
To branch the flow of execution within an HP VEE program, use the `If/Then/Else` or `Conditional` $\Longrightarrow$ objects found under the `Flow` menu.

By default, the `If/Then/Else` object tests an input value `A` against the conditional test `0<=A AND A<10`. The result of the expression is output on either the `Then` terminal (if the expression is true) or the `Else` terminal (if the expression is false).

To set up a conditional branch using the `If/Then/Else` object:

1. Add an `If/Then/Else` object (from the `Flow` menu) to the work area.
2. Modify the conditional test expression as desired. For example, to test values to see if they are in the range "greater than or equal to 0.5 and less than 0.6", use the expression `.5<=A AND A<.6`. (You can use any of the logical or relational math operators supported by HP VEE in the expression field of an `If/Then/Else` object.)
3. Add additional data input terminals if required by your new expression. By default, there is only one data input terminal (`A`). If you want to test a value `A` against a value `B` (for example, `A<=B`), add a data input terminal named `B`.
4. Connect the sources of data to be tested to the data input terminals.
5. Finally, connect the `Then` and `Else` output terminals to activate different objects or threads in your program, as desired.

Here is an example using **If/Then/Else**:



The value output by the **Random Number** object is tested against the expression **.5<=A AND A<.6**. If the value is in the specified range, the **Then** output terminal activates the top **AlphaNumeric** object, which displays the random number as shown in the above figure. If the value is not in the specified range, the **Else** output terminal activates the bottom **AlphaNumeric** object, which displays **Try Again**.

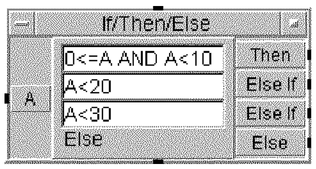
The above example, (saved as **manual01.vee** in your manual examples directory) illustrates a trivial case. Of course, you can use the **Then** and **Else** terminals to activate any sort of program thread, or a **UserObject**.

The Conditional Objects:

The objects under **Flow** $\Longrightarrow$ **Conditional** $\Longrightarrow$ (**If A == B**, **If A ~= B**, **If A != B**, **If A < B**, **If A > B**, **If A <= B**, and **If A >= B**) are really pre-defined **If/Then/Else** objects. Each has two data input terminals (**A** and **B**) and performs a pre-defined test comparing the value of **A** to the value of **B**. The **Then** and **Else** output terminals work just as in the **If/Then/Else** object. In fact, you can add or delete terminals and change the expression just as you would in an **If/Then/Else** object. For example, you could add a **C** terminal to the **If A == B** object and change the test to **A == B OR B <= C**.

Adding Else If Conditions:

The If/Then/Else and Conditional objects have another feature — the Else If condition. You can add one or more Else If conditions (*object menu* ⟹ Add Else/If). Each Else If condition is tested if the previous condition is not met. For example, the following If/Then/Else object has two Else If conditions:



This object works as follows:

1. The expression 0<=A AND A<10 is evaluated, and the result is output on the Then terminal if the expression is true.
2. Otherwise, the expression A<20 is evaluated, and the result is output on the corresponding Else If terminal if the expression is true.
3. If A<20 is false, the expression A<30 is evaluated, and the result is output on the corresponding Else If terminal if A<30 is true. If false, the result is output on the Else terminal.

To Loop Part of an HP VEE Program

You can loop (repeat) part of your HP VEE program using one of the following objects, found under `Flow` $\Longrightarrow$ `Repeat`:

Table 2-3. Repeat Objects

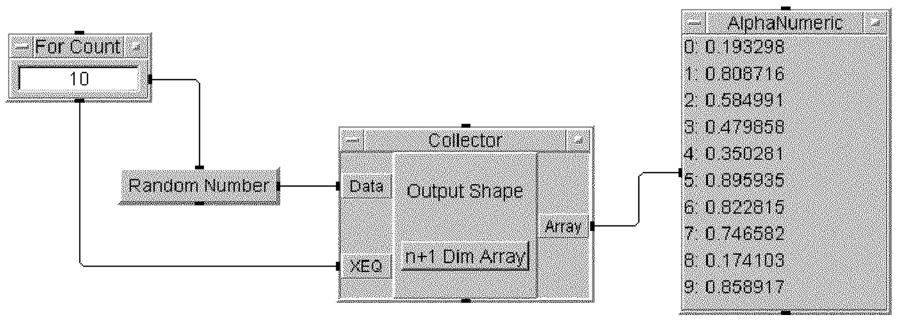
Object	Operation
`For Count`	Outputs the count, starting with 0, repeating the number of times specified in the entry field. The default is 10, which causes the count to repeat 10 times, from 0 to 9.
`For Range`	Outputs a value on the data output pin repeatedly, starting with the value `From`, incrementing by the value `Step` on each repeat. `For Range` repeats until the output value is greater than the `Thru` value if `Step` is positive, or until the output value is less than the `Thru` value if `Step` is negative.
`For Log Range`	Similar to `For Range`, but on a log scale. The initial value output is `From`, but the `n`th value is `From*exp10(n/(/Dec))`. `For Log Range` repeats until the output value is greater than the `Thru` value if `/Dec` is positive, or until the output value is less than the `Thru` value if `/Dec` is negative.
`Until Break`	Activates its data output pin continuously until a `Break` object operates within the thread.
`On Cycle`	Outputs the current time value periodically (the default period is 1 second) until a `Break` object operates within the thread.

NOTE

You can create a repeated thread or loop using any of the objects listed in Table 2-3. In addition, you can include the `Next` and `Break` objects (also found under `Flow` $\Longrightarrow$ `Repeat`) in a loop "hosted" by any of these objects. The `Next` object causes the loop to immediately begin its next iteration. The `Break` object causes the loop to immediately terminate.

To repeat part of your program in a loop:

1. Add one of the five **Repeat** objects, listed in Table 2-3 to the work area.
2. Connect the data output pin of the **Repeat** object to the object or thread that you want repeated. In the following example, the **For Count** object causes the **Random Number** object to operate 10 times:

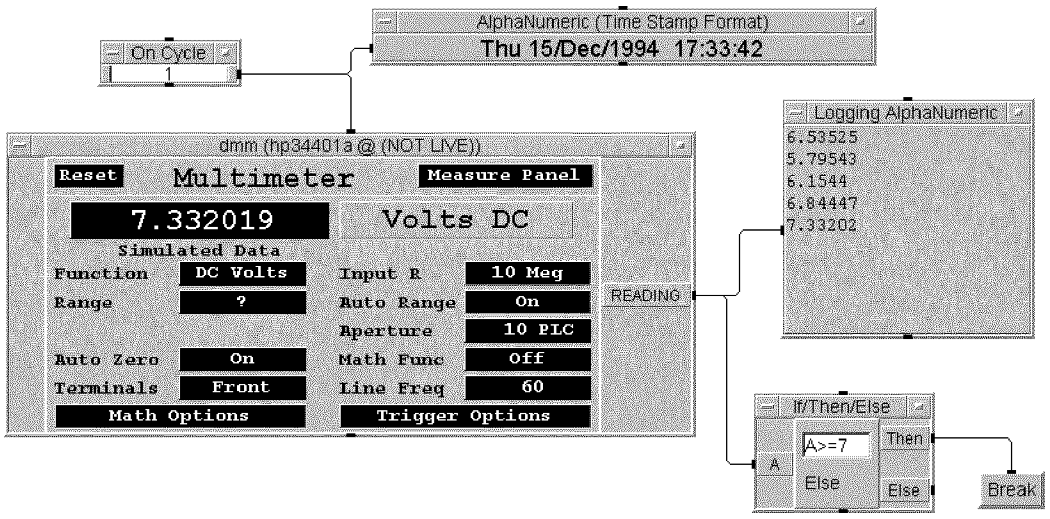


```
0: 0.193298
1: 0.808716
2: 0.584991
3: 0.479858
4: 0.350281
5: 0.895935
6: 0.822815
7: 0.746582
8: 0.174103
9: 0.858917
```

3. If needed, use the sequence output pin of the **Repeat** object to indicate when the **Repeat** object is done operating (the loop is done repeating). In the above example, the **Collector** collects the 10 random values in an array, but doesn't output the array until the sequence output pin of **For Count** activates the **XEQ** pin on the **Collector**.

Example: Taking a Reading Every Second

Let's look at another example, this time using the `On Cycle` object. `On Cycle` outputs the current time periodically (every second by default). In the following example (`manual02.vee` in your manual examples directory), the `On Cycle` object causes the instrument panel object for the HP 34401A Multimeter to output a reading every second:



```
7.332019    Volts DC
Simulated Data
Function     DC Volts     Input R      10 Meg
Range        ?            Auto Range   On
                          Aperture     10 PLC
Auto Zero    On           Math Func    Off
Terminals    Front        Line Freq    60
```

```
6.53525
5.79543
6.1544
6.84447
7.33202
```

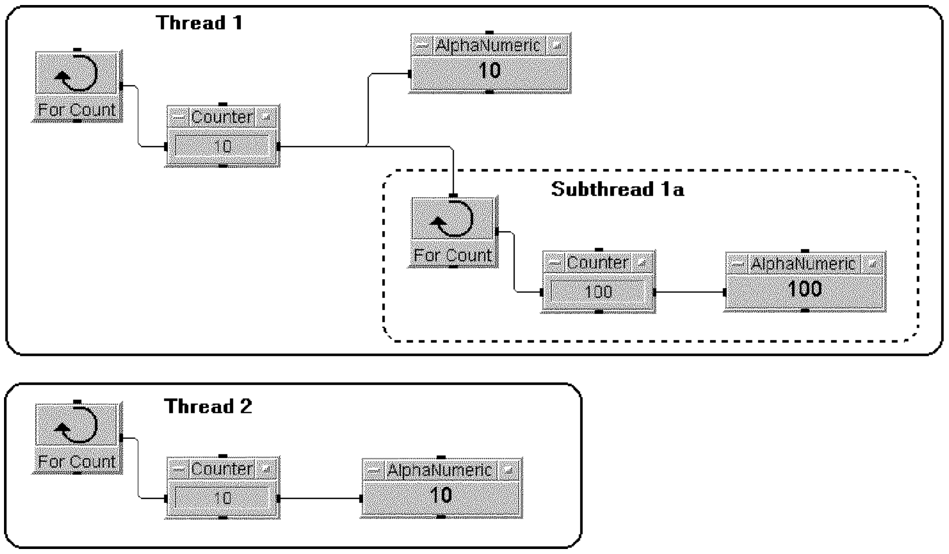
Actually, the above program is set up to produce simulated readings, since live mode is `OFF` for the HP 34401A. Refer to Chapter 5 for further information about using instrument panels.

Each reading is output to the `Logging AlphaNumeric` display, and to the `If/Then/Else` object. The `If/Then/Else` object causes the `Break` object to operate, stopping the loop, on the first value that is greater than or equal to 7. The `AlphaNumeric` object, configured for the `Time Stamp` format, reports the time of each reading as it is taken.

Example: Parallel and Nested Loops

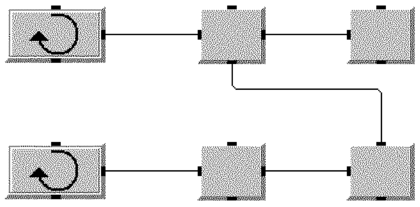
Your HP VEE program can include parallel loops, which run concurrently. In the example below, `Thread 1` and `Thread 2` are parallel loops.

You can also nest a loop within another loop. In the example below, `Subthread 1a` is nested within `Thread 1`. You can even have multiple levels of nesting.

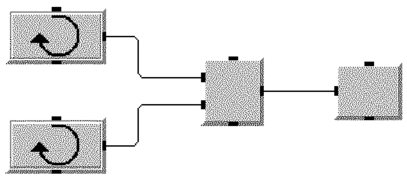


When using loops, you should avoid the following situations, which can cause a propagation paradox and lead to unpredictable results:

1. Do not cross connect parallel loops. *Avoid situations like the one shown below:*



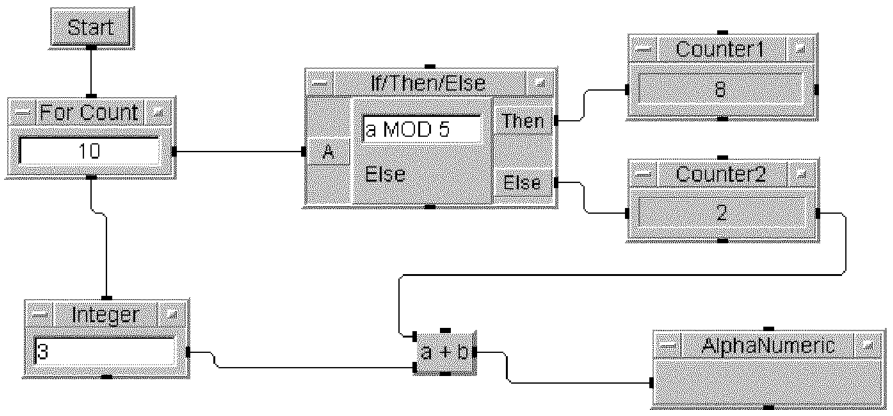
2. Do not merge loops (with a `JCT` or `Math` object, for example). *Avoid situations like the one shown below:*



Example: Avoiding Loss of Data with Sample & Hold

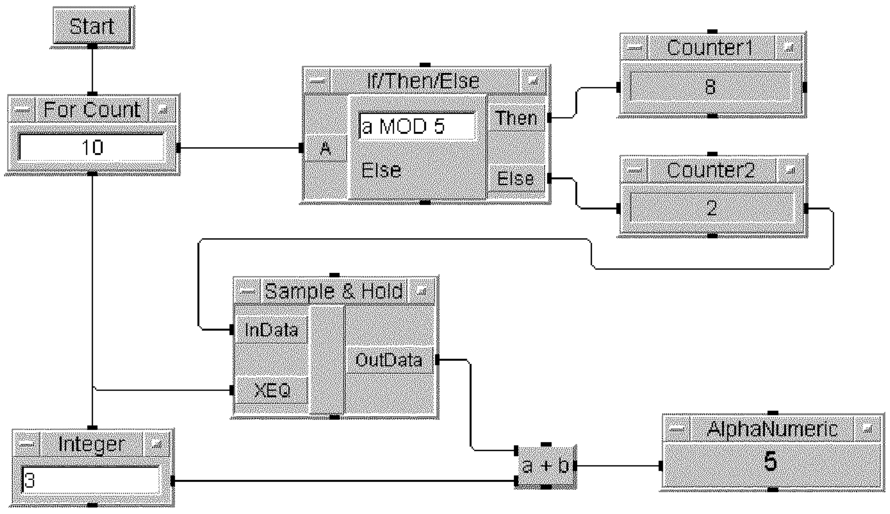
If your program includes a loop that contains an object that may not output data on every iteration of the loop, and if an object outside the loop tries to read data from that object after the loop finishes, the data may be lost. This is because when a loop completes an iteration, all data containers sent during the previous iteration are invalidated before the next iteration. This prevents "old" data from a previous iteration of the loop from being reused in the current iteration.

An example will show this (`manual03.vee` in your manual examples directory):



In the above program the `For Count` object "hosts" the loop consisting of the `If/Then/Else` object and the two `Counter` objects. But because of the branching performed by the `If/Then/Else` object, neither `Counter` operates on every iteration of the loop. On the last iteration `Counter1` operates, but `Counter2` does not, so there is no valid data on the output pin of `Counter2` when the `For Count` object activates the sequence input pin of the `Integer` constant. Thus, the `A+B` object can't operate. (Refer to the propagation rules in Chapter 1.)

To solve this problem, add a **`Sample & Hold`** object as shown below (**`manual04.vee`** in your manual examples directory):



Each time `Counter2` operates, it sends a data container to the `Sample & Hold` object, where it is stored internally. When the `For Count` object finishes and activates the `XEQ` pin on the `Sample & Hold` object, the stored data is output to the `A+B` object, and the program completes executing.

To Stop an HP VEE Program

Once you have started an HP VEE program, it will run according to the propagation rules covered in Chapter 1 until it finishes, and then stops. However, you can stop an HP VEE program before it finishes propagating, as follows:

- *The* **`Stop`** *button.* You can stop a running HP VEE program by pressing (clicking on) the **`Stop`** button in the HP VEE tool bar:
  - ☐ Click once on the **`Stop`** button to pause the program momentarily. Then use the **`Cont`** to continue the program, or the **`Step`** button to continue running the program one step at a time.
  - ☐ Click twice on the **`Stop`** button to permanently stop the program. Then use the **`Run`** button to start the program over at the beginning.
- *The* **`Stop`** *and* **`Exit Thread`** *objects.* You can include a **`Stop`** or **`Exit Thread`** object in your program to define when it should terminate. For example, you can can use an **`If/Then/Else`** object to conditionally branch to a **`Stop`** or **`Exit Thread`** object. When either of these objects operate, the program stops permanently just as though you clicked twice on the **`Stop`** button.
- *The* **`Raise Error`** *object.* You can cause an error to occur, pausing your program, with the **`Raise Error`** object. For further information about the **`Raise Error`** object, and about using **`Error`** output terminals, refer to "To Trap an Error" in Chapter 9.

Setting Colors, Fonts, and Titles

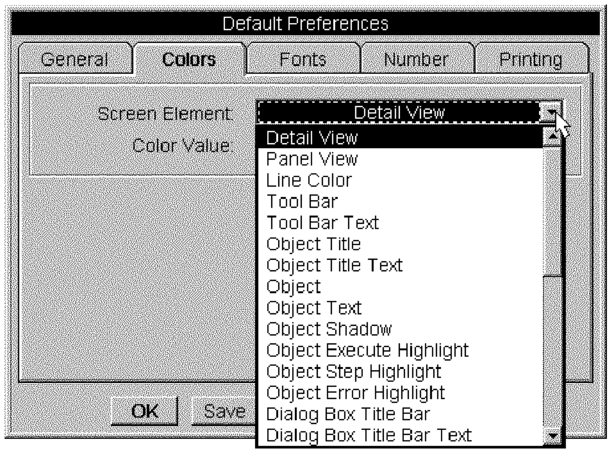
You can modify the appearance of your HP VEE program by changing the colors, fonts, and titles used:

- HP VEE allows you to change the default colors and fonts used in the entire work area (for all of your programs) by changing your default **preferences**. Refer to "To Change Default Colors" and "To Change Default Fonts", later in this section, for information on how to make these global changes.
- HP VEE allows you to change the work area title, and the colors and fonts used for the tool bar, by changing your work area **properties**. Refer to "To Change Colors for the Work Area" and "To Change an Object or Work Area Title", later in this section, for further information.
- HP VEE allows you to change the title of an object, and some of the colors and fonts used for the object, by changing the object **properties**. Refer to "To Change Colors for an Object", "To Change Fonts for an Object", "To Change an Object or Work Area Title", later in this section, for further information.

To Change Default Colors

You can change the default colors for all of your programs using the **`Default Preferences`** dialog box:

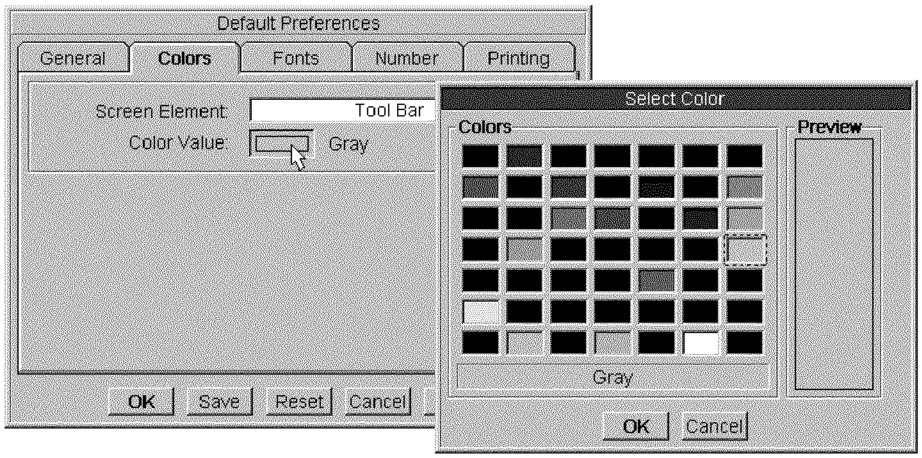
1. Select **`File`** ⟹ **`Edit Default Preferences`**, and select the **`Colors`** tab in the dialog box.
2. Click on the **`Screen Element`** field or its arrow. The drop-down list displays the screen element choices:



Screen Element: Detail View
Color Value:

Detail View
Panel View
Line Color
Tool Bar
Tool Bar Text
Object Title
Object Title Text
Object
Object Text
Object Shadow
Object Execute Highlight
Object Step Highlight
Object Error Highlight
Dialog Box Title Bar
Dialog Box Title Bar Text

3. Click on a screen element in the list. For example, click on **`Tool Bar`**. The drop-down list disappears and the current tool bar color is shown in the **`Color Value`** field. Now click on the **`Color Value`** button. The **`Select Color`** dialog box appears:



Screen Element: Tool Bar
Color Value: Gray
Gray

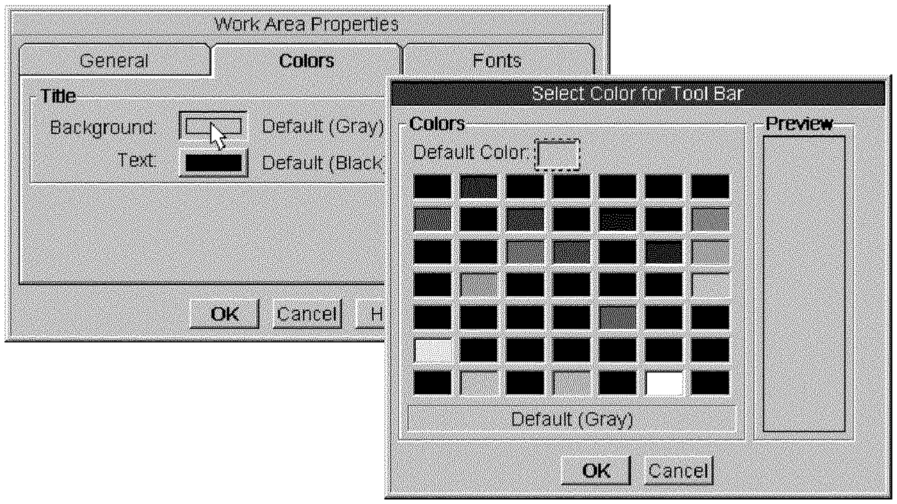
4. The `Select Color` dialog box displays a palette of the available colors. The current color selection (`Gray`) is highlighted in the palette with a dashed outline. The color is also shown in the `Preview` area. To select a new color for the tool bar, click on a different color in the palette, and then click on `OK` to return to the `Default Preferences` dialog box.
5. Choose how you want to save your color selection:
   a. If you want to save your new color selection for the tool bar as a permanent default color, click on `Save`. This will save the new default color in your `VEE.RC` or `.veerc` file. The new color will be the default for future HP VEE sessions (until you change it again).
   b. If you want to use your new color selection only for the current HP VEE session (until you select `File` $\Longrightarrow$ `New`, `Open`, or `Exit`), click on `OK`.
   c. If you don't want to save your new color selection at all, click on `Cancel`.

   Note that you can make color selections for multiple screen elements before you click on `Save`, `OK`, or `Cancel`.

To Change Colors for the Work Area

You can change the background and text colors for the work area tool bar by using `File` ⟹ `Edit Properties`.

1. Select `File` ⟹ `Edit Properties`. The `Work Area Properties` dialog box appears. Select the `Colors` tab in the dialog box.
2. To change the background color for the tool bar, click on the `Background` button:

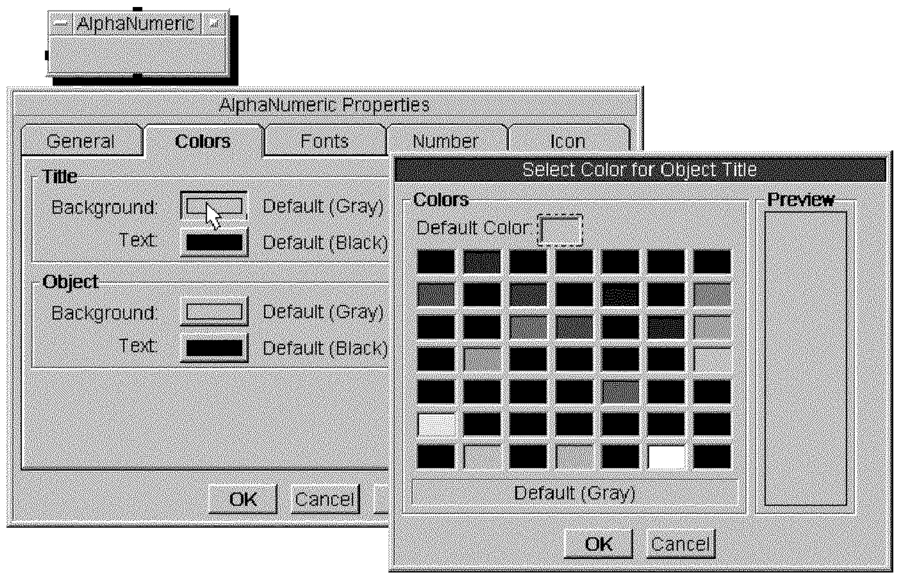


3. The `Select Color for Tool Bar` dialog box displays a palette of the available colors. The current color selection (`Default (Gray)` in our example) is highlighted in the palette with a dashed outline. The color is also shown in the `Preview` area. To select a new color for the tool bar, click on a different color in the palette, and then click on `OK` to return to the `Work Area Properties` dialog box.
4. Click on `OK` to accept the new color selection, or `Cancel` to reject it.

To Change Colors for an Object

Every object has an object properties dialog box that you can reach from the object menu. You can change certain colors for the object using this dialog box. Let's use an `AlphaNumeric` object in our example:

1. Add an `AlphaNumeric` object to the work area (`Display` ⟹ `AlphaNumeric`).
2. Select `Edit Properties` from the object menu, and then select the `Colors` tab in the `AlphaNumeric Properties` dialog box.
3. The colors that you can change depend on the particular class of object. For the `AlphaNumeric` object, click on the `Background` button under `Title`:



Title
Background: Default (Gray)
Text: Default (Black)
Object
Background: Default (Gray)
Text: Default (Black)

4. The `Select Color for Object Title` dialog box displays a palette of the available colors. The current color selection (`Default (Gray)` in our example) is highlighted in the palette with a dashed outline. The color is also shown in the `Preview` area. To select a new color, click on a different

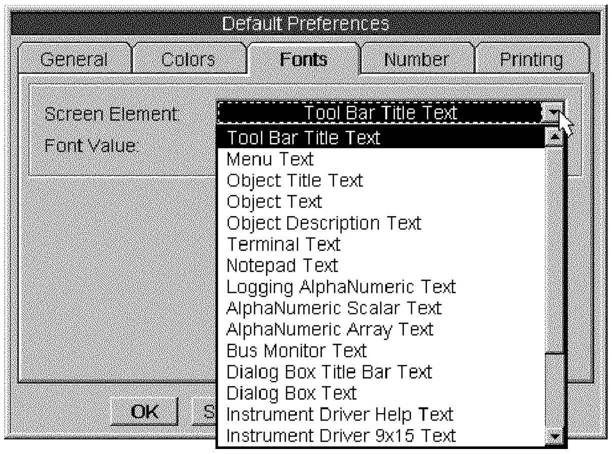
color in the palette, and then click on `OK` to return to the `AlphaNumeric Properties` dialog box.

5. Click on `OK` to accept the new color selection, or `Cancel` to reject it. The new color selection appears in the object and overrides the default color. The change affects only this particular object (not other `AlphaNumeric` objects), and is saved when you save the program.

To Change Default Fonts

You can change the default fonts using the `Default Preferences` dialog box:

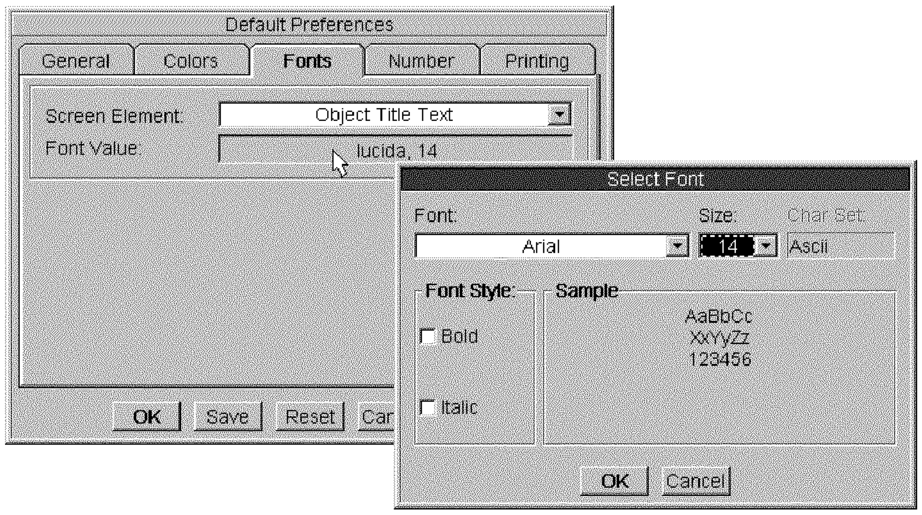
1. Select `File` $\Longrightarrow$ `Edit Default Preferences`, and select the `Fonts` tab in the dialog box.
2. Click on the `Screen Element` field or its arrow. The drop-down list displays the screen element choices:



Screen Element: Tool Bar Title Text
Font Value:

Tool Bar Title Text
Menu Text
Object Title Text
Object Text
Object Description Text
Terminal Text
Notepad Text
Logging AlphaNumeric Text
AlphaNumeric Scalar Text
AlphaNumeric Array Text
Bus Monitor Text
Dialog Box Title Bar Text
Dialog Box Text
Instrument Driver Help Text
Instrument Driver 9x15 Text

3. Click on a screen element in the list. For example, click on `Object Title Text`. The drop-down list disappears and the font value for the `Object Title Text` is displayed. Click on the `Font Value` button. The `Select Font` dialog box appears:



Screen Element: Object Title Text
Font Value: lucida, 14
Font: Arial
Size: 14
Char Set: Ascii
Font Style:
Sample
AaBbCc
XxYyZz
123456

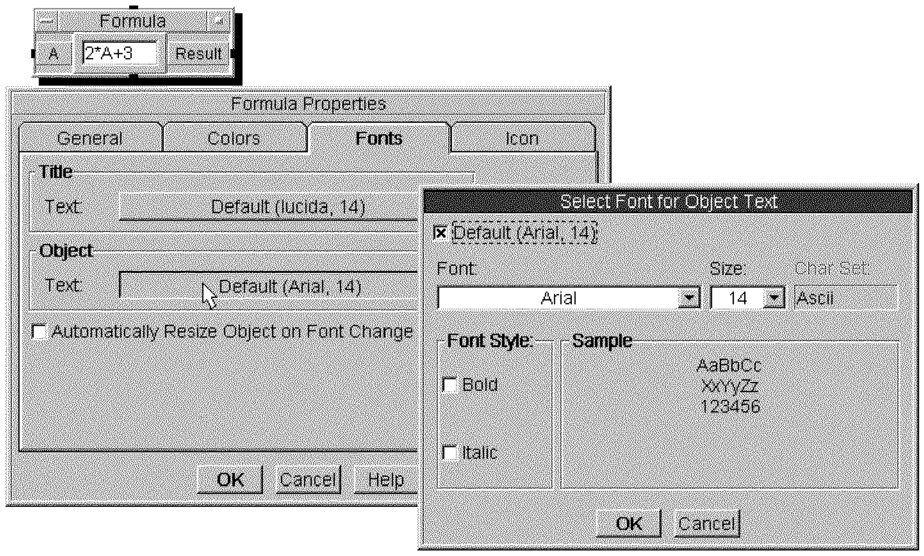
4. You can select a font name and font size from the two drop-down lists by clicking on the corresponding fields or their arrows. These lists allow you to select any font that is installed on your system. You can also select bold and italic by clicking on the appropriate check boxes. Click on `OK` to return to the `Default Preferences` dialog box.
5. Choose how you want to save your font selection:
   a. If you want to save your new font selection for `Object Title Text` as a permanent default font, click on `Save`. This will save the new default font in your `VEE.RC` or `.veerc` file. The new font will be the default for future HP VEE sessions (until you change it again).
   b. If you want to use your new font selection only for the current HP VEE session (until you select `File` $\Longrightarrow$ `New`, `Open`, or `Exit`), click on `OK`.
   c. If you don't want to save your new font selection at all, click on `Cancel`.

Note that you can make font selections for multiple screen elements before you click on `Save`, `OK`, or `Cancel`.

To Change Fonts for an Object

Every object has a properties dialog box that you can reach from the object menu. You can change fonts for the particular object using this dialog box. Let's use a `Formula` object in an example:

1. Add a `Formula` object to the work area (`Math` ⟹ `Formula`).
2. Select `Edit Properties` from the object menu, and then select the `Fonts` tab in the `Formula Properties` dialog box.
3. The fields that you can change depend on the particular class of object. For the `Formula` object you can change the font for the title text and the object text. Click on the `Text` button under `Object`, as shown below:



4. The `Select Font for Object Text` dialog box appears. To select a font or font size, click on the appropriate field or arrow. A drop-down list will display the choices. Also, you can select bold or italic by clicking on the appropriate checkbox. Once you have made your selections, click on `OK` to return to the `Formula Properties` dialog box.
5. If you click on the checkbox in front of `Automatically Resize Object on Font Change`, objects will be automatically resized as needed.
6. Click on `OK` to accept the new font selection, or `Cancel` to reject it. The new font selection appears in the object and overrides the default font. The change affects only this particular object (not other `Formula` objects), and is saved when you save the program.

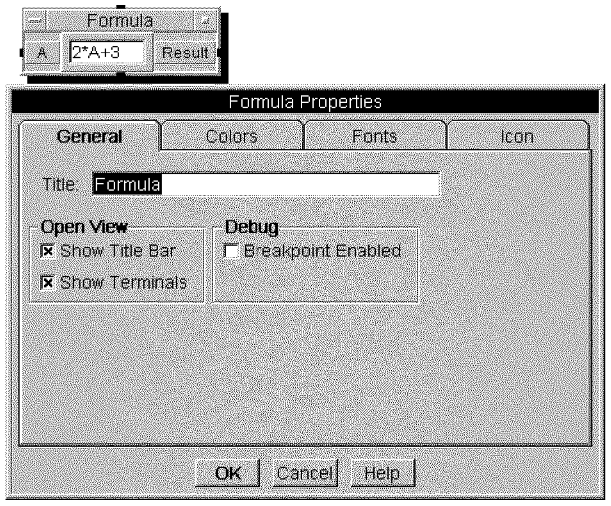
NOTE

If you are writing HP VEE programs to be used on more than one platform (Windows, HP-UX, and SunOS), be aware that each platform provides a different set of system fonts. Therefore, your program may appear slightly different on each platform.

To Change an Object or Work Area Title

You can change the title of any object, or of the work area, from the appropriate `Properties` dialog box. For a `Formula` object:

1. Select `Edit Properties` from the object menu (or double-click on the object title). The `Formula Properties` dialog box appears:



Title: Formula

Open View
☒ Show Title Bar
☒ Show Terminals

Debug
☐ Breakpoint Enabled

2. The title is already highlighted — just start typing and the old title is replaced.
3. Type in the new title and press Enter or Return.

The procedure is the same for the work area title, except that you use the **`Work Area Properties`** dialog box (**`File`** ⟹ **`Edit Properties`**).

Shortcut

To change the title of an object, double-click on the object title bar. The object `Properties` dialog box appears. Modify the title and press Enter or Return. To change the title of the work area, double-click on the tool bar. Then modify the title in the dialog box and press Enter or Return.

Using the Keyboard Shortcuts

You can use keyboard shortcuts to perform many common HP VEE tasks. Note that the keyboard shortcuts are *not* case sensitive. (A summary of the most often used keyboard shortcuts is given on the back cover of each manual.)

To Run, Pause, Step, or Continue an HP VEE Program

- Press CTRL+G to run an HP VEE program (shortcut for the **Run** button).
- Press CTRL+C to pause a running HP VEE program. (This is a shortcut for the **Stop** button. However, pressing CTRL+C twice will not stop the program permanently.)
- Press CTRL+T to step a paused HP VEE program (shortcut for the **Step** button).
- Press CTRL+V to continue a paused HP VEE program (shortcut for the **Cont** button).

To Open an HP VEE Program File

Press CTRL+O. You will be prompted for a file name to open (shortcut for `File` ⟹ `Open`).

To Save an HP VEE Program to a File

Press CTRL+S. The program will be re-saved to the current file, or you will be prompted for a file name (shortcut for **`File`** $\Longrightarrow$ **`Save`**).

To Save an HP VEE Program to a New File

Press CTRL+W. You will be prompted for a new file name, and the program will be saved there (shortcut for **`File`** $\Longrightarrow$ **`Save As`**).

To Exit HP VEE

Press CTRL+E to exit HP VEE (shortcut for **`File`** $\Longrightarrow$ **`Exit`**).

To Raise the Pop-Up Edit Menu

Press the *right* mouse button with the pointer over an empty part of the HP VEE work area. (This works for a `UserObject` work area as well.)

To Raise the Pop-Up Object Menu

Press the *right* mouse button with the pointer over any part of an object.

To Delete an Object

Press CTRL+D with the pointer over the object to be deleted, but not over a terminal. (If the pointer is over a terminal, the terminal will be deleted instead.)

To Delete a Line

Press and hold Shift+CTRL, and click the left mouse button with the pointer over or near the line you want to delete (shortcut for `Edit` ⟹ `Delete Line`).

To View Data on a Line

Press and hold Shift, and click the left mouse button with the pointer over or near the line you want to probe (shortcut for `Edit` ⟹ `Line Probe`). Information about the last container of data on the line is displayed.

To Select Objects

Press and hold CTRL, and click the left mouse button with the pointer over an object (shortcut for `Edit` ⟹ `Select Objects`). The object becomes selected (highlighted with a shadow) if it is not currently selected. (If the object is currently selected, it becomes un-selected.) You can use CTRL+*left mouse button* to select multiple objects. Hold down CTRL and click on each object that you want to select.

To Add a Data Input Terminal to an Object

Press CTRL+A with the pointer in the input terminal area to insert another data input terminal at the pointer location (shortcut for: *object menu* ⟹ `Terminals` ⟹ `Add Data Input`). *If you have just edited a field in the object, you may have to click the left mouse button first, and then press* CTRL+A.

To add a terminal using CTRL+A, `Show Terminals` must be active and the pointer must be over the input terminal area at the left side of the object.

To Add a Data Output Terminal to an Object

Press CTRL+A with the pointer in the output terminal area to insert another data output terminal at the pointer location (shortcut for: *object menu* ⟹ `Terminals` ⟹ `Add Data Output`). *If you have just edited a field in the object, you may have to click the left mouse button first, and then press* CTRL+A.

To add a terminal using CTRL+A, `Show Terminals` must be active and the pointer must be over the output terminal area at the right side of the object.

To Delete a Terminal from an Object

Press CTRL+D with the pointer over the terminal to be deleted. (A shortcut for: *object menu* ⟹ `Terminals` ⟹ `Delete Input` or *object menu* ⟹ `Terminals` ⟹ `Delete Output`.) *If you have just edited a field in the object, you may have to click the left mouse button first, and then press* CTRL+D.

To delete a terminal using CTRL+D, `Show Terminals` must be active and the pointer must be over the terminal to be deleted. Otherwise, CTRL+D will delete the object itself.

To Refresh the HP VEE Window

Press CTRL+R to refresh the HP VEE window.

To Print or Copy the HP VEE Window

On a UNIX workstation: Press Shift+Print to print the HP VEE window with the current options specified in `Printer Config` (shortcut for `File` ⟹ `Print Screen`).

On a PC: To copy the HP VEE window to the MS Windows Clipboard, press Alt+Print Screen with HP VEE as the active window. From the Clipboard, you can paste the HP VEE window into other Windows applications such as MS Windows Paintbrush.

Note that these shortcuts allow you to capture the HP VEE screen even when a dialog box is present.

To Cancel an Edit in an Entry Field

Press (Esc) to cancel an edit in an entry (type-in) field.

To Cancel Edits in a Dialog Box

Press (Esc) to cancel all edits in a dialog box. (This is equivalent to clicking on the `Cancel` button.)

To Move the Text Cursor

In alphanumeric text entry areas (for example, the `Note Pad` object) use the arrow keys ((▲), (▼), (◀), (▶)) to move the cursor in the direction of the arrow.

To Scroll the Work Area

With the pointer anywhere within the HP VEE work area:

- Press (Shift)+(▲), (Shift)+(▼), (Shift)+(◀), or (Shift)+(▶)) to scroll the work area in the direction of the arrow key.
- Press (Page Down) (Windows) or (Next) (UNIX) to scroll the work area up one screen.
- Press (Page Up) (Windows) or (Prev) (UNIX) to scroll the work area down one screen.

- Press Shift+Page Down (Windows) or Shift+Next (UNIX) to scroll the work area left one screen.
- Press Shift+Page Up (Windows) or Shift+Prev (UNIX) to scroll the work area right one screen.
- Press Home (Windows) or ↖ (UNIX) to move the upper left corner of the program to the upper left corner of the work area.
- Press Shift+Home (Windows) or Shift+↖ (UNIX) to move the lower right corner of the program to the lower right corner of the work area.

To Edit a Transaction

The following are some shortcuts that you can use when the mouse pointer is positioned over an object containing *transactions* (for example, the `Sequencer` or `To File` objects). For further information about transactions, refer to the chapter "Using Transaction I/O" in *HP VEE Advanced Programming Techniques*.

- Press CTRL+K with the pointer over a transaction to cut the transaction to the "cut-and-paste" buffer (*object menu* ⟹ `Cut Trans`).
- Press CTRL+Y to paste a transaction from the "cut-and-paste" buffer at the pointer location (*object menu* ⟹ `Paste Trans`).
- Press CTRL+O to insert a transaction at the pointer location (*object menu* ⟹ `Insert Trans`).
- Press CTRL+N to move to the next transaction.
- Press CTRL+P to move to the previous transaction.
- Press CTRL+X to step to the next transaction (`Sequencer` only). (`Sequencer` *object menu* ⟹ `Step Trans`.)

To Navigate the HP VEE Menu

You can navigate the HP VEE menu by using the keyboard menu accelerators. Each menu item has an underlined letter, which acts as an accelerator in conjunction with the Alt key on a PC or the Extend Char key on a UNIX workstation. The keyboard accelerator letters are case insensitive. In other words, even though the underlined letter may be capitalized, you can use the lower case letter as the accelerator. Let's look at an example.

In the `Device` menu the `D` is underlined (but `d` works fine as the accelerator) and in the `Data` menu the `t` is underlined:

- On the PC, press Alt+d to select the `Device` menu or Alt+t to select the `Data` menu.
- On the UNIX workstation, press Extend Char+d to select the `Device` menu, or Extend Char+t to select the `Data` menu.

Each sub-menu has its own accelerators, which you can use to navigate further down the menu tree. For example, to select `Data` $\Longrightarrow$ `Real Slider`:

- On the PC, press Alt+t and then r.
- On the workstation, press Extend Char+t and then r.

To Navigate an Operator Interface Panel

If you or another HP VEE programmer have created an operator interface panel view in an HP VEE program, you can navigate the panel with the following keys:

- Press (Tab) and (Shift)+(Tab) to move from one field in the operator interface panel to the next. The (Tab) key moves focus forward, and (Shift)+(Tab) moves focus backward from field to field. When a type-in field has focus (indicated by an inverse highlight), you can edit it. When a button has focus (indicated by a dashed outline), you can "push" it as described below.
- Press the space bar to "push" a button that currently has focus.
- Use the appropriate alphanumeric keys to edit a type-in field that currently has focus. Press (Enter) or (Return) to complete and accept the edit, or press (Esc) to abort the edit.
- Press (Enter) or (Return) to "push" the default button in the operator interface panel, if one has been defined by the programmer.

Operator interface panel views and how to use them are covered in detail in *Building an Operator Interface with HP VEE*.

3

Working with Data

Working with Data

This chapter gives some useful techniques for inputting and analyzing data using HP VEE.

Inputting Data

HP VEE provides several types of objects that allow you to enter data. Let's begin by looking at the conventions HP VEE recognizes for entering a number.

To Enter a Number

You can enter a number in any HP VEE numeric-entry field using the following conventions. (Of course, you can enter any arbitrary string of characters in a text-entry field.)

- Ordinary numeric entry:

`1001`	
`1020768`	*But not* `1,020,768`
`12.12`	
`−1.223`	

- Exponential numeric entry:

`1.23E3`	*Equivalent to* $1.23\text{x}10^{3}$ *or* 1230
`6.0225E23`	*Equivalent to* $6.0225\text{x}10^{23}$
`6.6256E−27`	*Equivalent to* $6.6256\text{x}10^{-27}$

- Numeric entry using the ISO abbreviations:

`1.23k`	*Equivalent to* $1.23\text{x}10^{3}$ *or* 1230
`3.61M`	*Equivalent to* $3.61\text{x}10^{6}$
`2.3n`	*Equivalent to* $2.3\text{x}10^{-9}$

The following table lists the ISO abbreviations recognized by HP VEE for entering a number.

Table 3-1. ISO Numeric Abbreviations

Abbreviation	Suffix	Multiple	Abbreviation	Suffix	Multiple
X	exa	10^{18}	m	milli	10^{-3}
P	peta	10^{15}	u	micro	10^{-6}
T	tera	10^{12}	n	nano	10^{-9}
G	giga	10^{9}	p	pico	10^{-12}
M	mega	10^{6}	f	femto	10^{-15}
k or K	kilo	10^{3}	a	atto	10^{-18}

The hypen (−) is used as a minus sign and the period (`.`) is used as a decimal point. You can include a plus sign (`+`) in a numeric entry field if desired. However, the comma (`,`) is not allowed as a separator.

HP VEE may change the form of the number you enter to another, equivalent form. For example:

`+1.23`	*Becomes* `1.23` *(plus signs are dropped)*
`6.3E−9`	*Becomes* `6.3n`
`6.0225E23`	*Becomes* `602.25E21` *(in the standard number format)*

You can change the number format that HP VEE uses to display numbers. Refer to the following section for further information.

To Change the Number Format

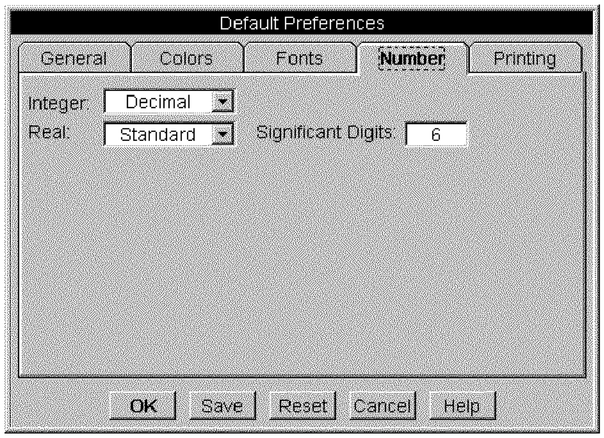
You can change the number format that HP VEE uses to display numbers:

1. You can change the global number format that HP VEE uses, by default, for all objects that display numbers.
2. You can change the number format for certain individual objects that display numbers.

To Change the Global Number Format:

To change the global number format for an HP VEE program:

1. Select `File` ⟹ `Edit Default Preferences`, and then click on the `Number` tab:



Integer: Decimal
Real: Standard
Significant Digits: 6

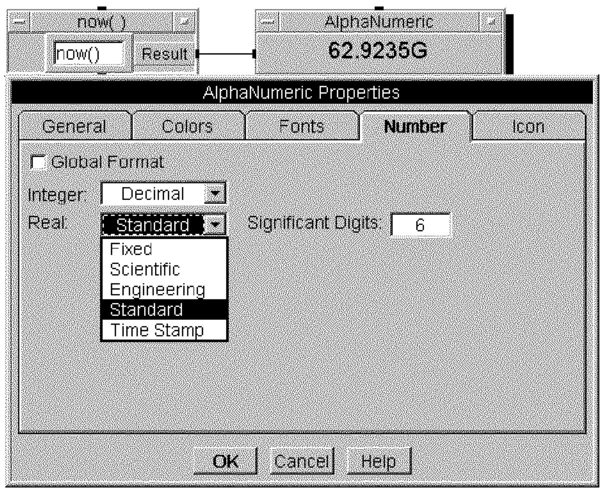
2. Select the desired number format by changing the appropriate fields:
   - `Integer` determines the number base for integers: `Decimal`, `Octal`, `Hexadecimal`, or `Binary`. (Click on the field or its arrow for the drop-down list.)
   - `Real` determines the format of real numbers: `Standard`, `Fixed`, `Scientific`, or `Engineering`. (Click on the field or its arrow for the drop-down list.)
   - `Sig digits` determines the number of significant figures. (Type in the desired number.)
3. Click on `Save` if you want the changes to be saved as new defaults (in `VEE.RC` or `.veerc`). Or click on `OK` if you want to use the changes for this HP VEE session only (until you select `File` ⟹ `New`, `Open`, or `Exit`).

To Change the Number Format for Certain Objects:

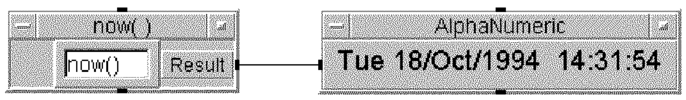
You can select a number format independently for certain objects. These objects include the numeric slider and constant objects, and the `AlphaNumeric`, `Logging AlphaNumeric`, and `Meter` display objects. In each case, the selected number format overrides the global number format, but only for the particular object.

Let's look at an example. Suppose that you connect an `AlphaNumeric` display object to the output of a `Now()` object. By default the time is displayed as something like `62.9183G` (in seconds). To change this to the `Time Stamp` format:

1. Select `Edit Properties` from the object menu of the `AlphaNumeric` object. Then click on the `Number` tab.
2. Now click on the check-box to turn off `Global Format`. Once you have done that, you can click on the `Real` field (or its arrow) to show the drop-down menu, as shown below:



3. Select the `Time Stamp` format. Other choices appear. By default, the `Date & Time` 24-hour format is used. Click on `OK` and the time is displayed in that format:



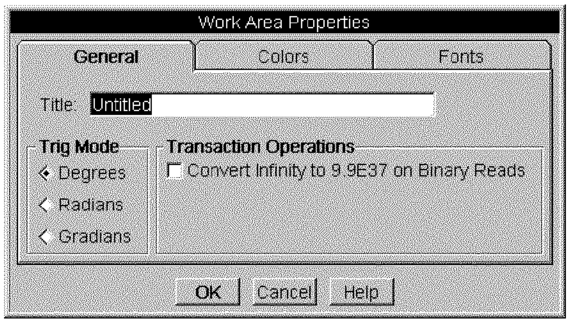
Tue 18/Oct/1994 14:31:54

To Change the Trig Mode

In HP VEE, trigonometric values may be entered and displayed in degrees, radians, or gradians, depending on the **trig mode**.

To change the trig mode:

1. Select `File` ⟹ `Edit Properties`. The `Work Area Properties` dialog box appears:



Title: Untitled

Trig Mode: [✓ Degrees] Radians Gradians

Convert Infinity to 9.9E37 on Binary Reads ☐

2. Select the desired trig mode (**Degrees**, **Radians**, or **Gradians**), and then click on **OK**.

The new trig mode is active for the duration of the current HP VEE session. When you restart HP VEE, the trig mode reverts to the default, (**Degrees**).

NOTE

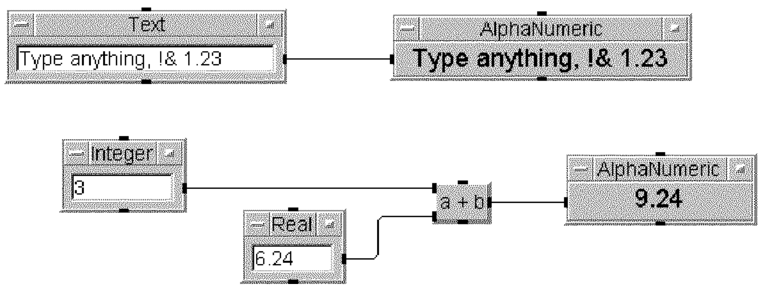
All internal calculations are done in radians. You'll get slightly faster performance if you set the trig mode to **Radians** because HP VEE won't have to make the conversion to radians and back.

To Use a Data Constant

HP VEE provides several types of data constants. To use a *scalar* data constant:

1. Select a data constant object (**Data** $\Longrightarrow$ **Constant** $\Longrightarrow$). The choices are **Text**, **Integer**, **Real**, **Coord**, **Complex**, **PComplex**, **Date/Time**, and **Record**.
2. Enter the appropriate type of data in the constant object. The most common types are the **Text**, **Integer**, and **Real** constant objects:
   - In the **Text** constant, you can enter any text string (the maximum length is 32,767 characters).
   - In the **Integer** constant, you can enter any 32-bit, two's-complement integer (the range is −2147483648 through +2147483647).
   - In the **Real** constant, you can enter any 64-bit real number (the range is ±1.7976931348623157E308).

In the example below, the **`Text`**, **`Integer`**, and **`Real`** constant objects are connected into HP VEE program threads:



Note that data types are automatically converted as needed by the object receiving the data. In this example the integer `3` is converted into a real value and added to `6.24`. The `A+B` object outputs a real value, `9.24`. Refer to Appendix A for further information about data type conversion.

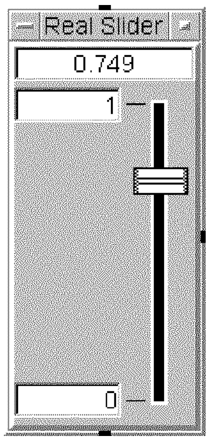
You can also create one-dimensional *arrays* using the data constant object. Refer to "To Create an Array Constant", later in this chapter, for further information.

To Use a Data Slider

HP VEE provides two types of data **slider** objects: the **`Real Slider`** and **`Integer Slider`**. These objects allow you to input a value in your program by moving a linear slider.

To use a **`Real Slider`**:

1. Select **`Data`** $\Longrightarrow$ **`Real Slider`** and place the object in the work area.
2. By default the **`Real Slider`** object has a range of `0` through `1`. By moving the slider you can change the setting, which is displayed in the field at the top (`0.749` in the example that follows).

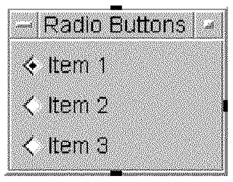


3. To change the range, click on the upper or lower limit field and change the value. For example, click on the upper limit of `1` and change the value to `2`. Now the slider ranges from `0` through `2`.

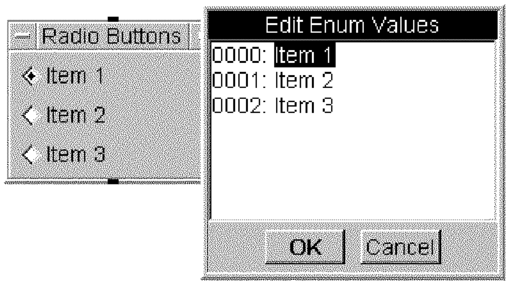
To Use a Data-Selection Object

HP VEE provides five types of data-selection objects: **`Radio Buttons`**, **`Cycle Button`**, **`List`**, **`Drop-Down List`**, and **`Pop-Up List`**. For example, to use a **`Radio Buttons`** object:

1. Select **`Data`** $\Longrightarrow$ **`Selection Controls`** $\Longrightarrow$ **`Radio Buttons`** and place the object in the work area:

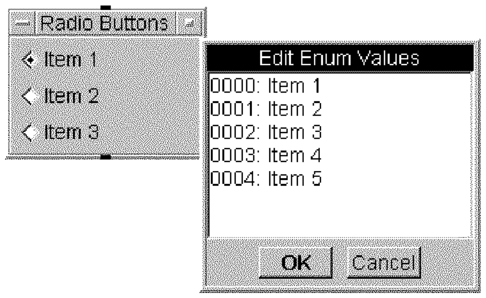


2. Note that the default radio button selections are **`Item 1`**, **`Item 2`**, and **`Item 3`**. To change these selections, select **`Edit Enum Values`** from the object menu. The **`Edit Enum Values`** dialog box appears:



```
0000: Item 1
0001: Item 2
0002: Item 3
```

3. Edit the values as needed. Use (Tab) (or (▼)) to advance, (Shift)+(Tab) (or (▲)) to back up among the fields. You can replace the text in each field or, to add fields, tab past the last field and type in additional text:



```
0000: Item 1
0001: Item 2
0002: Item 3
0003: Item 4
0004: Item 5
```

4. When you are finished editing the fields, click on **`OK`**.

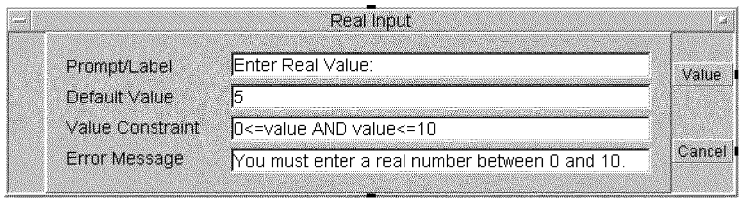
The same techniques are used to edit the fields for all five types of data-selection objects. In fact, by selecting **`Edit Properties`** from the object menu you can change the format from one type of object to another. For example, you can change a **`Radio Buttons`** object into a **`Drop-Down List`** object, and so forth.

To Use a Data Input Dialog Box

HP VEE provides six built-in dialog box objects under `Data` ⟹ `Dialog Box` (`Text Input`, `Integer Input`, `Real Input`, `Message Box`, `List Box`, and `File Name Selection`). You can use the `Text Input`, `Integer Input`, and `Real Input` objects to prompt for data input.

To use a `Real Input` pop-up dialog box:

1. Add a `Real Input` dialog box object to the work area (`Data` ⟹ `Dialog Box` ⟹ `Real Input`):



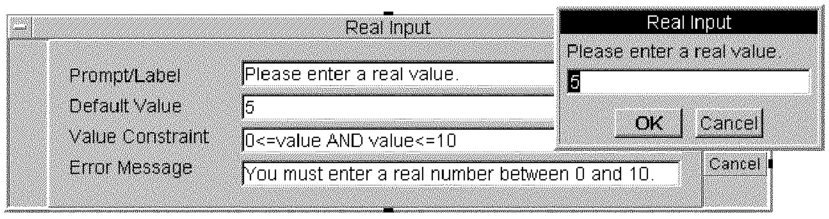
Prompt/Label	Enter Real Value:
Default Value	5
Value Constraint	0<=value AND value<=10
Error Message	You must enter a real number between 0 and 10.

When the `Real Input` object operates, it displays a pop-up dialog box requesting a real number from the user. You can set the following parameters:

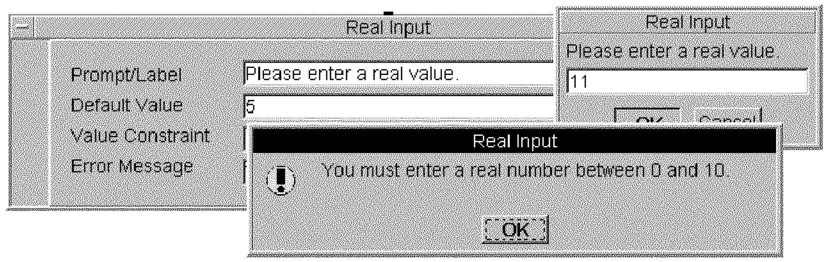
- `Prompt/Label` determines the message displayed in the dialog box.
- `Default Value` determines the optional default value to be displayed in the dialog box.
- `Value Constraint` determines the range or other test that the value entered must match.
- `Error Message` determines the error message to be displayed if the user responds with data that fails the test in `Value Constraint`.

2. Change the `Prompt/Label` field to `Please enter a real value.`

3. Run the program. The pop-up dialog box appears:



4. Enter `11` in the dialog box and click on `OK`. The following error message appears:


Real Input
Prompt/Label
Please enter a real value.
Default Value
5
Value Constraint
Error Message
Real Input
Please enter a real value.
11
Real Input
You must enter a real number between 0 and 10.
OK

5. Click on `OK` in the error message box. Now enter a number in the range `0` through `10` in the `Real Input` dialog box, and click on `OK`. The number is accepted and output on the `Value` terminal.

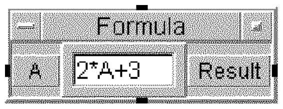
The `Integer Input` and `Text Input` dialog box objects work in much the same way. For further information on using dialog boxes, refer to *Building an Operator Interface with HP VEE.*

Using Mathematical Expressions

This section describes some useful techniques for entering and solving mathematical expressions in HP VEE. For a detailed summary of the HP VEE mathematical conventions, refer to “Mathematically Processing Data” in Chapter 3 of the *HP VEE Reference* manual.

To Enter a Formula

The basic mathematical object in HP VEE is the **Formula** object:



The **Formula** object, when selected from the menu and placed in the work area, includes the expression **2*A+3** (as an example) and has one data input terminal, **A**. *Note that variable* **A** *in the expression corresponds to data input terminal* **A**.

Whenever HP VEE evaluates an expression in an object, each variable in the expression is replaced by data on the input terminal with the same name as the variable. However, variable names and terminal names are not case-dependent (upper-case and lower-case letters my be used interchangeably). For example, the variable **a** in the expression **a+1** is recognized as corresponding to terminal **A**.

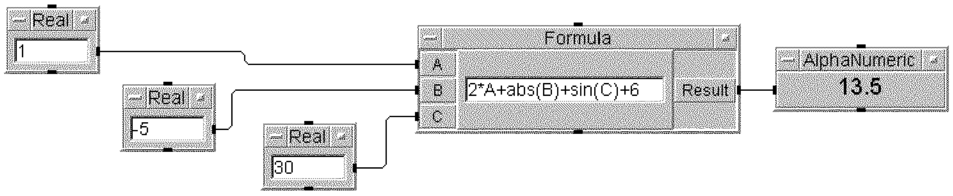
NOTE

In general, if a named variable in an expression does not have a corresponding input terminal, HP VEE cannot evaluate the expression and returns an error. However, there is an exception to this. A global variable requires no corresponding data input terminal. Global variables are covered in "Working with Global Variables" at the end of this chapter.

To use the **Formula** object:

1. Add the **Formula** object to the work area (**Math** ⟹ **Formula**).
2. Click on the expression field in the **Formula** object and modify the expression as desired. For example, enter the expression **2*A+abs(B)+sin(C)+6**. Refer to Chapter 3 in the *HP VEE Reference* manual, or to the online help, for the HP VEE math syntax rules.
3. Add data input terminals as required. (Place the pointer in the data input terminal area and press Ctrl+A.) Delete any unused data input terminals. (Place the pointer over the terminal and press Ctrl+D.)
4. Connect all of the data input terminals to appropriate sources of data, and the **Result** data output terminal to a display object, to complete your program.

The following is a simple example of an HP VEE program that uses the **Formula** object to evaluate an expression and display the result:



To Use the Math and AdvMath Objects

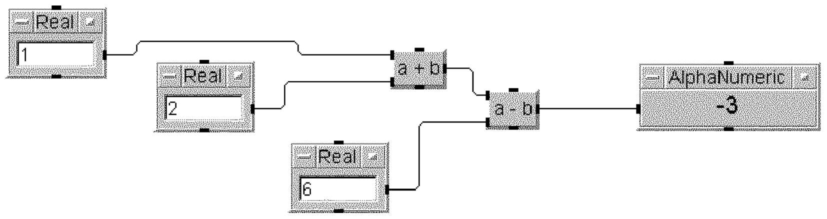
For your convenience, over 150 mathematical objects are provided in the `Math` and `AdvMath` menus. However, all of these objects are really `Formula` objects that have been pre-configured with a descriptive title, an expression, and the appropriate data input terminals. Because these objects are really `Formula` objects, you can modify them as follows:

1. Change the expression to any valid HP VEE expression.
2. Change the title to describe the new expression.
3. Add or delete data input terminals as required.

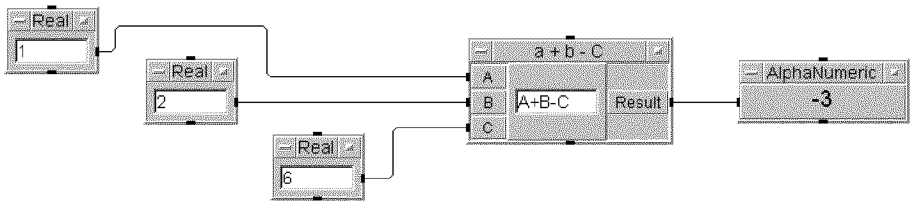
Here are some tips for using these objects:

1. Don't use several math and advanced objects to "wire" an expression. Instead, combine the mathematical operators and functions into a single expression in a single object. *HP VEE programs run faster when fewer objects are used.*

   Don't do this:



   Do this instead:



2. When you modify the expression in a `Math` or `AdvMath` object, make sure that you both change the expression and add the necessary data input terminals.
3. If you are going to minimize the object to its icon, be sure to change the title to describe what the new mathematical expression does.

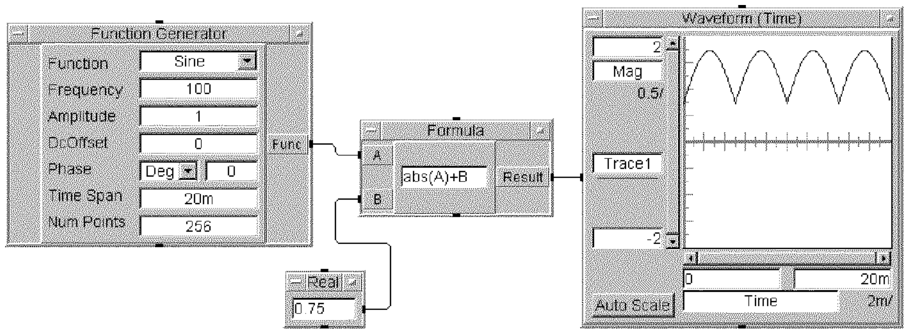
NOTE

You can change the names of the data input terminals for the the `Math` and `AdvMath` objects in the usual way. (Just double-click on the terminal and type in a new name.) However, if you change the terminal names, make sure you change the corresponding variable names in the expression as well.

To Use Mixed Data Types in an Expression

Because HP VEE automatically converts data types as needed, you can solve expressions involving mixed data types. In general, you don't need to do anything special. Just enter the expression in an object such as `Formula` and connect the appropriate data sources to the data input terminals.

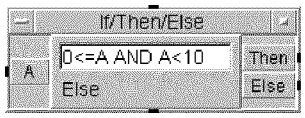
For example, in the following program the `Formula` object receives waveform data on input terminal `A` and real data on input terminal `B`.



The **`Formula`** object evaluates the expression **`abs(A)+B`** by adding the real value **`0.75`** to the absolute value of each of the 256 points in the waveform. All of the data type conversion occurs automatically.

To Use an Expression in an If/Then/Else Object

HP VEE can evaluate mathematical expressions and branch program flow according to the result. The basic conditional object in HP VEE is the **`If/Then/Else`** object:

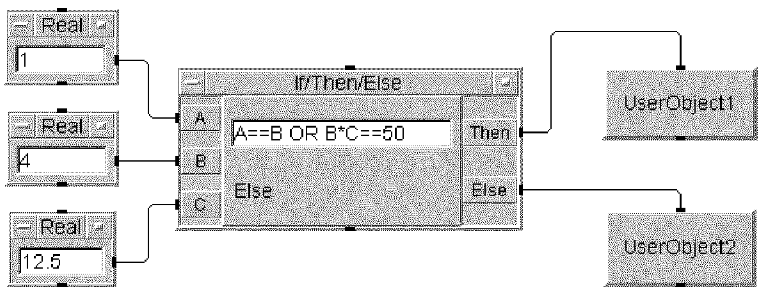


The default expression in the **`If/Then/Else`** object is **`0<=A AND A<10`**. However, any conditional expression can be used. The expression can be as simple as **`A==B`**. In fact, the **`If A==B`** object and the other objects found under **`Flow`** $\Longrightarrow$ **`Conditional`** are really just **`If/Then/Else`** objects pre-configured with expressions, descriptive titles, and data input terminals.

To use the **If/Then/Else** object:

1. Add the **If/Then/Else** object to the work area (**Flow** ⟹ **If/Then/Else**).
2. Click on the expression field in the **If/Then/Else** object and modify the expression as desired. For example, enter the expression **A==B OR B>=C**.
3. Add data input terminals as required. (Place the pointer in the data input terminal area and press Ctrl+A.) Delete any unused data input terminals. (Place the pointer over the terminal and press Ctrl+D.)
4. Connect all of the data input terminals and one or both of the data output terminals to complete your program. If the expression is "true," the result of the expression (generally a **1**) is output on the **Then** terminal. If the expression is "false" a **0** is output on the **Else** terminal.
5. If desired, you can add **Else If** conditions to the object, as described in "To Branch within an HP VEE Program" in Chapter 2.

In the following program, if the expression **A==B OR B*C==50** is true, the **Then** output terminal activates **UserObject1**. If the expression is false, the **Else** output terminal activates **UserObject2**.

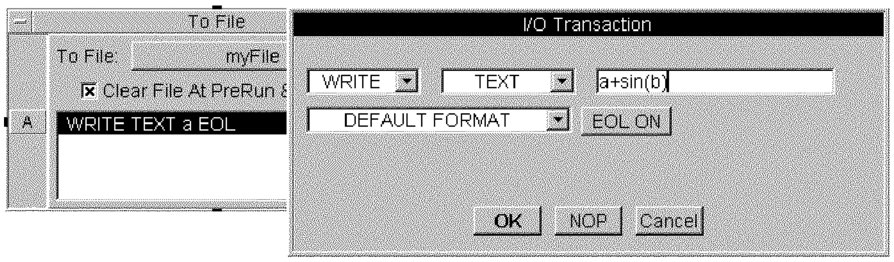


To Use an Expression in a Transaction

HP VEE mathematical expressions can be evaluated and used in an HP VEE I/O transaction. Several HP VEE objects are transaction based, including the `To File` and `From File` objects, which are covered in detail in Chapter 4.

Let's look at an example of evaluating an expression in a `To File` transaction:

1. Add a `To File` object to the work area (`I/O` ⟹ `To` ⟹ `File`).
2. Double-click on the transaction `WRITE TEXT a EOL`. The `I/O Transaction` dialog box appears.
3. Click on the default expression, `a`, and change the expression to `a+sin(b)`:

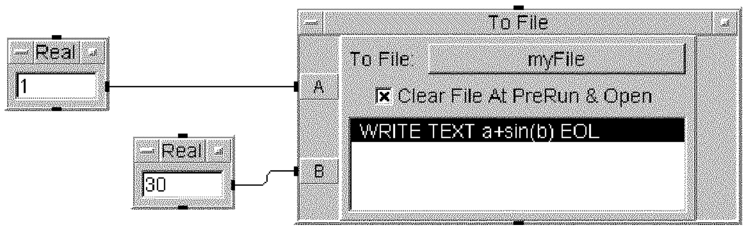


WRITE TEXT a EOL

WRITE TEXT a+sin(b)
DEFAULT FORMAT EOL ON

4. Click on `OK` to save the new transaction (`WRITE TEXT a+sin(b) EOL`).
5. Add a `B` data input terminal to the object, and connect the data input terminals.

In the following program, the `To File` object adds the value of `a` (`1`) to the value of `sin(b)` (the sine of 30 degrees is `0.5`), and outputs the result `1.5` to the file named `myFile`.



1

30

To File: myFile

☒ Clear File At PreRun & Open

WRITE TEXT a+sin(b) EOL

For further information about transactions, refer to the "Using Transaction I/O" chapter in *HP VEE Advanced Programming Techniques*.

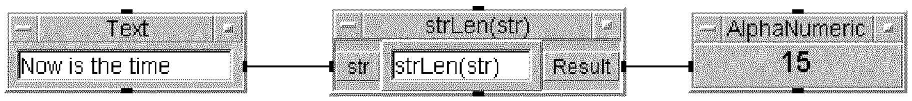
Working with Strings

This section shows how to use some of the HP VEE string-manipulation features found under **Math** $\Longrightarrow$ **String**, and how to use some other techniques to process string data.

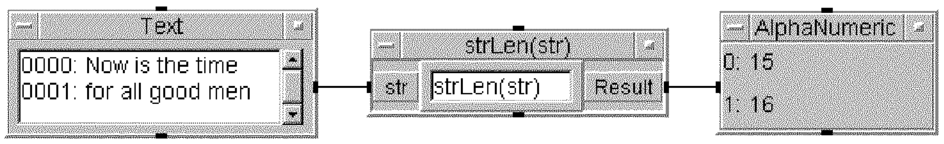
To Find the Length of a String

To find the length of a string using the **strLen(str)** function:

1. Add the **strLen(str)** object to work area (**Math** $\Longrightarrow$ **String** $\Longrightarrow$ **strLen(str)**).
2. Connect the source of string data to the **str** data input terminal. Connect the **Result** output as desired.
3. Run the program. When the **strLen(str)** object operates, the length of the string is output on the **Result** terminal. For example:



The **strLen(str)** object outputs the length of the string in bytes as an Int32 number. If **str** is a one-dimensional string array, **strLen(str)** returns the length of each element of the array:



To Obtain a Substring

You can use the `strFromThru(str,from,thru)` function to obtain a substring, where:

`str` is the original string.

`from` is the position where the substring starts.

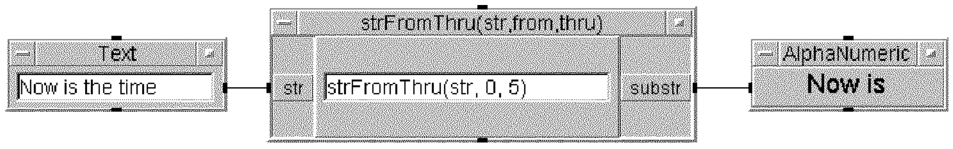
`thru` is the position where the substring stops.

All character positions in a string are zero-based. That is, the first character is at position `0`.

To obtain a substring from a string:

1. Add the `strFromThru(str,from,thru)` object to the work area (`Math` ⟹ `String` ⟹ `strFromThru(str,from,thru)`).
2. Connect the source of string data to the `str` input terminal.
3. Specify the `from` and `thru` values:
   - Connect data to the `from` and `thru` input terminals, or
   - Change the expression to explicitly specify `from` and `thru`. For example, the expression `strFromThru(str,0,5)` returns characters `0` through `5` (the first six characters) of string `str`. Remember to delete the `from` and `thru` terminals if you do this.

   The program below returns the first six characters of the string:

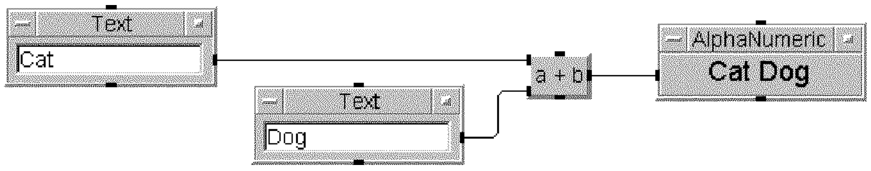


You can also use the `strFromLen(str,from,len)` function to obtain a substring. In this case, `len` specifies the length of the substring.

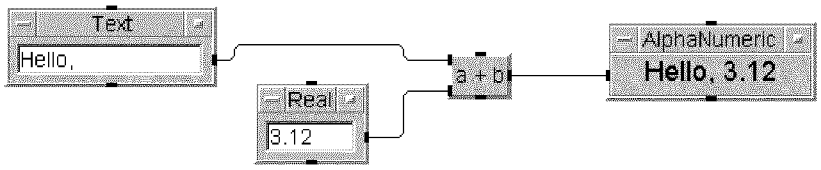
To Concatenate Strings

You can use addition (+) and multiplication (*) in HP VEE to concatenate strings. Here are some examples of what you can do:

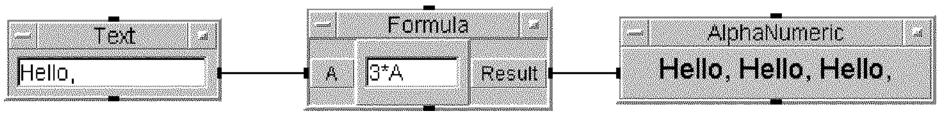
- You can concatenate two strings:



- You can concatenate a string and a number (the number is converted to a string):

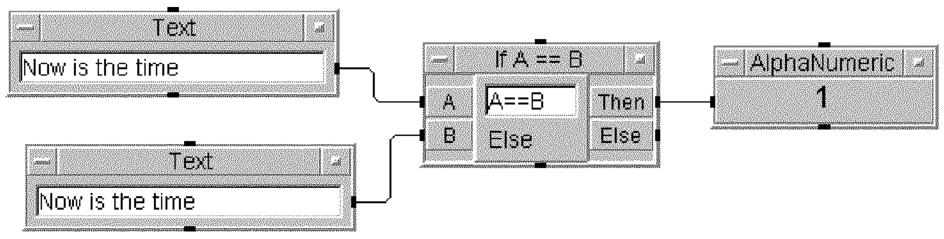


- You can multiply a number and a string to repeatedly concatenate the string:



To Compare Two Strings for Equality

You can use the HP VEE conditional tests to compare two strings. For example, the following program compares two strings for equality:



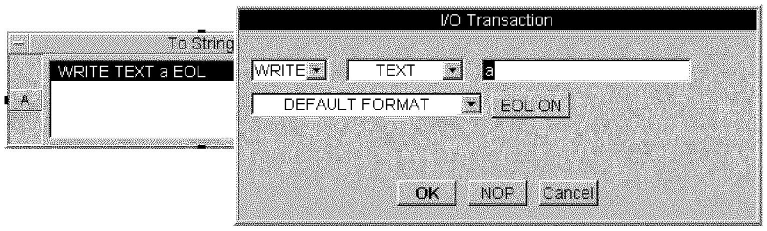
If the two strings are *exactly* the same (including spaces), the result of the expression (in this case, a `1`) is output on the `Then` terminal. (Otherwise, a `0` is output on the `Else` terminal.) Note that this comparison is case-sensitive (`Now` is not equivalent to `now`).

To Convert a Number into a String

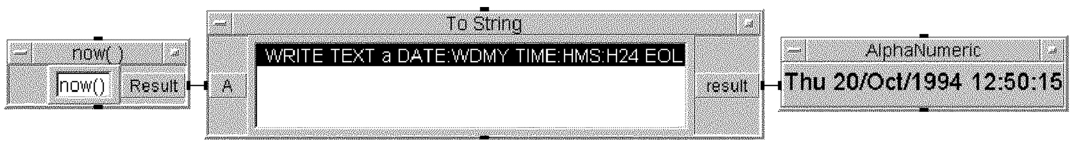
You can convert a number into a string using `To String`. As an example, let's convert the current time in seconds into a string in the time stamp format:

1. Add a `To String` object to the work area (`I/O` $\Longrightarrow$ `To` $\Longrightarrow$ `String`). Double-click on the icon to show the open view object.

2. Double-click on the default transaction **WRITE TEXT a EOL**. The **I/O Transaction** dialog box appears:



3. Click on **Default Format** (or its arrow) to show the drop down menu. Select **Time Stamp Format** and click on **OK**.
4. Connect a **now()** object to the data input terminal and a **AlphaNumeric** display to the data output terminal of **To String**.
5. Run the program. The **now()** object outputs the time in seconds (something like 62.9185G). This value is converted to the time stamp format and displayed, as shown below:



WRITE TEXT a DATE:WDMY TIME:HMS:H24 EOL

Thu 20/Oct/1994 12:50:15

For further information about transactions, refer to the "Using Transaction I/O" chapter in *HP VEE Advanced Programming Techniques*.

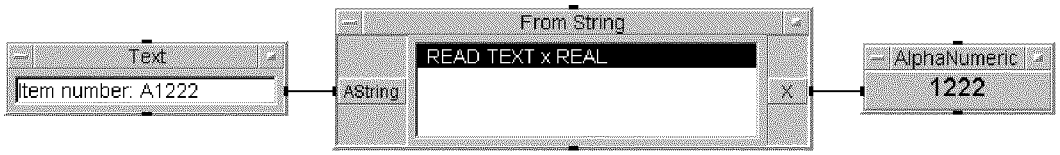
NOTE

You can convert an integer (0 to 255) to an ASCII character by using the **intToChar(a)** function. This does not work for two-byte (for example, Kanji) characters.

To Convert a String into a Number

If a string includes numeric information that can be read as a number, you can use **From String** to extract the number. Let's look at an example:

1. Add a **From String** object to the work area (**I/O** ⟹ **From** ⟹ **String**). Double-click on the icon to obtain the open view object.
2. The default transaction is **READ TEXT a REAL**, which reads the string as text and outputs a real number if possible. Connect a **Text** constant object to the data input terminal, and an **AlphaNumeric** display to the data output terminal.
3. Enter the string **Item number: A1222** in the **Text** constant and run the program. The real number value **1222** is output to the display:



NOTE

You can convert an ASCII character to an integer by using the **charToInt(a)** function. This does not work for two-byte (for example, Kanji) characters.

Working with Arrays

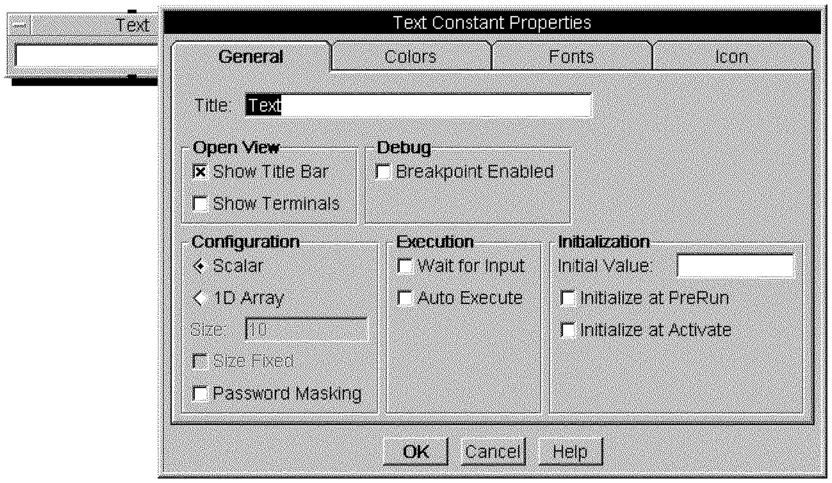
HP VEE provides several features for processing arrays. In fact, HP VEE is optimized for array mathematics. *To increase the performance of your HP VEE programs, use arrays whenever possible if you are processing large amounts of data.*

To Create an Array Constant

You can create one-dimensional arrays using the data constant objects. Use this technique when you want to manually set each value in a one-dimensional array.

For example, to create a one-dimensional text array with nine elements:

1. Add a `Text` constant object to the work area (`Data` ⟹ `Constant` ⟹ `Text`).
2. Select `Edit Properties` from the object menu. The `Text Constant Properties` dialog box appears:



Title: Text

Open View
[✓ Show Title Bar]
Show Terminals

Debug
Breakpoint Enabled

Configuration
[✓ Scalar]
1D Array
Size: 10
Size Fixed
Password Masking

Execution
Wait for Input
Auto Execute

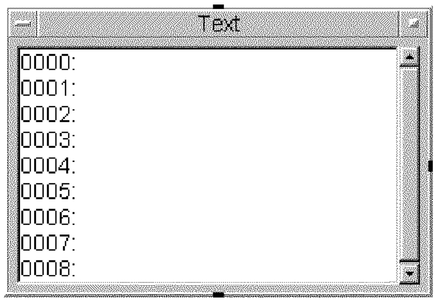
Initialization
Initial Value:
Initialize at PreRun
Initialize at Activate

3. By default the `Text` constant object is configured as a `Scalar`. Click on `1D Array` under `Configuration`, and then click on the `Size` field and change it to `9`.

NOTE

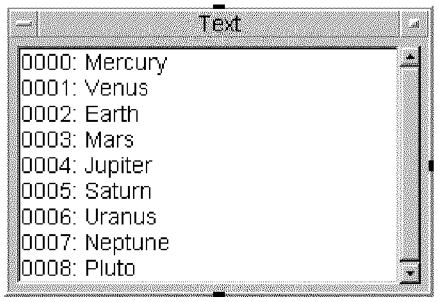
If you want your array to have a fixed number of elements, click on the `Size Fixed` checkbox as well. If you want to be able to add additional elements on at the end, leave `Size Fixed` unselected.

4. Click on `OK`. The `Text` constant object should now appear as follows. Note that the nine elements run from `0000:` through `0008:`. All HP VEE arrays are zero-based.



```
0000:
0001:
0002:
0003:
0004:
0005:
0006:
0007:
0008:
```

5. Click to the right of `0000:` and type in the first element of your array. Use (Tab) (or (▼)) to advance, (Shift)+(Tab) (or (▲)) to back up among the array elements. Type in text strings for each of the elements. The following is an example of a nine-element, one-dimensional text array.



```
0000: Mercury
0001: Venus
0002: Earth
0003: Mars
0004: Jupiter
0005: Saturn
0006: Uranus
0007: Neptune
0008: Pluto
```

You can convert any constant object (**`Integer`**, **`Real`**, and so forth) into an array using the same technique.

To Allocate an Array

Another way to create an array in HP VEE is to **allocate** it. Use one of the **`Allocate Array`** features when you want to create an array and set its values programmatically, and/or when you want to create an array of two or more dimensions.

To allocate an array, use the appropriate object, found under **`Data`** ⟹ **`Allocate Array`**. These objects include **`Alloc Text`** for text arrays and **`Alloc Integer`**, **`Alloc Real`**, **`Alloc Coord`**, **`Alloc Complex`**, and **`Alloc Pcomplex`** for numeric arrays. You can allocate arrays of from 1 to 10 dimensions using these objects. In general, an allocated array is "filled" with a repeated initial value. For example, the following is a five-element, one-dimensional array with an initial value of **`1`**:

```
0:  1
1:  1
2:  1
3:  1
4:  1
```

The following is a 3-by-3 matrix (two-dimensional array) with an initial value of 0 repeated throughout:

```
   0:   1:   2:
0: 0    0    0
1: 0    0    0
2: 0    0    0
```

Key Points

All HP VEE array indexes are zero based. That is, the first element has index 0 in any dimension.

All HP VEE arrays are passed as single data containers. That is, a three-by-three two-dimensional array has nine values in it, but the entire array is passed from object to object as a single data container.

For one-dimensional integer and real arrays only, you can initialize the array with a linear or logarithmic ramp of values. For example, the following is a five-element, one-dimensional, linear-ramp array:

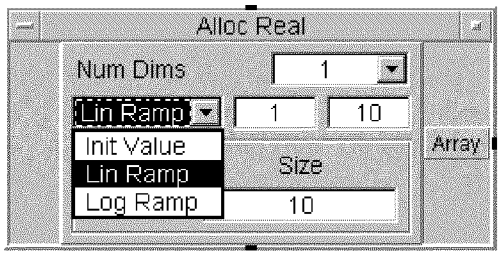
```
0:  1
1:  2
2:  3
3:  4
4:  5
```

Now let's allocate some arrays.

To Allocate a One-Dimensional Array:

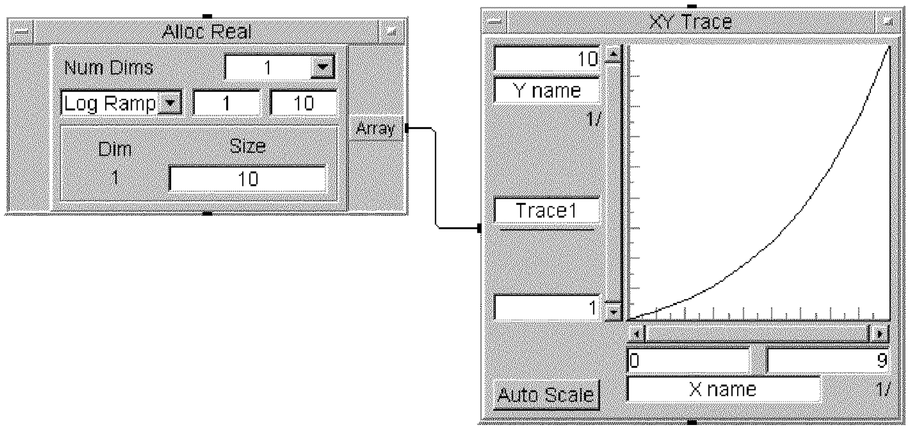
As an example, let's allocate a one-dimensional real array:

1. Add an `Alloc Real` object to the work area (`Data` ⟹ `Allocate Array` ⟹ `Real`).
2. By default the `Alloc Real` object allocates a one-dimensional array with a linear ramp of values. Click on the `Lin Ramp` field (or its arrow) to show the drop-down menu:



3. You can select `Init Value`, `Lin Ramp`, or `Log Ramp`. Click on `Log Ramp` to select a logarithmic ramp of values.

4. Connect an `XY Trace` display object to the `Alloc Real` object and run the program. Click on the `Auto Scale` button to see the data trace:

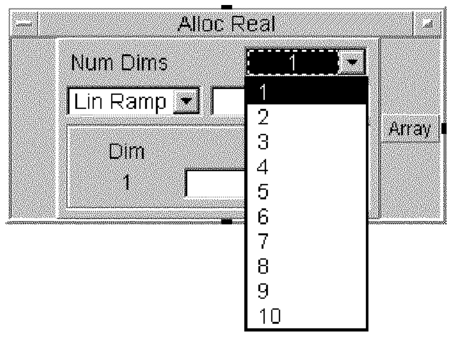


As you can see, the one-dimensional array consists of a logarithmic ramp of 10 values. The `Alloc Real` object outputs this array as a single data container. To verify this, probe the line between the two objects with `Line Probe` (Shift+left mouse button with the pointer near the line).

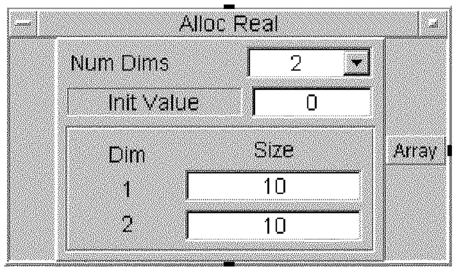
To Allocate a Two-Dimensional Array:

Let's allocate another real array, this one with two dimensions:

1. Add an `Alloc Real` object to the work area (`Data` ⟹ `Allocate Array` ⟹ `Real`).
2. By default the `Alloc Real` object allocates a one-dimensional array with a linear ramp of values. Click on the `Num Dims` field (or its arrow) to show the drop-down menu:



3. Click on `2` to select a two-dimensional array. (You can select up to `10` dimensions.) The object appears as shown below:



Three changes have occurred:

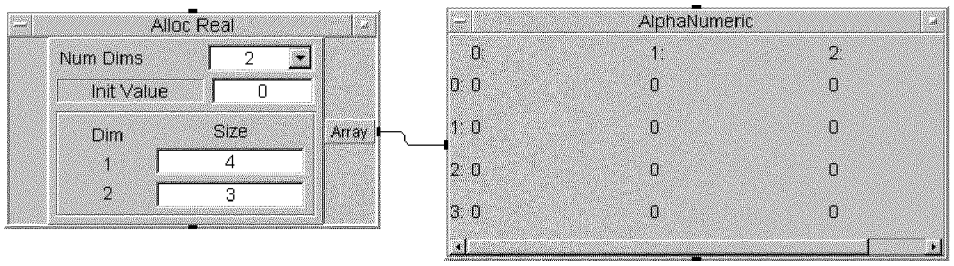
a. The number of dimensions is now `2`.

b. The `Lin Ramp` field has changed to `Init Value`, the only choice for an array of two or more dimensions. The default initial value is `0`.

c. A second dimension field appears. Both dimensions have a default size of `10`.

4. Change the size for the first dimension to `4`, and for the second to `3`. Then connect an `AlphaNumeric` display object to the output of the `Alloc Real` object.

5. Run the program, and then resize the **`AlphaNumeric`** object to show the entire array:



Num Dims 2
Init Value 0

Dim	Size
1	4
2	3

```
   0:   1:   2:
0: 0    0    0
1: 0    0    0
2: 0    0    0
3: 0    0    0
```

Of course, an array allocated with `0` values is of little use until you access those values and change them. Refer to "To Change Values in an Array", later in this chapter, for further information.

To Allocate an Array of Three or More Dimensions:

You can allocate an array of up to ten dimensions. The procedure is the same as for two dimensions. Just specify the number of dimensions and specify a size for each dimension. However, note the following points:

1. HP VEE display objects cannot display more than two dimensions.
2. The memory required to allocate an array is determined by the number of dimensions, the sizes of the dimensions, and the data type. If your computer has the recommended amount of memory (and swap space for UNIX systems), you should be able to allocate arrays of up to five dimensions, each with a default size of `10`. To allocate arrays of six or more dimensions you may need to declare smaller size parameters. If you try to allocate too large an array, the HP VEE process may run out of memory, resulting in an error.

To Create an Array Directly:

There are three ways that you can create an array directly, without allocating it:

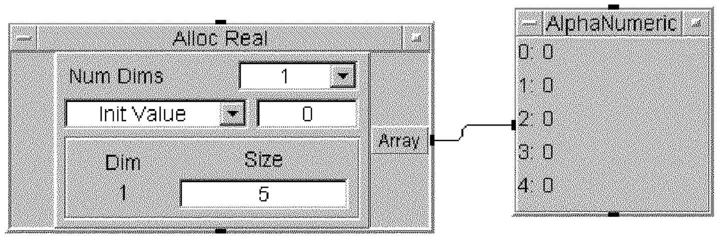
1. Create a one-dimensional array directly by creating a data constant. Refer to "To Create an Array Constant", earlier in this chapter, for further information.
2. Collect data into an array using the `Collector` or `Sliding Collector` object. Refer to "To Collect Data into an Array", later in this chapter, for further information.
3. Build the array in an expression. Refer to "To Build an Array in an Expression", later in this chapter, for further information.

To Change Values in an Array

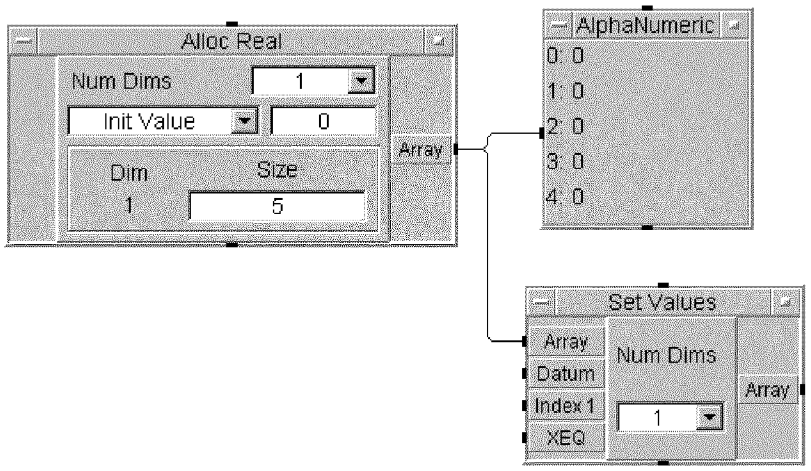
Once you have allocated an array, you can change values in the array with the `Set Values` object. This object refers to a value in an array by its index, and enters a new value.

To set a new value in a one-dimensional real array:

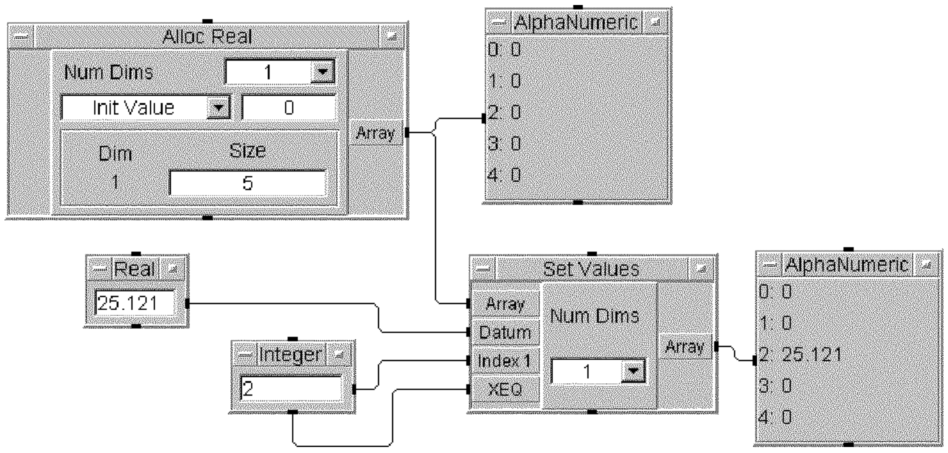
1. Allocate the array with `Alloc Real` (refer to "To Allocate an Array", earlier in this chapter). To save time, load the program `manual05.vee` from your manual examples directory. In this program, a five-element, one-dimensional real array is allocated with `0` as the initial value:



2. Add a `Set Values` object to the work area (`Data` ⟹ `Access Array` ⟹ `Set Values`). Double-click on the icon to show the open view `Set Values` object.
3. Connect the `Array` data output pin of `Alloc Real` to the `Array` data input terminal of `Set Values`:



4. Connect data constants to the `Datum` terminal (the value that you want to put into the array) and the `Index 1` terminal (the index where you want to put the value). For example, in the program below the real value `25.121` is input on the `Datum` terminal and the integer `2` is input on the `Index 1` terminal. Also, connect the the `XEQ` terminal, as shown.

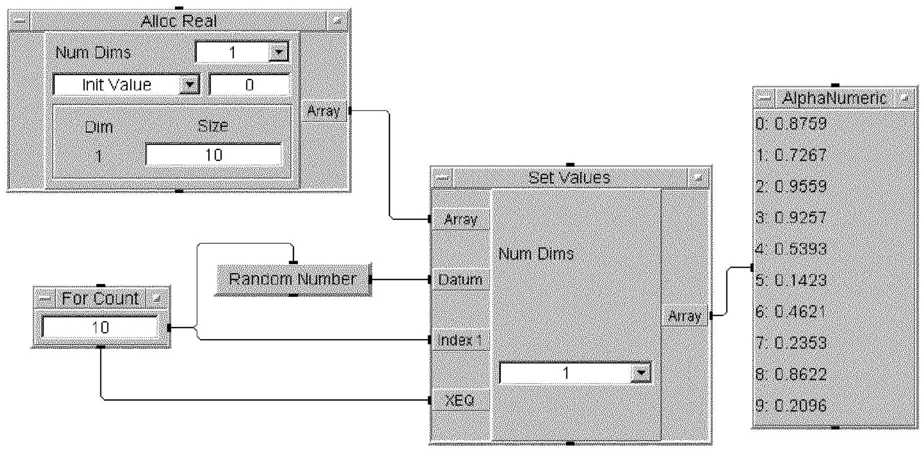


In this program, the **`Set Values`** object receives the allocated array as an input, along with the value `25.121` to set at index `2`. When the **`XEQ`** terminal is activated, **`Set Values`** outputs the new array with `25.121` at index `2` (the third value in the array).

Examples:

Let's look at some examples where all of the values in an array are changed.

The following program is saved as `manual06.vee` in your manual examples directory.

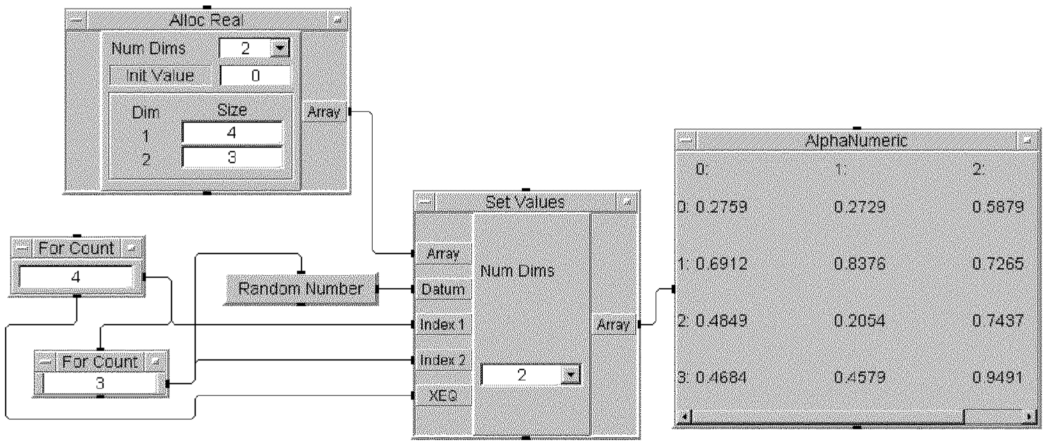


```
0: 0.8759
1: 0.7267
2: 0.9559
3: 0.9257
4: 0.5393
5: 0.1423
6: 0.4621
7: 0.2353
8: 0.8622
9: 0.2096
```

In this program:

1. The **`Alloc Real`** object allocates a one-dimensional, ten-element array with an initial value of `0`.
2. The array of zeros is input on the **`Array`** input terminal of the **`Set Values`** object, but **`Set Values`** outputs nothing until its **`XEQ`** terminal is activated.
3. The **`For Count`** object repeats 10 times. On each repeat, two things happen:
   a. The **`Random Number`** object outputs a new random number to the **`Datum`** input on the **`Set Values`** object.
   b. The count is output to the **`Index 1`** input on the **`Set Values`** object. Thus, each random number is indexed consecutively into the array.
4. When the **`For Count`** object finishes operating (after 10 repeats), its sequence output pin activates the **`XEQ`** terminal on the **`Set Values`** object, which then outputs the new array of 10 random numbers.

The following program (`manual07.vee` in your manual examples directory) performs the same basic actions as the previous example. However, this program uses two `For Count` objects to index the `Index 1` and `Index 2` inputs and create a two-dimensional array of random numbers:



Alloc Real: Num Dims 2, Init Value 0

Dim	Size
1	4
2	3

For Count: 4

For Count: 3

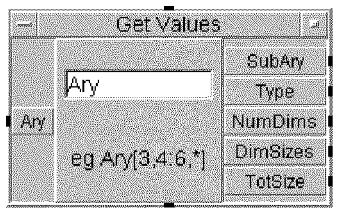
Set Values: Num Dims 2

```
   0:         1:         2:
0: 0.2759     0.2729     0.5879
1: 0.6912     0.8376     0.7265
2: 0.4849     0.2054     0.7437
3: 0.4684     0.4579     0.9491
```

To Extract Values from an Array

You can use the **Get Values** object to extract values from an array:

1. Add a **Get Values** object to the work area (**Data** ⟹ **Access Array** ⟹ **Get Values**). Double-click on the icon to show the open view object:



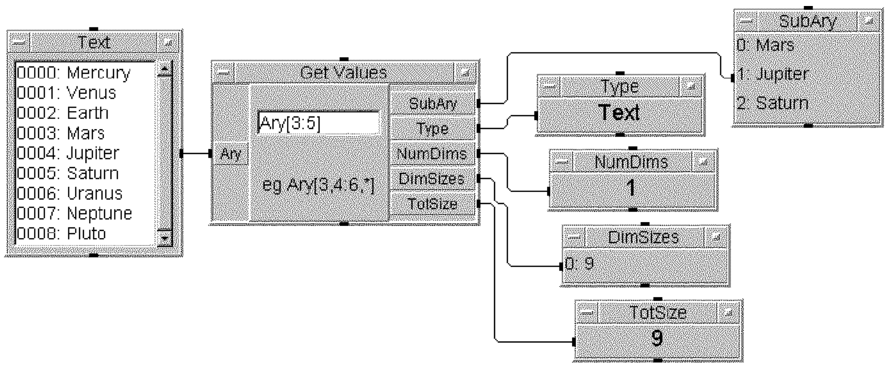
2. Connect any data source that outputs an array to the **Ary** data input terminal on **Get Values**. When **Get Values** operates, the **SubAry** terminal outputs a sub-array that depends on the expression in the **Get Values** expression field. The default expression **Ary** causes the entire original array to be output.

   The rest of the data output terminals give information about the original array on the **Ary** terminal:

   - **Type** outputs the data type of the array.
   - **NumDims** outputs the number of dimensions in the array.
   - **DimSizes** outputs a one-dimensional array listing the size (the number of elements) of each dimension.
   - **TotSize** outputs the total size of the array (the total number of elements).

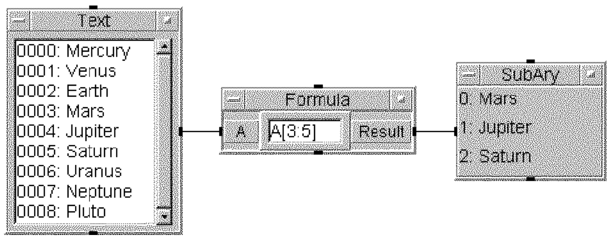
Examples:

In the following example (`manual08.vee` in your manual examples directory) a text constant one-dimensional array is connected to the `Ary` input terminal on the `Get Values` object.



The expression `Ary[3:5]` causes the sub-array consisting of the elements at index `3` through index `5` to be output on the `SubAry` terminal. In addition, the type, number of dimensions, size of the one dimension, and the total size of the original array are output. Note that for a one-dimensional array, the dimension size and total size are the same.

If you only want to obtain a sub-array, without the type and size information, you can include an expression in a `Formula` object instead. For example, the following program uses the expression `A[3:5]` to obtain the same sub-array as in the previous example:



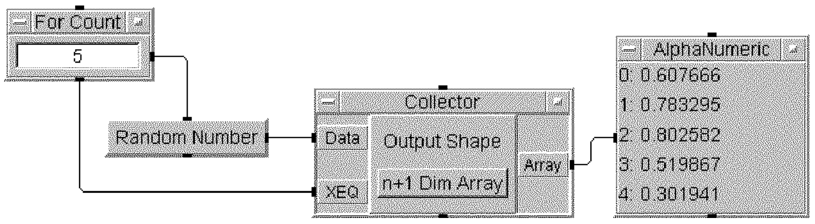
Refer to "To Use Arrays or Array Elements in Expressions", later in this chapter, for further information about using expressions to specify sub-arrays.

To Collect Data into an Array

Very often you may want to collect data into an array, and you may not know how big an array to allocate. In this case, use the **`Collector`** object to collect the data into an array.

For example, to collect five random values into an array:

1. Add a **`Collector`** object to the work area (**`Data`** $\Longrightarrow$ **`Collector`**). Double-click on the icon to show the open view object.
2. Add the **`For Count`**, **`Random Number`**, and **`AlphaNumeric`** objects to the work area and connect the program as shown below:



```
0: 0.607666
1: 0.783295
2: 0.802582
3: 0.519867
4: 0.301941
```

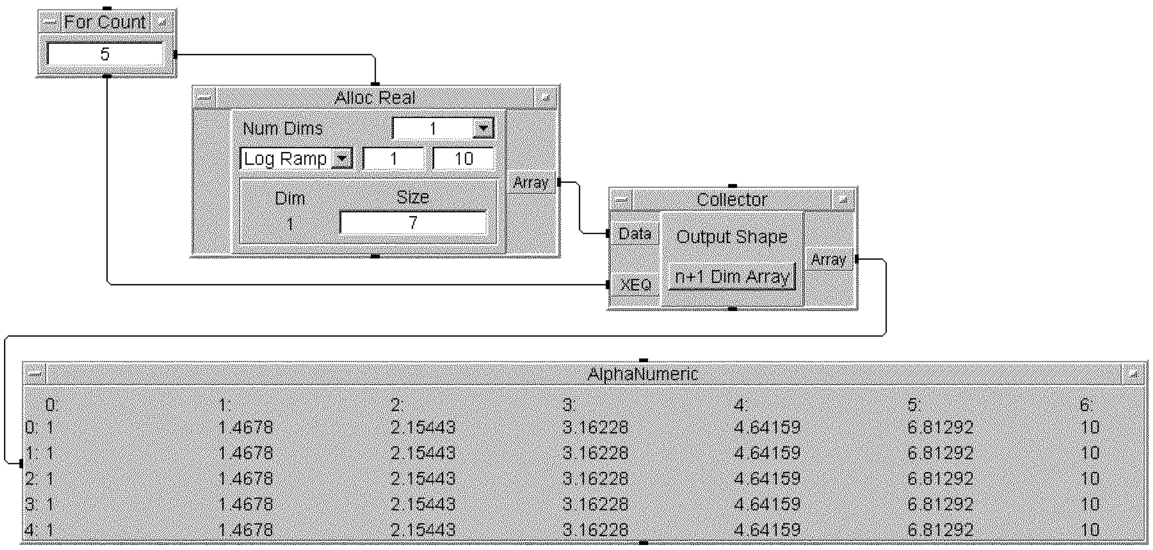
3. Run the program. (Resize the **`AlphaNumeric`** display to show the entire array.)

The **`Collector`** collects data until its **`XEQ`** terminal is activated, and then outputs the collected data as an array. In this case, the **`For Count`** object repeats five times, causing five random numbers to be generated, and then activates its sequence output pin. This activates the **`XEQ`** terminal on the **`Collector`** causing the array to be output on the **`Array`** terminal.

The **`Collector`** is set by default to output an array with **`n+1`** dimensions. (The other choice is **`1 Dim Array`**.) For scalar data, the output array is one-dimensional, as shown above.

Now let's modify our example to collect one-dimensional arrays of data into a two-dimensional array:

1. Remove the **Random Number** object from the previous program, and replace it with an **Alloc Real** object.
2. Connect the program as shown below. Configure the **Alloc Real** object to allocate a **Log Ramp** one-dimensional array with a size of **7**.



```
For Count: 5
Alloc Real: Num Dims 1; Log Ramp 1 10; Dim 1 Size 7
Collector: Output Shape n+1 Dim Array
```

```
     0:      1:       2:        3:        4:        5:        6:
0: 1       1.4678   2.15443   3.16228   4.64159   6.81292   10
1: 1       1.4678   2.15443   3.16228   4.64159   6.81292   10
2: 1       1.4678   2.15443   3.16228   4.64159   6.81292   10
3: 1       1.4678   2.15443   3.16228   4.64159   6.81292   10
4: 1       1.4678   2.15443   3.16228   4.64159   6.81292   10
```

3. Run the program. Resize the **AlphaNumeric** display to show the entire array.

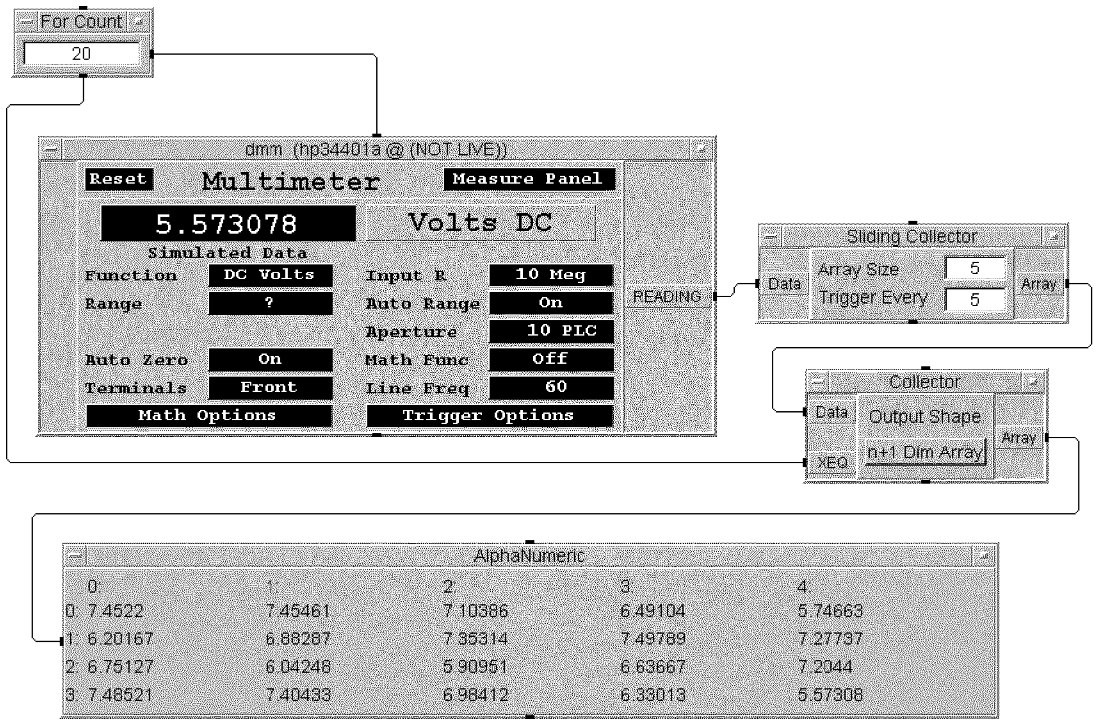
In this case, the **Collector** object is set to output an array of **n+1** dimensions. Since **n** (the number of dimensions in the input arrays) is **1**, the output array has **n+1**, or **2** dimensions. If you change the **Output Shape** to **1 Dim Array**, the data will be collected into a one-dimensional array with 35 elements.

In addition to the **Collector** object, you can use the **Sliding Collector** object to collect data into an array. The **Sliding Collector** differs from the **Collector** in two significant ways:

1. The output is always a one-dimensional array.

2. There is no **XEQ** terminal. Instead, the **Sliding Collector** is set to trigger after a specified number of values.

The following program (**manual09.vee** in your manual examples directory) uses both a **Sliding Collector** and a **Collector** to build an array of simulated readings from an instrument panel object for the HP 34401A Multimeter:



```
For Count: 20

5.573078   Volts DC
Simulated Data
Function     DC Volts     Input R      10 Meg
Range        ?            Auto Range   On
                          Aperture     10 PLC
Auto Zero    On           Math Func    Off
Terminals    Front        Line Freq    60

Sliding Collector
Array Size     5
Trigger Every  5

Collector
Output Shape   n+1 Dim Array
```

```
   0:        1:         2:         3:         4:
0: 7.4522    7.45461    7.10386    6.49104    5.74663
1: 6.20167   6.88287    7.35314    7.49789    7.27737
2: 6.75127   6.04248    5.90951    6.63667    7.2044
3: 7.48521   7.40433    6.98412    6.33013    5.57308
```

In most cases, set **Array Size** and **Trigger Every** to the same size, as shown above, for the **Sliding Collector** object.

The program works as follows:

- The **For Count** object triggers the **dmm (hp34401a)** instrument panel object 20 times.

- The instrument panel object generates a simulated reading (because live mode is `OFF`) on each iteration of the loop.
- The **`Sliding Collector`** triggers after every fifth reading, outputting a one-dimensional array of five readings.
- The **`Collector`** collects four consecutive one-dimensional arrays from the **`Sliding Collector`**, and outputs all 20 readings as a two-dimensional, four-by-five array.

For further information about using instrument panel objects, refer to Chapter 5.

To Use Arrays or Array Elements in Expressions

We covered how to use **`Get Values`** to extract a sub-array from an array in "To Extract Values from an Array", earlier in this chapter. We mentioned there that you can use an expression in either **`Get Values`** or in a **`Formula`** object to specify a sub-array. In fact, you can specify an array or sub-array in an expression in any HP VEE object that has an expression field.

In an expression, any reference to an array uses the following syntax:

- Brackets `[ ]` are used after the array name to enclose the parameters specifying a sub-array. For example, if `B` is a two-dimensional array, `B[0,0]` specifies the first element in it.
- The comma (`,`) separates array dimensions. Each sub-array operation *must* have exactly one specification for each array dimension.
- The colon (`:`) separates a range of elements within one of the array dimensions.
- The asterisk (`*`) is a wild-card character to specify *all* elements from a particular array dimension.

Note that all HP VEE array indexes are zero based. That is, the first element has index `0` in any dimension.

Examples:

Assume `A` is a one-dimensional array, 10 elements long:

```
0:  100m
1:  0.2
2:  0.3
3:  0.4
4:  0.5
5:  0.6
6:  0.7
7:  0.8
8:  0.9
9:  1
```

The following expressions extract portions of array `A`:

- `A` or `A[*]` returns the entire array `A`.
- `A[1]` returns a scalar, which is the the second element (index `1`) in `A`:

  ```
  0.2
  ```

- `A[1:1]` returns a one-dimensional array with only one element, which is the second element (index `1`) in `A`:

  ```
  0:  0.2
  ```

- `A[0:5]` returns a one-dimensional array that contains the first six elements (indexes `0` through `5`) of `A`:

  ```
  0:  100m
  1:  0.2
  2:  0.3
  3:  0.4
  4:  0.5
  5:  0.6
  ```

- `A[2:*]` returns a one-dimensional array that contains the third through the tenth elements (indexes `2` through `9`) of `A`. (The indexes are redefined as `0` through `7` in the subarray.)

  ```
  0:  0.3
  1:  0.4
  2:  0.5
  3:  0.6
  4:  0.7
  5:  0.8
  6:  0.9
  7:  1
  ```

Assume `B` is a 5-by-5 matrix (a two-dimensional array):

```
     0:     1:     2:     3:     4:
 0:  0      0.2    0.4    0.6    0.8
 1:  1      1.2    1.4    1.6    1.8
 2:  2      2.2    2.4    2.6    2.8
 3:  3      3.2    3.4    3.6    3.8
 4:  4      4.2    4.4    4.6    4.8
```

The following expressions extract portions of array `B`:

- `B` or `B[*,*]` returns the entire array `B`.
- `B[*]` returns an error because it specifies only one parameter, and `B` is a two-dimensional array.
- `B[1,2]` returns a scalar value from the second row (index `1`), third element (index `2`):

  ```
  1.4
  ```

- `B[1,*]` returns all of the second row (index `1`) as a one-dimensional array:

  ```
  0:  1
  1:  1.2
  2:  1.4
  3:  1.6
  4:  1.8
  ```

- `B[1,1:*]` returns all of the second row (index `1`), except for the first element (index `0`), as a one-dimensional array:

  ```
  0:  1.2
  1:  1.4
  2:  1.6
  3:  1.8
  ```

To Build an Array in an Expression

You can also build an array by specifying portions of other arrays in an expression. However, within an expression, all sub-arrays specified must have the same number of dimensions and contain the same number of values in each dimension.

Assuming `B` is the 5-by-5 matrix of the previous examples (shown at the top of the previous page), the expression `[ B[1,1] B[2,2] B[0,0] B[2,3] ]` returns the following one-dimensional array:

```
0:  1.2
1:  2.4
2:  0
3:  2.6
```

You can also build an array constant in an expression. For example, the expression `[1.1 1.3 1.2]` creates a three-element, one-dimensional real array:

```
0:  1.1
1:  1.3
2:  1.2
```

NOTE

Negative values in array constant expressions may cause problems. For example, `[5 4 -3 2]` is evaluated as [5 1 2]. So enter `[5 4 (-3) 2]` instead.

Working with Records

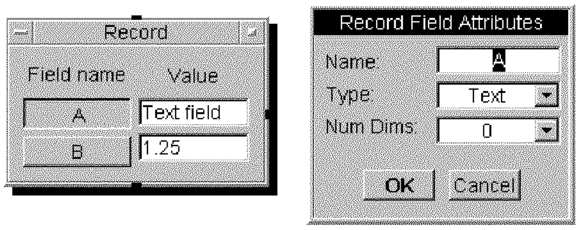
Arrays, described in the previous section, are extremely useful for handling large amounts of data. However, arrays have a limitation. You cannot mix data types within an array. For example, you cannot mix text data and numeric data in the same array. However, you can use the **record** data type for this purpose. You can create a record that contains several different data types and data shapes, each in a different record **field**.

This section covers some of the basics of using records in HP VEE. For further information about using records, and about storing records in DataSets, refer to the "Using Records and DataSets" chapter in *HP VEE Advanced Programming Techniques*.

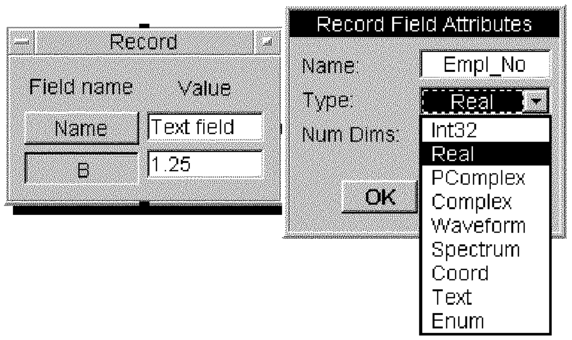
To Create a Record Constant

The best way to see how to create a record constant is to try an example. Let's create a record constant containing two record fields. The first field, a text field, contains a person's name. The second field, an integer, contains the person's employee number.

1. Add a `Record` constant object to the work area (`Data` $\Longrightarrow$ `Constant` $\Longrightarrow$ `Record`).
2. Note that the `Record` constant is unlike any of the other constant objects. It has two named fields (`A` and `B` by default), each of which has a type-in field. Click on the button labeled `A`. The `Record Field Attributes` dialog box appears:

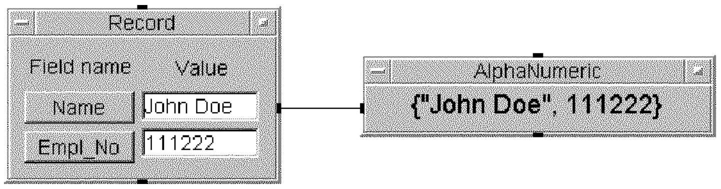


3. You can change the name of the field by typing in a new name in place of `A`. Change the field name to `Name`, but leave the data type set to `Text` and the number of dimensions set to `0` (scalar). Click on `OK`.
4. Now click on the button labeled `B`. Change the field name to `Empl_No`, and then click on `Real` (or the arrow) to show the drop-down menu:



5. Click on `Int32` to select the integer data type, but leave the number of dimensions set to `0`. Click on `OK`.

6. Enter **John Doe** in the the type-in field next to **Name**, and enter **111222** in the type-in field next to **Empl_No**, as shown below. Connect an **AlphaNumeric** display object to the output pin, and run the program:



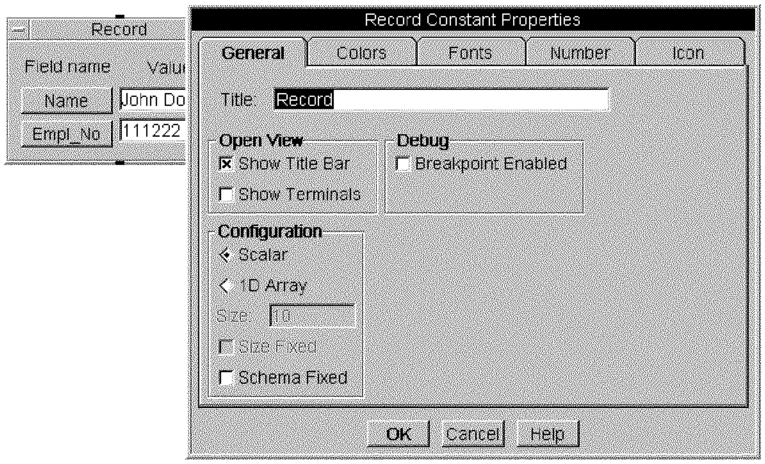
Name: John Doe
Empl_No: 111222

{"John Doe", 111222}

The record is displayed in the format `{"John Doe", 111222}`. This is a single record consisting of two record fields. In this case each field is a scalar. The first field is text, and the second field is an integer.

So far, our record constant can only store one person's name and employee number. Let's create an *array of records* to contain this information for a list of three people:

1. Beginning where we left off above, select **Edit Properties** from the object menu of the **Record** constant:



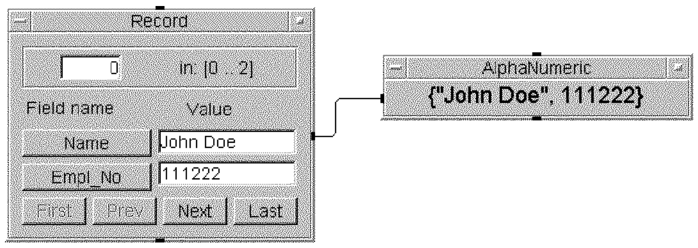
Name: John Do
Empl_No: 111222

Title: Record

[✓ Show Title Bar]
Show Terminals
Breakpoint Enabled

[✓ Scalar]
1D Array
Size: 10
Size Fixed
Schema Fixed

2. Now click on the radio button in front of **`1D Array`** to convert the **`Record`** constant into an array of records. Set the **`Size`** to **`3`**, and then click on **`OK`**. The object appears as follows:

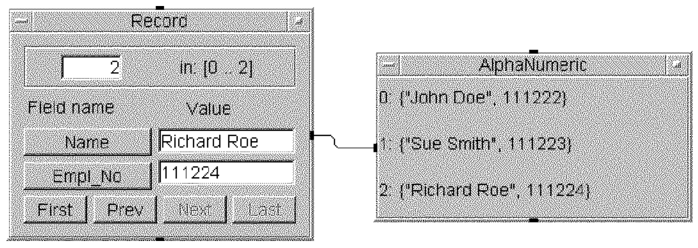


3. The **`First`**, **`Prev`**, **`Next`**, and **`Last`** buttons allow you to scroll through the array of records. Right now, each record is the same (**`{"John Doe", 111222}`**). Let's change this:

   a. Click on **`Next`** to scroll to the second record. Change the name to **`Sue Smith`** and the number to **`111223`**.

   b. Click on **`Next`** to scroll to the third (and last) record. Change the name to **`Richard Roe`** and the number to **`111224`**.

4. Now run the program. Re-size the **`AlphaNumeric`** display to show the entire array of records:

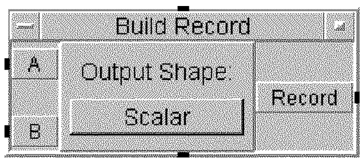


```
0: {"John Doe", 111222}
1: {"Sue Smith", 111223}
2: {"Richard Roe", 111224}
```

To Collect Data into a Record

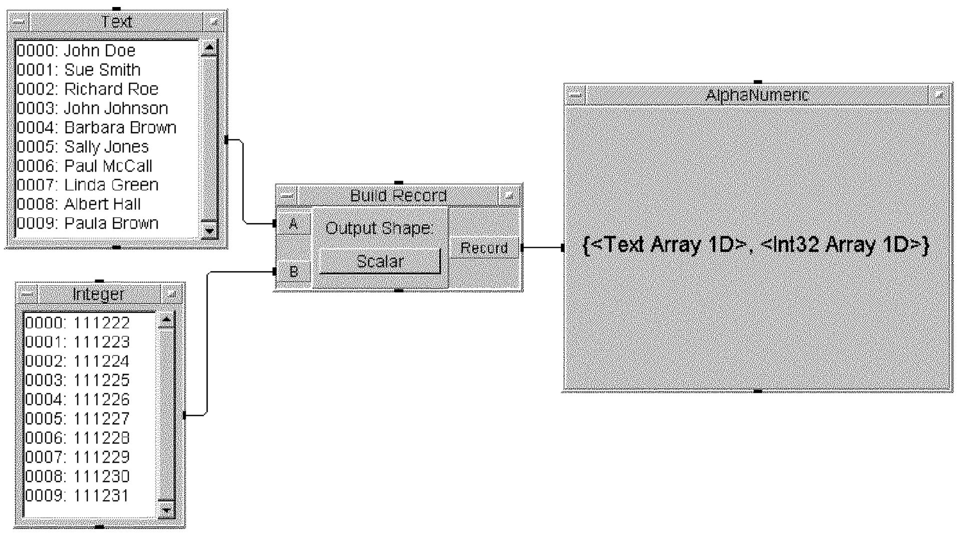
Entering individual fields into a `Record` constant could be rather tedious for large amounts of data. Fortunately, there is another way to create a record, by using the `Build Record` object.

To add a `Build Record` object to the work area, select `Data` ⟹ `Build Data` ⟹ `Record`.



Note that the `Build Record` object has two data input terminals, labeled `A` and `B` by default. The `Output Shape` is `Scalar` by default. Click on `Scalar` and the shape toggles to `Array 1D`. Most of the time you will want to leave the shape set to `Scalar`.

The `Build Record` object builds a record from the data on its data input terminals and outputs a record on its `Record` output terminal. To see how this works, let's look at an example. Load the program `manual10.vee` from your manual examples directory, and run the program:

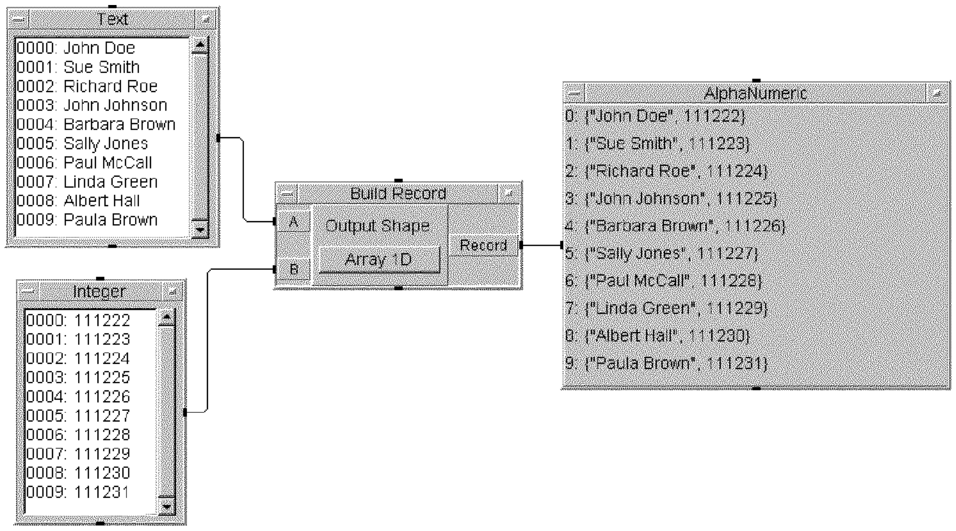


The program works as follows:

- The `Text` array on input `A` is a list of names.
- The `Integer` array on input `B` is a list of employee numbers.
- The `Build Record` object is set for an output shape of `Scalar`, so it outputs the *scalar record* `{<Text Array 1D>, <Int32 Array 1D>}` consisting of two record fields, each containing a one-dimensional array. The first record field is the array of names, and the second is the array of employee numbers.

Try changing the output shape:

1. Click on `Scalar` to display `Array 1D`.
2. Run the program. The following should be displayed.



```
0000: John Doe
0001: Sue Smith
0002: Richard Roe
0003: John Johnson
0004: Barbara Brown
0005: Sally Jones
0006: Paul McCall
0007: Linda Green
0008: Albert Hall
0009: Paula Brown
```

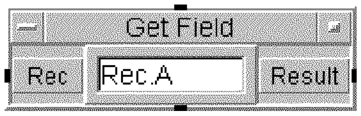
```
0000: 111222
0001: 111223
0002: 111224
0003: 111225
0004: 111226
0005: 111227
0006: 111228
0007: 111229
0008: 111230
0009: 111231
```

```
0: {"John Doe", 111222}
1: {"Sue Smith", 111223}
2: {"Richard Roe", 111224}
3: {"John Johnson", 111225}
4: {"Barbara Brown", 111226}
5: {"Sally Jones", 111227}
6: {"Paul McCall", 111228}
7: {"Linda Green", 111229}
8: {"Albert Hall", 111230}
9: {"Paula Brown", 111231}
```

Now the program builds an *array of records*. The **Build Record** object matches each name and employee number to form an individual record in the array of records to be output on the **Record** terminal. When the output shape is set to **Array 1D**, the input arrays must be of the same size. (However, this is not true if the output shape is **Scalar**.)

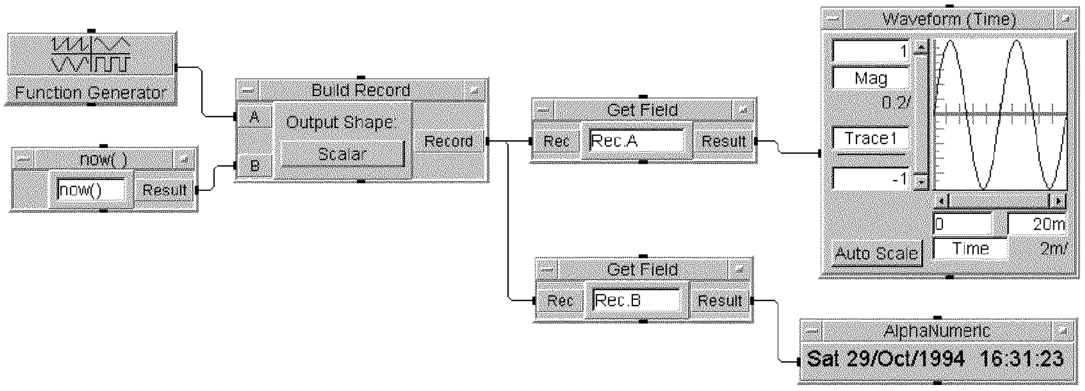
To Extract a Field from a Record

You can extract a field from a record by using the **Get Field** object (**Data** ⟹ **Access Record** ⟹ **Get Field**):



The **Get Field** object is really a **Formula** object pre-configured with a **Rec** input terminal and the expression **Rec.A**. This expression uses *record syntax* to extract a field from a record input on the **Rec** terminal. The expression **Rec.A** means "field **A** of record **Rec**". You can use this same syntax to refer to records from within a **Formula** object, or other object with an expression field. For further information on record syntax, refer to "To Use a Record in an Expression".

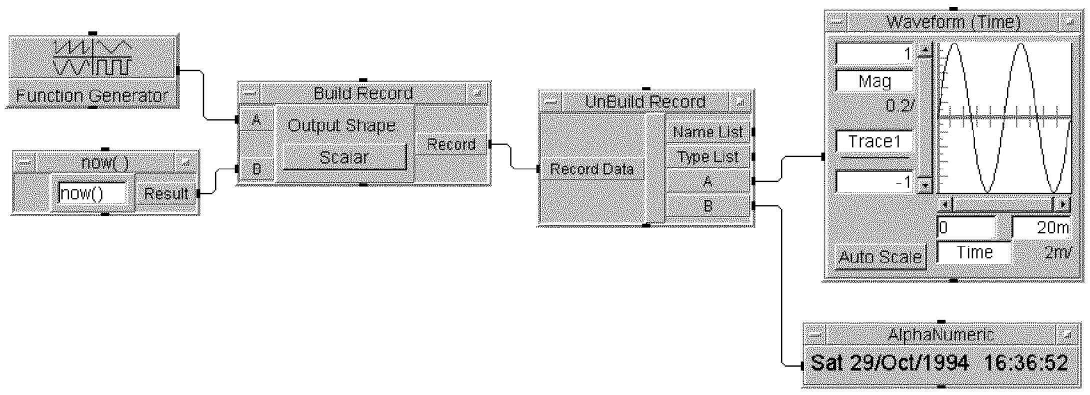
Let's look at an example. The following program (**manual11.vee** in your manual examples directory) builds a record consisting of two fields. The first field (**A**) is a waveform, and the second (**B**) is the time (a scalar).



The upper **Get Field** object uses the expression **Rec.A** to extract "field **A** of record **Rec**" (the waveform). The waveform is then displayed by the **Waveform** object.

The lower **Get Field** object uses the expression **Rec.B** to extract "field **B** of record **Rec**" (the time). The time is then displayed by the **AlphaNumeric** object (which has been configured for the time stamp format).

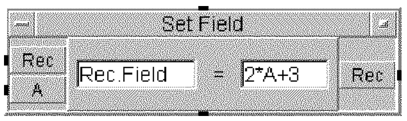
You can also use the **UnBuild Record** object (**Data** ⟹ **UnBuild Data** ⟹ **Record**) to extract fields from a record. In the following program (**manual12.vee** in your manual examples directory) both **Get Field** objects have been replaced with an **UnBuild Record** object, which extracts fields **A** and **B**.



You can use the **UnBuild Record** object to extract all of the fields in a record. Also, **UnBuild Record** returns a **Name List** (listing the names of all of the fields) and a **Type List** (listing the data types of all of the fields).

To Change a Field in a Record

To change a field in a record, use the **Set Field** object (**Data** ⟹ **Access Record** ⟹ **Set Field**):



The **Set Field** object has two expression fields:

- The *left-hand expression* specifies a field in the record **Rec** (the record connected to the **Rec** data input terminal). This expression uses the record syntax **Rec.Field** where **Field** can be any field name within the record. For example, **Rec.B** specifies "field **B** of record **Rec**".

- The *right-hand expression* contains an expression that specifies a new value for the field specified by the left-hand expression. The default example is `2*A+3`, where `A` is the variable corresponding to data input terminal `A`.

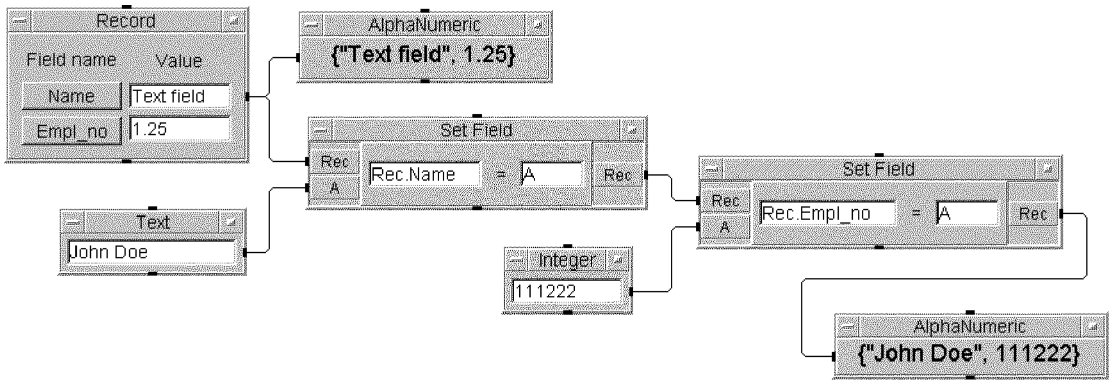
In other words, if the left-hand expression is `Rec.B` and the right-hand expression is `A`, the meaning is:

"Let field `B` of record `Rec` be equal to the value on data input terminal `A`."

NOTE

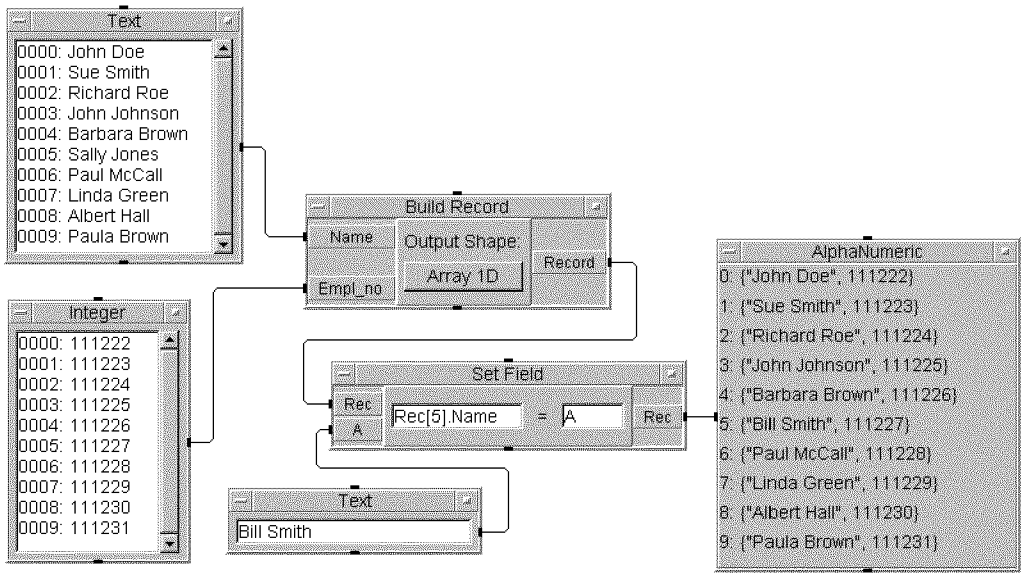
The syntax of the left-hand expression of the `Set Field` object is limited. Refer to the `Set Field` topic in the *HP VEE Reference* manual for further information.

Let's look at an example to see how `Set Field` works. The following program (`manual13.vee` in your manual examples directory) uses two `Set Field` objects to change fields `Name` and `Empl_no` in a record constant:



The original record constant has two fields: `Name` has the value `"Text Field"` and `Empl_no` has the value `1.25`. The first `Set Field` object changes the value of **Name** to **"John Doe"**, and the second changes the value of **Empl_no** to **111222**. It takes two **Set Field** objects connected in series to change both fields (**Name** and **Empl_no**) in the output record.

You can also use **Set Field** to update an array of records. For example, in the following program the expression **Rec[5].Name = A** specifies that array element **5** in the **Name** field of record **Rec** is to be set equal to the data on input terminal **A**. So **Bill Smith** replaces **Sally Jones** in the resulting array of records.



```
0000: John Doe
0001: Sue Smith
0002: Richard Roe
0003: John Johnson
0004: Barbara Brown
0005: Sally Jones
0006: Paul McCall
0007: Linda Green
0008: Albert Hall
0009: Paula Brown
```

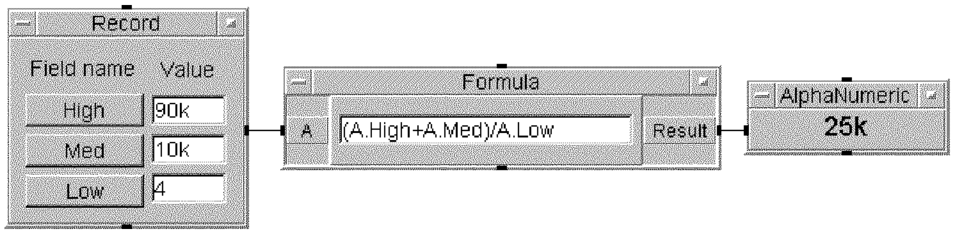
```
0000: 111222
0001: 111223
0002: 111224
0003: 111225
0004: 111226
0005: 111227
0006: 111228
0007: 111229
0008: 111230
0009: 111231
```

```
Bill Smith
```

```
0: {"John Doe", 111222}
1: {"Sue Smith", 111223}
2: {"Richard Roe", 111224}
3: {"John Johnson", 111225}
4: {"Barbara Brown", 111226}
5: {"Bill Smith", 111227}
6: {"Paul McCall", 111228}
7: {"Linda Green", 111229}
8: {"Albert Hall", 111230}
9: {"Paula Brown", 111231}
```

To Use a Record in an Expression

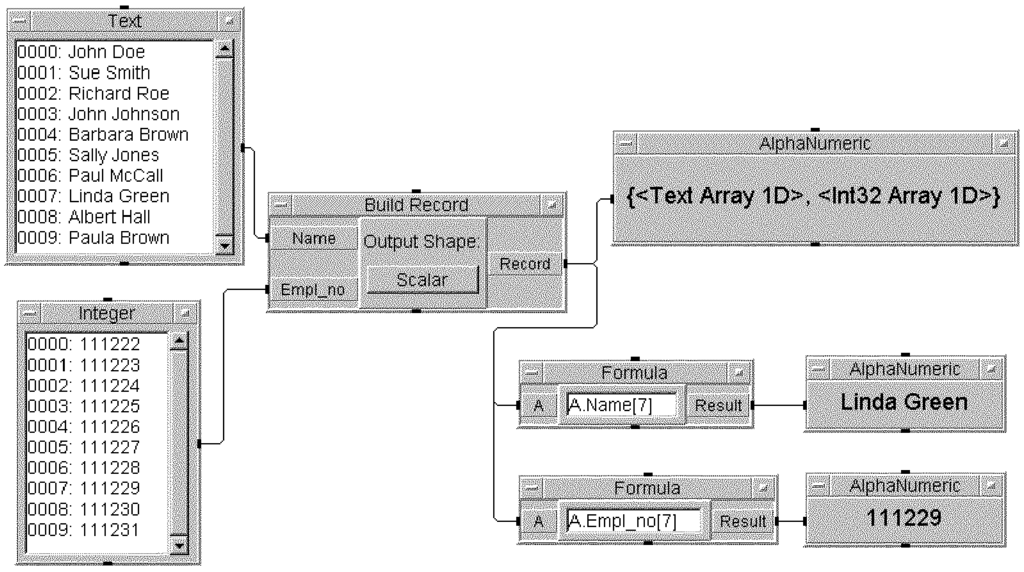
You can use the record syntax **Rec.Field**, where **Field** is the name of a field within record **Rec**, in any HP VEE expression field. For example, in the following program the expression **(A.High + A.Med)/A.Low** adds fields **High** and **Med** of record **A**, and divides the result by field **Low**:



NOTE

To avoid confusion, give your record fields descriptive names. For example, if in the above example the three fields were named **A**, **B**, and **C**, the expression would be **(A.A + A.B)/A.C**, which is hard to understand.

You can combine the record syntax with array syntax to access an array element from a record field. For example, the following program is saved as `manual14.vee` in your manual examples directory:



In this program, record field **`Name`** is a one-dimensional text array of names, and field **`Empl_no`** is a one-dimensional integer array of employee numbers. The expression **`A.Name[7]`** extracts array element `7` from the array in record field **`Name`**, which is **`Linda Green`**. The expression **`A.Empl_no[7]`** extracts array element `7` from the array in record field **`Empl_no`**, which is `111229`.

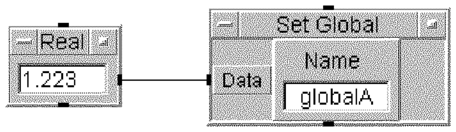
For further information about using record syntax in expressions, refer to "Mathematically Processing Data" in Chapter 3 of the *HP VEE Reference* manual.

Working with Global Variables

Thus far, all of the mathematical expressions in this chapter have used **local variables**. A local variable has a value that comes from the data input terminal with the same name as the variable. For example, if a `Formula` object includes the expression `2*A+3`, the variable `A` in the expression corresponds to the `A` data input terminal. Local variables require corresponding data input terminals so that the expression can be evaluated. **Global variables** are different. Once you create a global variable, you can use it in an expression anywhere in your HP VEE program — you don't need to include a data input terminal for it. Let's look at how to create and use global variables.

To Create a Global Variable

To create a global variable, use the `Set Global` object (`Data` ⟹ `Globals` ⟹ `Set Global`). In the following example, a `Real` constant object is connected to the `Data` terminal of a `Set Global` object.



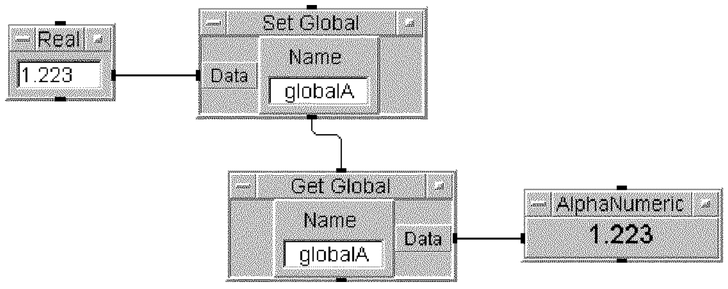
That's all there is to it. The `Set Global` object sets the value `1.223` (from the `Real` constant) as the value of the global variable named `globalA`.

NOTE

You can use any valid variable name in place of `globalA` for a global variable. Just enter the desired name in the `Set Global` object. However, if a global variable has a name that is also used for a local variable, the local variable takes precedence. To avoid naming conflicts, don't name a global variable **A**, **B**, or **C**, or other names used by local variables in your program. Global names are not case sensitive (`globalA` is equivalent to `GlobalA`).

To Use a Global Variable

Once you have created a global variable, you can access it with a **`Get Global`** object (**`Data`** $\Longrightarrow$ **`Globals`** $\Longrightarrow$ **`Get Global`**). The following program shows how to do this:

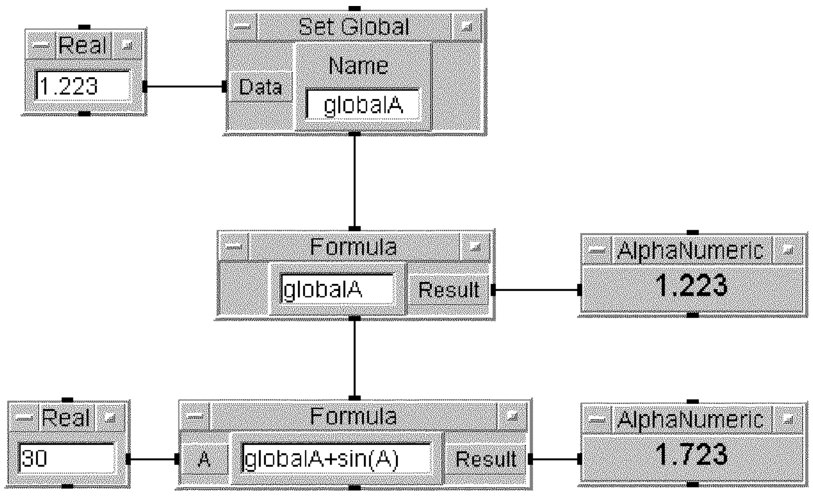


In this program, the **`Set Global`** object creates the global variable **`globalA`**, which has the value `1.223`. The **`Get Global`** object accesses **`globalA`** and outputs its value.

Here are two important points:

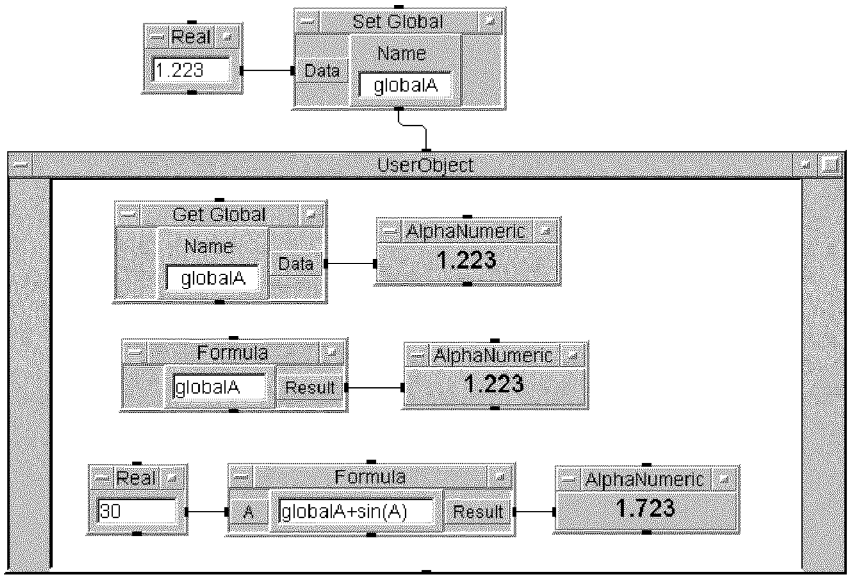
1. In order to avoid using obsolete data from a previous program run, the values of all global variables are deleted at PreRun. That is, any "old" values are cleared when you run the program. (You can turn off `Delete Globals at PreRun` with `Edit Default Preferences`. However, this cancels the protection against using obsolete data.)
2. Because the value of `globalA` is deleted at PreRun, it is necessary to connect the sequence output pin of the `Set Global` object to the sequence input pin of the `Get Global` object. Otherwise, an error occurs because `globalA` has no defined value.

You cannot include an expression in the name field of a `Get Global` object. However, you can use a global variable in an expression in a `Formula` object, or other object that includes an expression. For example, in the program below two `Formula` objects use `globalA` in expressions:



Note that in the above program the sequence input pins of both `Formula` objects must be connected as shown to ensure that `globalA` is set before it is accessed by an expression.

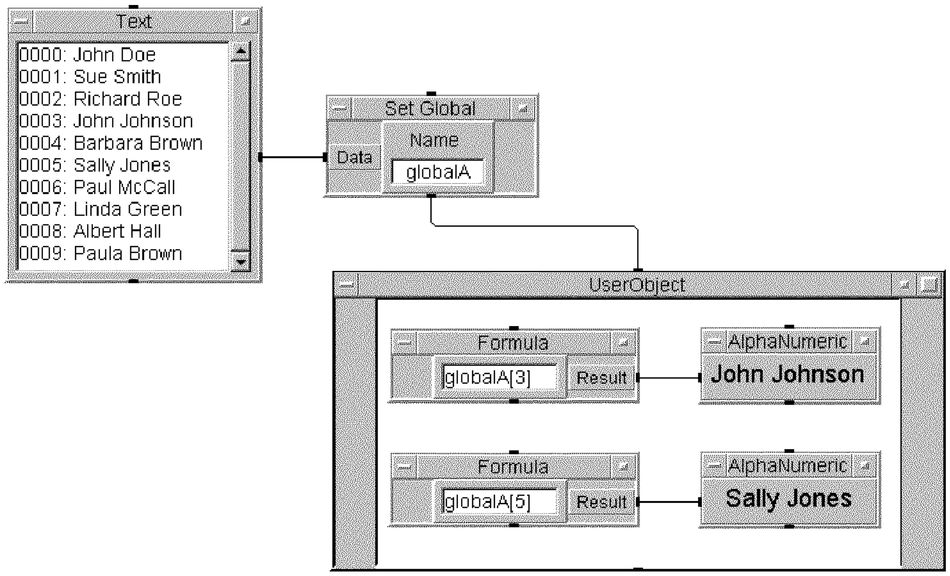
There is another way to ensure that a global variable is defined, without directly connecting the sequence pins of all of the objects accessing it. You can define a global variable in the main context of an HP VEE program and then access the global variable from within the context of a `UserObject`. The following program (`manual18.vee` in your manual examples directory) shows how this is done:



The sequence output pin of the `Set Global` object is connected to the sequence input pin of the `UserObject`. This ensures that the `Set Global` object operates before the `UserObject`, or any object within it, operates. Thus, the global variable `globalA` is defined before any object within the `UserObject` attempts to access it. This method simplifies the program and makes it easier to understand. For further information about `UserObjects` refer to Chapter 8.

To Use a Global Array

You can declare an array to be a global variable. This makes array handling easier because you can then access the array from anywhere in your HP VEE program. Here is a simple example (`manual19.vee` in your manual examples directory):



In the above program, the two **`Formula`** objects use array syntax to access elements of the global array. For example, the expression **`globalA[3]`** retrieves element **`3`** of the global array **`globalA`**.

You can do the same thing with a record, or any other type of data. Just use **`Set Global`** to declare the data to be a global variable, and then use an expression to access the global variable.

4

Creating and Using Data Files

Creating and Using Data Files

This chapter shows how to create and use data files from HP VEE.

Storing and Retrieving Data

This section shows how to use the `To File` object to create data files and to output data to such files, and how to use the `From File` object to retrieve data from data files.

NOTE

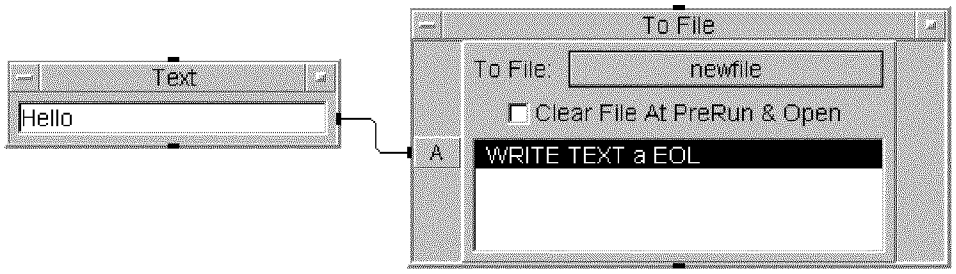
HP VEE also provides two special file I/O objects to create and access a special kind of data file, the **DataSet**, which is created from record data. For information about how to use the `To DataSet` and `From DataSet` objects, refer to the "Using Records and DataSets" chapter in *HP VEE Advanced Programming Techniques*.

To Create a Data File

You can write data to a new file name with the `To File` object:

1. Add a `To File` object to your program (`I/O` ⟹ `To` ⟹ `File`).
2. Connect the data input pin of the `To File` object to a source of data that you want to output to a file.
3. Change the name of the destination data file to the desired name. To do this, click on the `To File` button (by default, this button displays `myFile`) and enter a new file name in the dialog box. Then click on `OK`.

For example, in the program that follows the `To File` object outputs `Hello` to a file called `newfile`. If the file `newfile` does not already exist, HP VEE creates it.



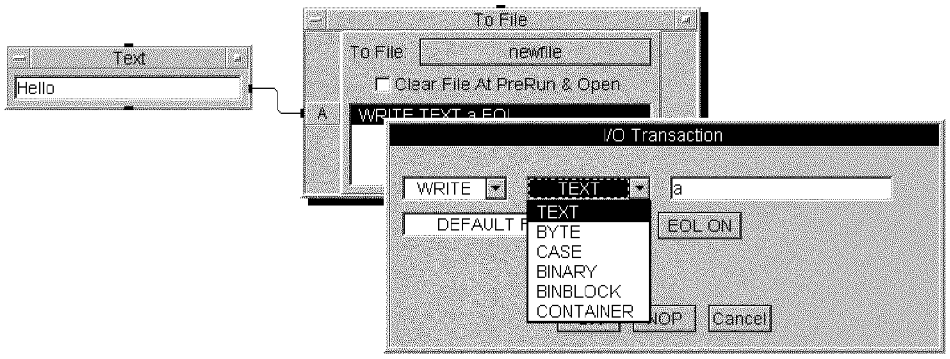
Note that the `To File` object uses a write **transaction** (by default, `WRITE TEXT a EOL`) to output the data to the file. The following section describes file I/O transactions and how to edit them.

To Edit a File I/O Transaction

The `To File` object uses a write **transaction** to communicate with the specified data file. The default transaction is `WRITE TEXT a EOL`, which writes the data on input `A` as text, followed by an end-of-line sequence (a carriage-return/line-feed by default). In the previous example, the text `Hello` is written to `newfile`, terminated with a carriage-return/line-feed. (The `From File` object uses similar transactions to read data from a file.)

You can edit a transaction by using the `I/O Transaction` dialog box:

1. Double-click on the transaction field in the `To File` object. The `I/O Transaction` dialog box appears:
2. Click on the `Text` field (or its arrow). A drop-down menu appears:



3. If you want to change the transaction to a **WRITE CONTAINER** transaction, click on **CONTAINER**, and then click on **OK**.

Now the program will write the data (**Hello**) to the file named **newfile** as a container (the HP VEE internal data format). If you examine the file with a text editor, it will appear as follows:

```
(Text
 (data  "Hello" )
)
```

In general, the container format is useful if you want to retrieve the data back into HP VEE using **From File**, but not if you want to access the data file with an external application such as a spreadsheet program. We'll look at this again later in this chapter.

You can add additional transactions to the **To File** and **From File** objects. You can also configure various options in such transactions, such as the end-of-line sequence. For a detailed discussion of transactions, refer to the "Using Transaction I/O" chapter in *HP VEE Advanced Programming Techniques*.

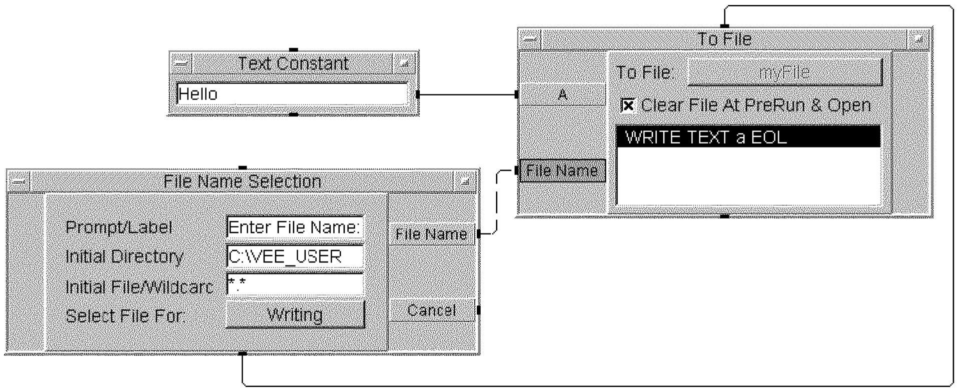
To Select a File Name Programmatically

As mentioned previously, you can select a new file name in the **To File** (or **From File**) object by clicking on the file name button, which by default displays **myFile**. Enter a new file name in the dialog box, and click on **OK** to accept it.

However, you can also set up your program to request a file name at run time by using the **File Name Selection** dialog object:

1. Add a **File Name** control input terminal to the **To File** object. (Object menu ⟹ **Add Terminal** ⟹ **Control Input**.)
2. Add a **File Name Selection** object (**Data** ⟹ **Dialog Box** ⟹ **File Name Selection**) to the work area and connect its **File Name** output to the **File Name** control pin on the **To File** object. Also, connect the sequence output pin of the **File Name Selection** object to the sequence input pin of the **To File** object. (This ensures that the **To File** object will not operate until a file name has been selected.)
3. Change the **Select File For** field to **Writing**. (This ensures that the user is alerted before an existing file is overwritten.)

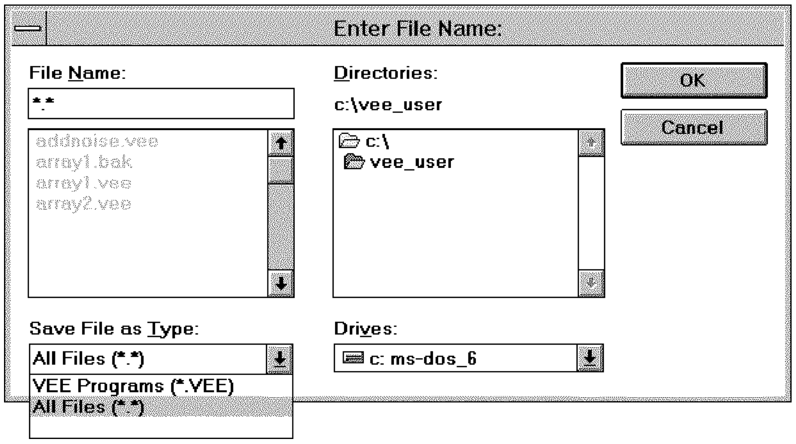
Here is an example program using the **File Name Selection** object as described above:



WRITE TEXT a EOL

Prompt/Label: Enter File Name:
Initial Directory: C:\VEE_USER
Initial File/Wildcard: *.*
Select File For: Writing

When you run the program, the **Enter File Name** dialog box appears, prompting for a file name. The following dialog box is displayed by HP VEE for Windows. Under **Save File as Type**, select **All Files (*.*)** since the file to be saved is not a program file:



Enter a new file name or click on an existing file name, and then click on `OK`. The data is saved in the named file.

To Write Data to a File

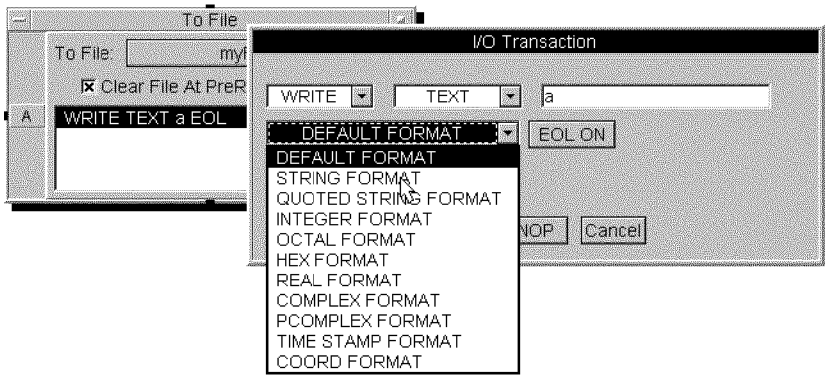
Let's take a closer look at using the `To File` object to write data to a file. To see the versatility of the `To File` object, follow along and create the simple program that follows:

1. Create a `To File` object (`I/O` ⟹ `To` ⟹ `File`). Click on the checkbox to turn on `Clear File At PreRun & Open`.

NOTE

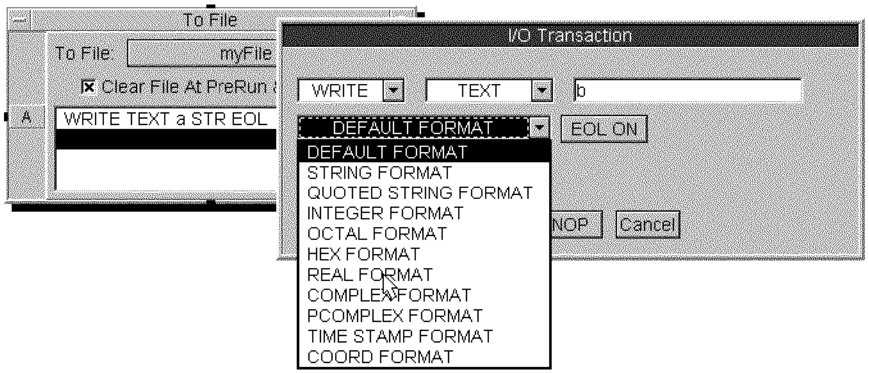
In general, you'll want to turn on **Clear File At PreRun and Open** by clicking on the checkbox. This ensures that the file is cleared and that no old data is in the file.

2. Double-click on the **WRITE TEXT** transaction to bring up the **I/O Transaction** dialog box.
3. Now click on the **Default Format** field (or its arrow). A drop-down menu appears:

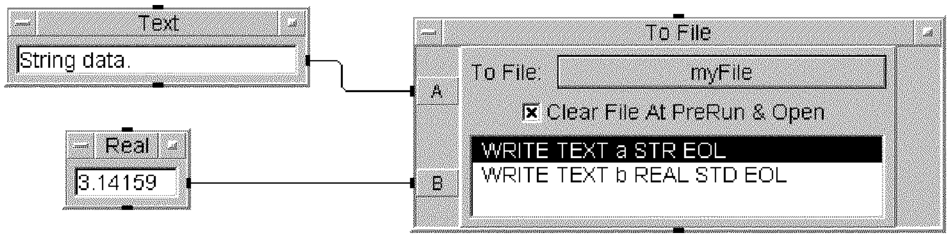


4. Click on **String Format**, and then click on **OK**. Now the modified transaction is **WRITE TEXT a STR EOL**. This transaction writes variable **a** (the data on input pin **A**) to **myFile** as text data in string format, followed by an end-of-line sequence (carriage-return/line-feed).
5. Now add a second transaction to the **To File** object (object menu ⟹ **Add Trans**). The **I/O Transaction** dialog box appears.
6. Change the variable field (**a** by default) to **b**.

7. Click on the `Default Format` field (or its arrow). The drop-down menu appears:



8. Click on `Real Format`, and then click on `OK`. The second transaction is `WRITE TEXT b REAL STD EOL`. This transaction writes variable `b` (you'll need to add terminal `B`) as text in `REAL STD` format to `myFile`.
9. Add terminal `B` to the `To File` object, and then connect a `Text` constant object to terminal `A` and a `Real` constant object to terminal `B`. The finished program should appear as follows:

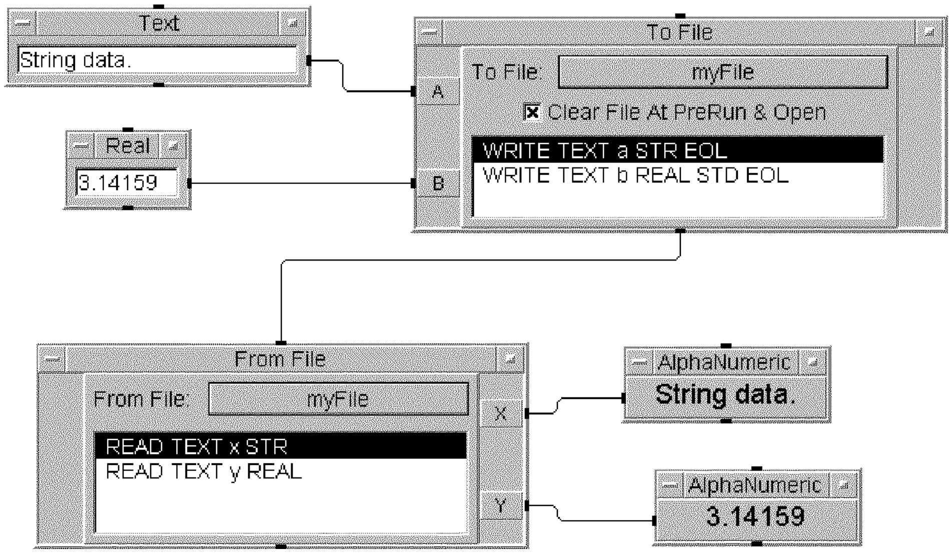


When you run the program, two lines of data are written to `myFile`, each followed by an end-of-line sequence (carriage-return/line-feed). The first is the string `String data.` The second is the real number `3.14159`. Now let's read the data back into HP VEE.

To Read Data from a File

Let's continue with the program of the previous section, which writes data to **myFile**. We'll add a **From File** object to the program and read the data back into HP VEE.

1. Add a **From File** object to the program (**I/O** ⟹ **From** ⟹ **File**).
2. Change the default transaction to **READ TEXT x STR** using the **I/O Transaction** dialog box as described in the previous section.
3. Add a second transaction **READ TEXT y REAL**, again using the **I/O Transaction** dialog box as described in the previous section.
4. Add a **Y** data output terminal to the **From File** object.
5. Connect **AlphaNumeric** displays to both outputs (**X** and **Y**).
6. Connect the sequence output pin of the **To File** object to the sequence input pin of the **From File** object. The finished program should appear as follows:



WRITE TEXT a STR EOL
WRITE TEXT b REAL STD EOL

READ TEXT x STR
READ TEXT y REAL

This program writes two types of data to the same file (`myFile`) and then reads the data back into HP VEE:

- The string data on input terminal `A` is written by `To File`, as variable `a`, to the file. The `From File` object reads variable `x` as string data and outputs the data on terminal `X`.
- The real number on input terminal `B` is written by `To File`, as variable `b`, to the file. The `From File` object reads variable `y` as a real number and outputs the data on terminal `Y`.

Note that the sequence pins must be connected in this program to ensure that data is written to `myFile` before the `From File` object attempts to read data from the file.

Using File I/O — Some Examples

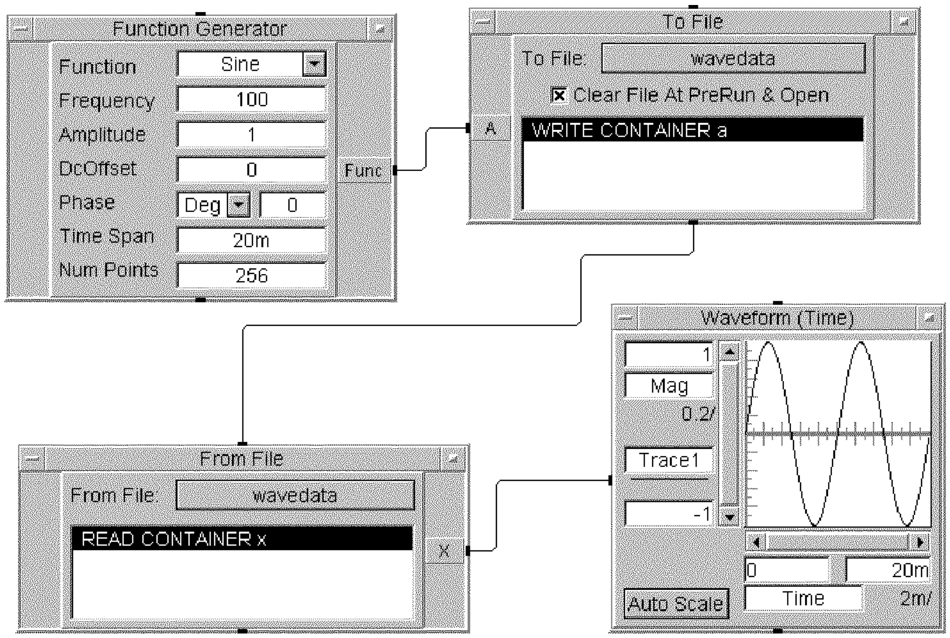
This section shows some practical applications of data file I/O.

To Store and Retrieve a Waveform

Waveform data is best stored and retrieved in the form of a data container. The basic procedure is as follows:

1. Use a **WRITE CONTAINER** transaction in the **To File** object to write the waveform data to a file.
2. Use a **READ CONTAINER** transaction in the **From File** object to read the waveform data back from the file.

The following example (**manual20.vee** in your manual examples directory) shows how this can be done:



Function: Sine
Frequency: 100
Amplitude: 1
DcOffset: 0
Phase: Deg 0
Time Span: 20m
Num Points: 256

To File: wavedata
☒ Clear File At PreRun & Open
WRITE CONTAINER a

From File: wavedata
READ CONTAINER x

1
-1
0
20m

The program works as follows:

1. The waveform data from the **Function Generator** is written to the **wavedata** file by the **To File** object using a **WRITE CONTAINER a** transaction.
2. The data is read back from the **wavedata** file as a waveform container by **From File** using a **READ CONTAINER x** transaction. The data is output on terminal **X**, and displayed by the **Waveform (Time)** display.
3. The sequence pin connections ensure that the waveform is written to the file before **From File** attempts to read it.

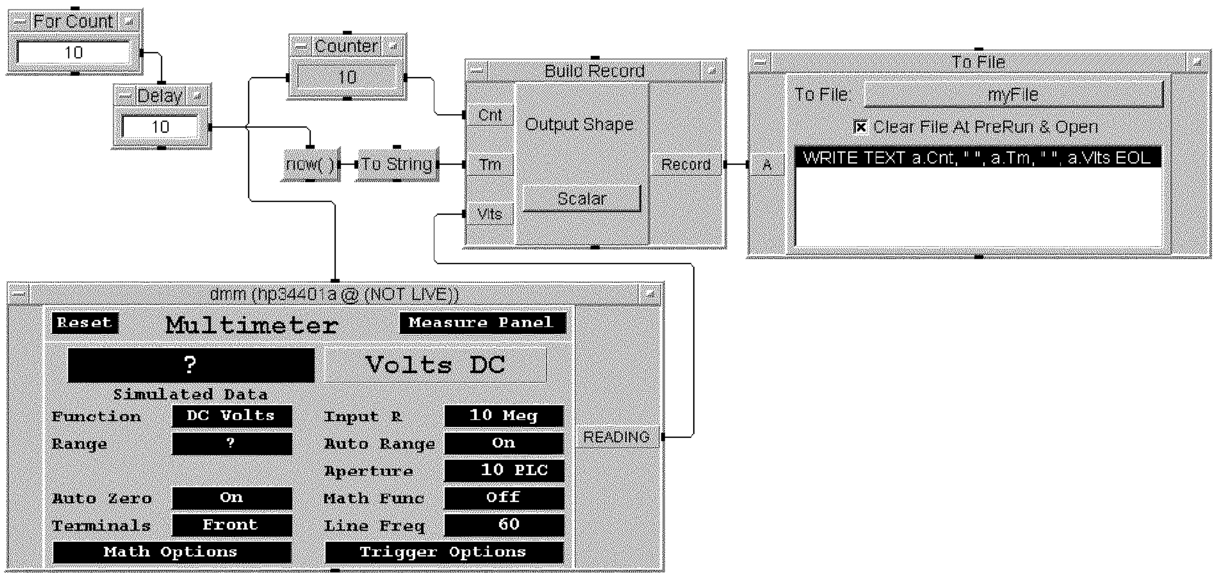
Of course, in a more typical situation, one HP VEE program would store the waveform data into a file, and then a second HP VEE program would read the data back into HP VEE, analyze it, and display it.

To Store Data for a Spreadsheet Program

You can use HP VEE file I/O to produce a data file that can be read by a spreadsheet program such as Microsoft Excel or Lotus 1-2-3. In general, the steps are as follows:

1. Determine what data file format the spreadsheet program requires. For example, how are rows and columns delimited?
2. Create an HP VEE program that collects the data, and then outputs the data to a file in an ASCII text format using the appropriate delimiters. Normally, the `To File` object is the best way to do this.

Let's look at an example of a program that collects (or simulates) 10 voltage readings, each with a time stamp, and outputs the data to a file that can be read by Microsoft Excel. The following program is saved as `manual21.vee` in your manual examples directory:



To File: myFile
[✓ Clear File At PreRun & Open]

```
WRITE TEXT a.Cnt, " ", a.Tm, " ", a.Vlts EOL
```

Function	DC Volts	Input R	10 Meg
Range	?	Auto Range	On
		Aperture	10 PLC
Auto Zero	On	Math Func	Off
Terminals	Front	Line Freq	60

The program works as follows:

1. The `For Count` and `Delay` objects trigger the following events approximately every 10 seconds:
   a. The `Counter` object advances by 1.
   b. The `Now` object outputs the current time (to within 1 second), which is converted to the Time Stamp format by the `To String` object.
   c. The `dmm (hp34401a @ (NOT LIVE))` instrument panel simulates a reading from a HP 34401A Multimeter.
2. The `Build Record` object builds a record for each reading (every 10 seconds), as follows: field `Cnt` is the count or test number, field `Tm` is the current time for the reading (in the format `hh:mm:ss`), and field `Vlts` is the simulated voltage reading returned by the instrument panel object.
3. The `To File` object writes the data out to a file in text form. In the transaction `WRITE TEXT a.Cnt," ",a.Tm," ",a.Vlts EOL`
   - `a.Cnt` represents "field `Cnt` of record `a`" (the count),
   - `" "` inserts a space (as a delimiter),
   - `a.Tm` represents "field `Tm` of record `a`" (the time),
   - `" "` inserts another space (as a delimiter),
   - `a.Vlts` represents "field `Vlts` of record `a`" (the reading), and
   - `EOL` inserts the end-of-line sequence (carriage-return/line-feed).

   Thus, each record is converted to string format and output (with spaces as delimiters) in the following format:

   ```
   1 11:29:06 5.687753697513882
   ```

When the program is finished, the file `myfile` will consist of 10 rows of data, for example:

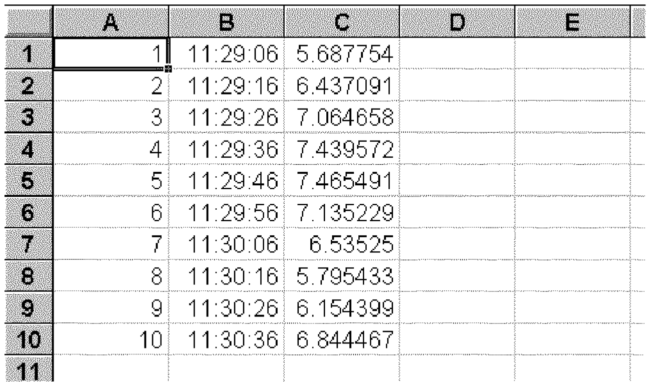
```
1 11:29:06 5.687753697513882
2 11:29:16 6.437091378803515
3 11:29:26 7.064658105870967
4 11:29:36 7.439572075814199
5 11:29:46 7.465490768920665
6 11:29:56 7.135229243023529
7 11:30:06 6.53525004955898
8 11:30:16 5.795432812581498
9 11:30:26 6.154398687554457
10 11:30:36 6.844467171462198
```

In each row, the first field (the count) is followed by a space (the delimiter), the second field (the time) is followed by another space, and the third field (the reading) is followed by an end-of-line sequence.

If you are using HP VEE for Windows on the PC, the output file will be a Windows ASCII file, which can be read directly into Microsoft Excel. Just open the file from Excel and, when asked, tell Excel that the file is delimited, and that the delimiter is the space.

If you are using HP VEE on a UNIX workstation, the data file will be a UNIX text file. You will have to transfer the UNIX text file to the PC. If you use `ftp` to transfer the file, it will automatically be converted into a Windows ASCII file, which Excel can read.

The data should appear as follows in the Excel spreadsheet:



	A	B	C	D	E
1	1	11:29:06	5.687754		
2	2	11:29:16	6.437091		
3	3	11:29:26	7.064658		
4	4	11:29:36	7.439572		
5	5	11:29:46	7.465491		
6	6	11:29:56	7.135229		
7	7	11:30:06	6.53525		
8	8	11:30:16	5.795433		
9	9	11:30:26	6.154399		
10	10	11:30:36	6.844467		
11					

By using appropriate delimiters you can create files that can be read by a variety of spreadsheet programs.

If you are using HP VEE for Windows you can automate the process by using DDE (Dynamic Data Exchange). Refer to *HP VEE Advanced Programming Techniques* for information about using DDE.

5

Controlling Instruments

Controlling Instruments

HP VEE provides several features to make it easy to control instruments from your HP VEE program. If you have read through *Getting Started with HP VEE*, you already know about the three primary types of HP VEE instrument I/O objects:

- **Instrument panel** objects provide a virtual "front panel" for your instrument. When you change parameters in the instrument panel object, the corresponding state of the instrument is changed. To use an instrument panel, you must first install an **instrument driver** (`.cid`) file for the instrument.
- **Component driver** objects provide similar capabilities to those of instrument panel objects, but without the front panel and without the direct control over the state of the instrument. However, component drivers offer slightly more efficient I/O performance. To use a component driver you must also install an instrument driver file for the instrument.
- **Direct I/O** objects provide a means of communicating with instruments without the use of instrument drivers. Also, `Direct I/O` objects offer the most efficient I/O performance.

This chapter shows how to use these objects to control instruments. We'll also cover how to use the `Device Event`, `Interface Event`, and `Bus I/O Monitor` objects to poll instruments and monitor bus messages. For a more detailed discussion of instrument I/O, refer to the "Using Instruments" chapter in *HP VEE Advanced Programming Techniques*.

NOTE

The examples in this chapter use instrument panels, component drivers, and `Direct I/O` objects for the HP 34401A Multimeter. If you have this instrument connected to your system, you can use the examples to communicate with the instrument by turning live mode `ON`. However, the examples will run without the instrument if you leave live mode `OFF`.

Using Instrument Drivers

Let's begin by looking at how you can use instrument drivers to control instruments and obtain data from them.

To Load an Instrument Driver

Before you can use either an instrument panel object or a component driver object, the appropriate instrument driver (`.cid`) file must be installed on your hard disk:

- *HP VEE for Windows*—A package of instrument driver disks is provided with HP VEE for Windows. Only a few instrument driver files are installed by default when you install HP VEE. To install additional instrument driver files, select the **`Install Drivers`** icon in the HP VEE group window. Follow the instructions in the *Installing HP VEE for Windows* manual.
- *HP VEE for the UNIX Workstation*—All of the instrument driver files supplied with HP VEE are installed when you install HP VEE. If you need to install additional instrument drivers, copy them to your hard disk in the directory **`/usr/lib/veetest/instruments/`**.

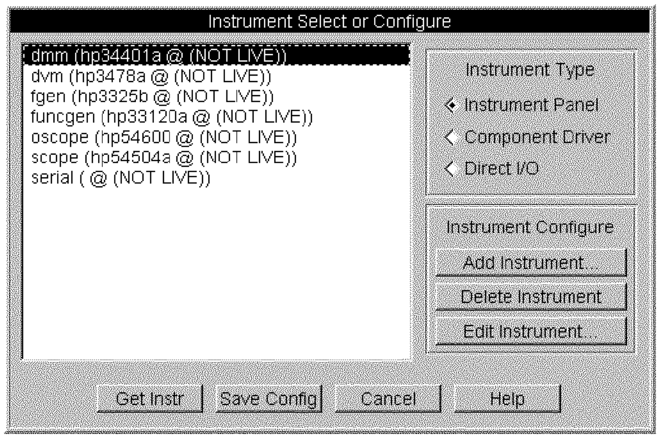
NOTE

From time to time additional, updated instrument driver files become available. For your convenience, Hewlett-Packard has established a computer bulletin board for downloading the latest drivers. Refer to `Latest Information` in *HP VEE Help* for further information.

To Use an Instrument Panel Object

To add an instrument panel object to the work area, use the **`Instrument Select or Configure`** dialog box. Let's try an example:

1. Select **`I/O`** ⟹ **`Instrument`** . . . . The dialog box appears:

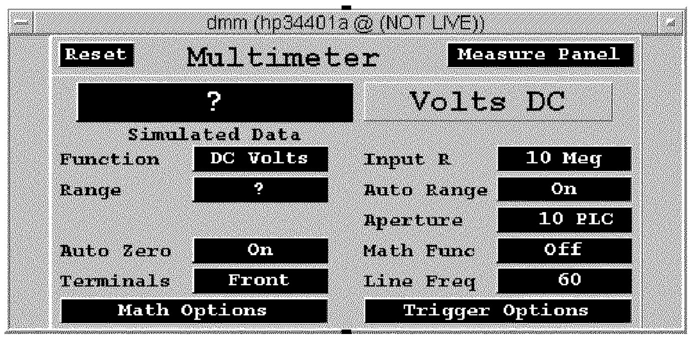


dmm (hp34401a @ (NOT LIVE))
dvm (hp3478a @ (NOT LIVE))
fgen (hp3325b @ (NOT LIVE))
funcgen (hp33120a @ (NOT LIVE))
oscope (hp54600 @ (NOT LIVE))
scope (hp54504a @ (NOT LIVE))
serial (@ (NOT LIVE))

The dialog box lists several I/O devices that are preconfigured as part of the HP VEE installation. Let's add an instrument panel for the HP 34401A Multimeter to the work area.

2. Select the **`dmm (hp34401a @ (NOT LIVE))`** field by clicking on it (if it is not already highlighted). Make sure that the **`Instrument Panel`** checkbox is selected under **`Instrument Type`**, and then click on the **`Get Instr`** button.

3. The outline of the instrument panel object appears. Place the outline where you want it in the work area, and click the mouse button. The object appears:



dmm (hp34401a @ (NOT LIVE))

Multimeter

? Volts DC

Simulated Data

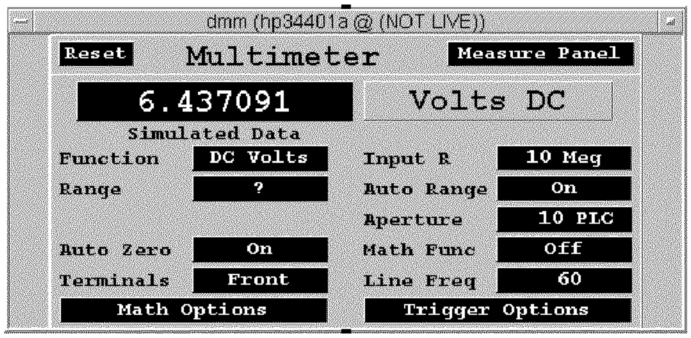
Function DC Volts
Range ?
Auto Zero On
Terminals Front

Input R 10 Meg
Auto Range On
Aperture 10 PLC
Math Func Off
Line Freq 60

NOTE

By default, the instrument panel is configured with live mode `OFF`, indicated by `(Not Live)` in the title bar. This makes it possible to run your I/O program even if an instrument is not present. Once you've tested the program and connected an instrument, you can set live mode to `ON`.

4. With live mode `OFF`, the `hp34401a` instrument panel generates simulated data. Try clicking on the `?` in the display field. A simulated reading appears:



dmm (hp34401a @ (NOT LIVE))

Reset | Multimeter | Measure Panel

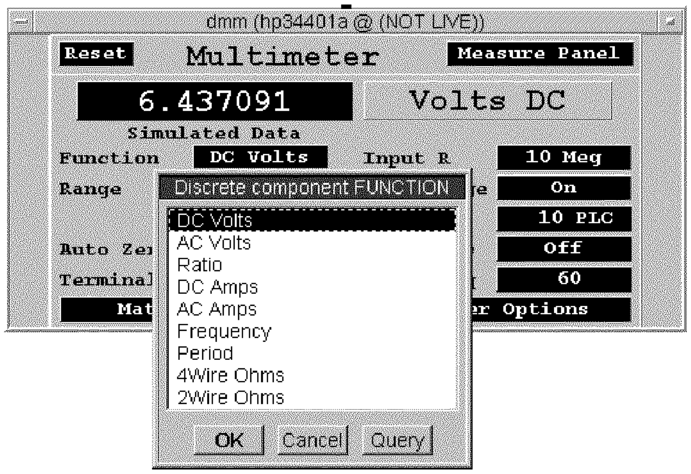
6.437091 | Volts DC

Simulated Data

Function	DC Volts	Input R	10 Meg
Range	?	Auto Range	On
		Aperture	10 PLC
Auto Zero	On	Math Func	Off
Terminals	Front	Line Freq	60

Math Options | Trigger Options

5. The `Measure Panel` is shown by default, which allows you to select the `Function`, `Range`, and so forth. Let's try changing the `Function`. Click on the `Function` button labeled `DC Volts`. The following dialog box appears:



dmm (hp34401a @ (NOT LIVE))

Discrete component FUNCTION

DC Volts
AC Volts
Ratio
DC Amps
AC Amps
Frequency
Period
4Wire Ohms
2Wire Ohms

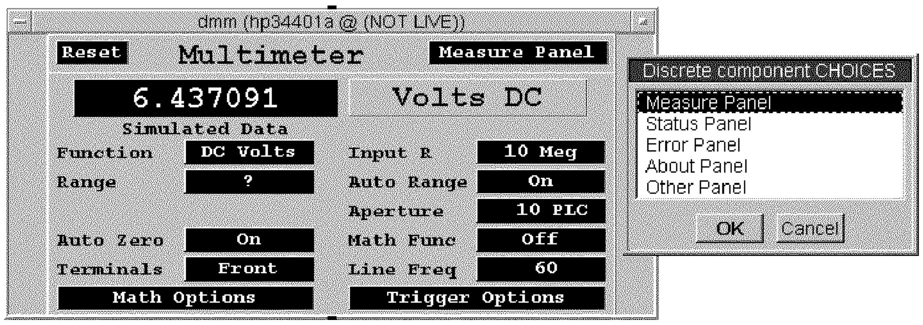
OK | Cancel | Query

6. To change the function to **DC Amps**, double-click on **DC Amps** in the list. To change back to **DC Volts**, click on the **Function** button, now labeled **DC Amps**, and then double-click on **DC Volts** in the list.

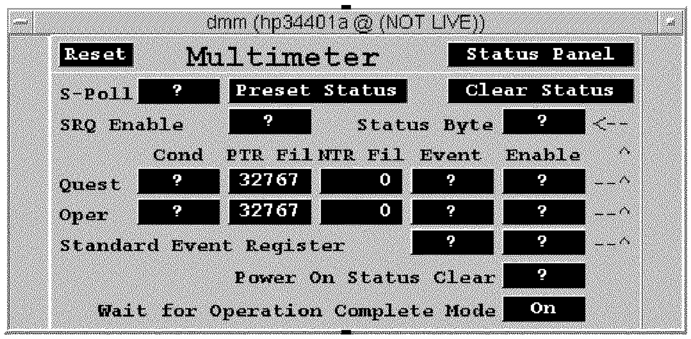
NOTE

With live mode **OFF**, none of these changes affect an instrument, even if one is connected to your system. However, if live mode is **ON** and you have an HP 34401A Multimeter connected to the HP-IB (correctly addressed and configured), the changes on the HP VEE instrument panel object will automatically change the state of the instrument.

7. The **Measure Panel** is only one of several panels that you can select. Click on the **Measure Panel** button at the upper-right corner of the object. The following dialog box appears:



8. Double-click on `Status Panel` in the list. The `Status Panel` appears:



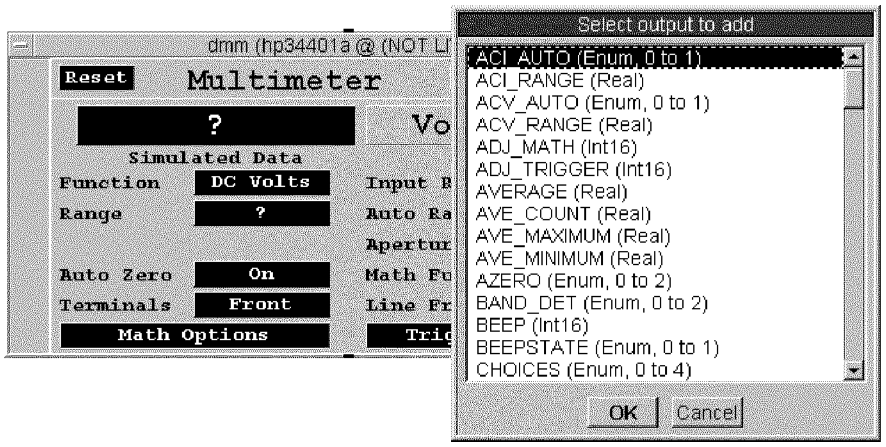
9. To go back to the `Measure Panel`, click on the `Status Panel` button and double-click on `Measure Panel`.

The various panels in an instrument panel object allow you to control almost all functions of an instrument. However, you need to be familiar with a particular instrument in order to use the instrument panel effectively. Refer to the instrument owner's manual for further information. For information about an instrument panel object, select `Help` from the object menu.

To Read Data with an Instrument Panel Object

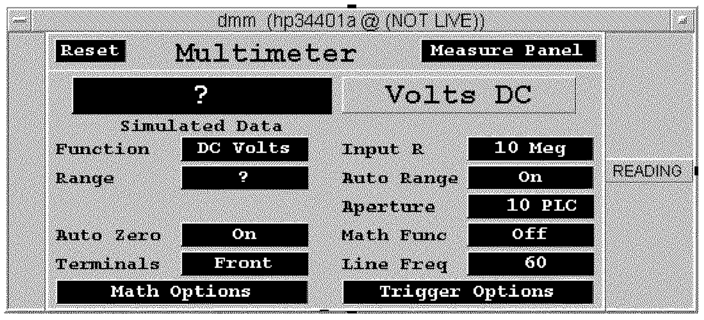
Let's create a simple program that displays simulated voltage readings from an instrument panel object for an HP 34401A Multimeter.

1. Add an instrument panel object for the HP 34401A as described in the previous section.
2. Select `Add Terminal` $\Longrightarrow$ `Data Output` from the object menu of the instrument panel. The following dialog box appears:



ACI_AUTO (Enum, 0 to 1)
ACI_RANGE (Real)
ACV_AUTO (Enum, 0 to 1)
ACV_RANGE (Real)
ADJ_MATH (Int16)
ADJ_TRIGGER (Int16)
AVERAGE (Real)
AVE_COUNT (Real)
AVE_MAXIMUM (Real)
AVE_MINIMUM (Real)
AZERO (Enum, 0 to 2)
BAND_DET (Enum, 0 to 2)
BEEP (Int16)
BEEPSTATE (Enum, 0 to 1)
CHOICES (Enum, 0 to 4)

3. Scroll down the list and select the `Reading` output terminal (double-click on it):

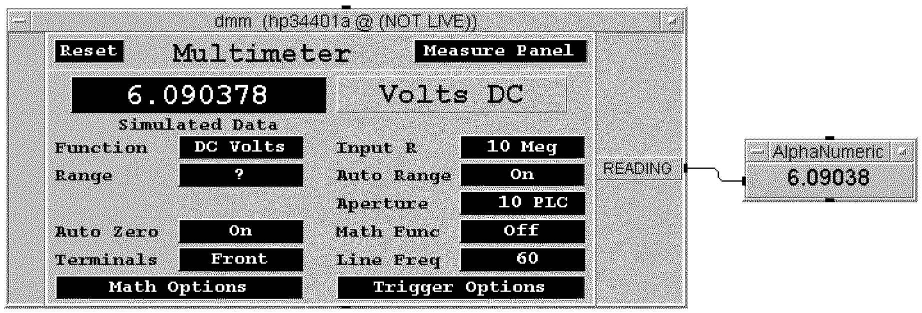


Function	DC Volts	Input R	10 Meg
Range	?	Auto Range	On
		Aperture	10 PLC
Auto Zero	On	Math Func	Off
Terminals	Front	Line Freq	60

NOTE

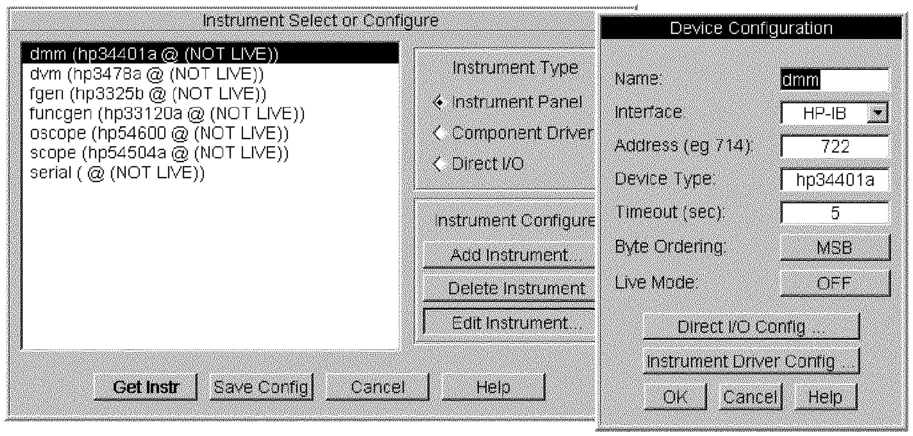
Another output terminal selection is `Readings`, which is different. The `Reading` terminal outputs a scalar, while the `Readings` terminal outputs an array of data, provided the instrument is configured to output multiple readings.

4. Add an `AlphaNumeric` display object to the work area, connect it as shown below, and run the program.



Since live mode is `OFF`, the data is simulated, which is indicated by the `Simulated Data` label (and by `(NOT LIVE)` in the title bar).

5. To turn live mode `ON`, select `I/O` ⟹ `Instrument` . . . , select `dmm (hp34401a @ (NOT LIVE))` in the list (if it is not already highlighted), and then click on `Edit Instrument`. The `Device Configuration` dialog box appears showing the configuration for the HP 34401A Multimeter:



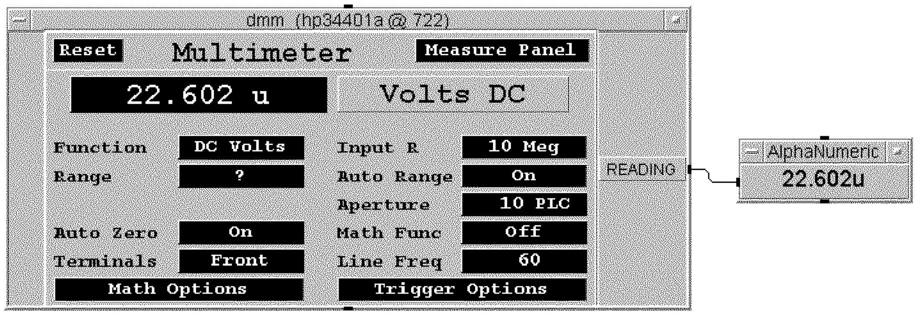
Instrument Select or Configure

dmm (hp34401a @ (NOT LIVE))
dvm (hp3478a @ (NOT LIVE))
fgen (hp3325b @ (NOT LIVE))
funcgen (hp33120a @ (NOT LIVE))
oscope (hp54600 @ (NOT LIVE))
scope (hp54504a @ (NOT LIVE))
serial (@ (NOT LIVE))

Device Configuration

Name:	dmm
Interface:	HP-IB
Address (eg 714):	722
Device Type:	hp34401a
Timeout (sec):	5
Byte Ordering:	MSB
Live Mode:	OFF

6. If the address shown (the default is `722`) doesn't match the address of the instrument, enter the correct address. (For information on addressing a particular instrument, refer to the instrument owner's manual.)
7. Click on the button to the right of `Live Mode`. The button toggles from `OFF` to `ON`. Then click on `OK`.
8. Click on the `Save Config` button in the `Instrument Select or Configure` dialog box. The instrument panel now has live mode `ON`, as shown below:



dmm (hp34401a @ 722)

Multimeter

22.602 u Volts DC

Function	DC Volts	Input R	10 Meg
Range	?	Auto Range	On
		Aperture	10 PLC
Auto Zero	On	Math Func	Off
Terminals	Front	Line Freq	60

AlphaNumeric
22.602u

The new title, `dmm (hp34401a @ 722)`, shows that live mode is `ON` and that the instrument panel is configured to communicate with an HP 34401A Multimeter at HP-IB address `722`. Of course, the program will time out unless there actually is an HP 34401A at address `722`.

At this point, if you have an HP 34401A Multimeter, you can connect it and run the program. Otherwise, use the `Device Configuration` dialog box (`I/O` $\Longrightarrow$ `Instrument` ... $\Longrightarrow$ `Edit Instrument`) to return to live mode `OFF`.

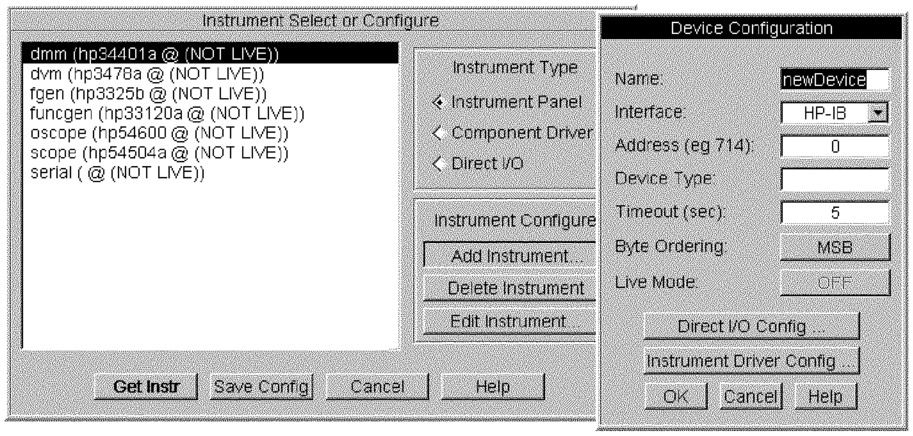
NOTE

You can add a wide variety of additional input and output terminals to an instrument panel object. For further information, refer to the "Using Instruments" chapter in the *HP VEE Advanced Programming Techniques* manual.

To Set Up an Additional Instrument

As we've seen in the previous sections, it is easy to use an instrument panel object for an I/O device once that device is listed in the **`Instrument Select or Configure`** dialog box. Now let's add another instrument configuration to the dialog box.

1. Select **`I/O`** ⟹ **`Instrument`** . . . . The **`Instrument Select or Configure`** dialog box appears.
2. Click on **`Add Instrument`**. The **`Device Configuration`** dialog box appears:



Instrument Select or Configure

dmm (hp34401a @ (NOT LIVE))
dvm (hp3478a @ (NOT LIVE))
fgen (hp3325b @ (NOT LIVE))
funcgen (hp33120a @ (NOT LIVE))
oscope (hp54600 @ (NOT LIVE))
scope (hp54504a @ (NOT LIVE))
serial (@ (NOT LIVE))

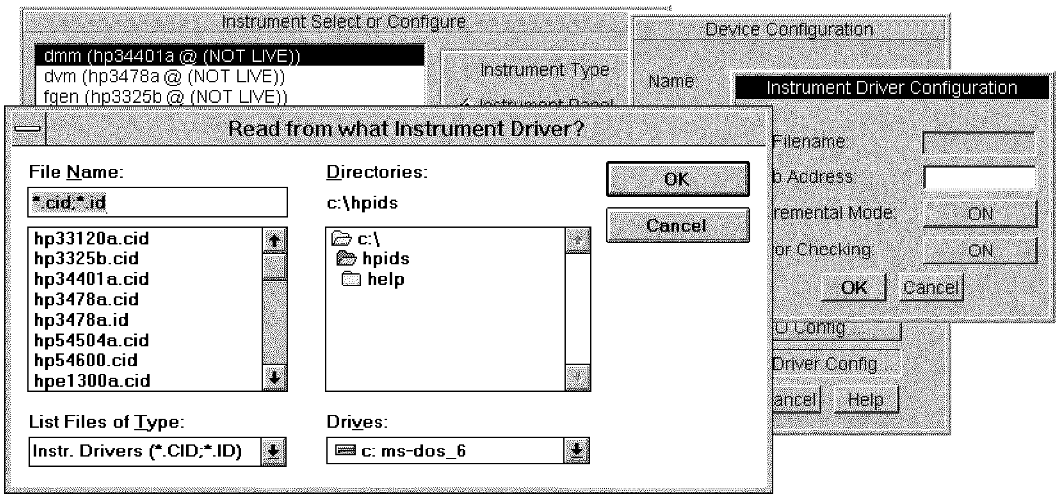
Instrument Type: Instrument Panel (selected), Component Driver, Direct I/O

Device Configuration

Name: newDevice
Interface: HP-IB
Address (eg 714): 0
Device Type:
Timeout (sec): 5
Byte Ordering: MSB
Live Mode: OFF

3. This dialog box allows you to configure the new instrument. For our example, let's use the default name (**`newDevice`**) and interface (**`HP-IB`**). To select an instrument driver file, click on the **`Instrument Driver Config`** button.

4. The `Instrument Driver Configuration` dialog box appears. Click on the `ID Filename` button (the "blank" button) and the `Read from what Instrument Driver?` dialog box appears:



(The HP VEE for Windows dialog box is shown. The UNIX dialog box is different, but the procedure is similar.)

5. Let's select the instrument driver file named `hp34401a.cid`. (This is the compiled instrument driver ("`.cid`") file for the HP 344401A Multimeter.)

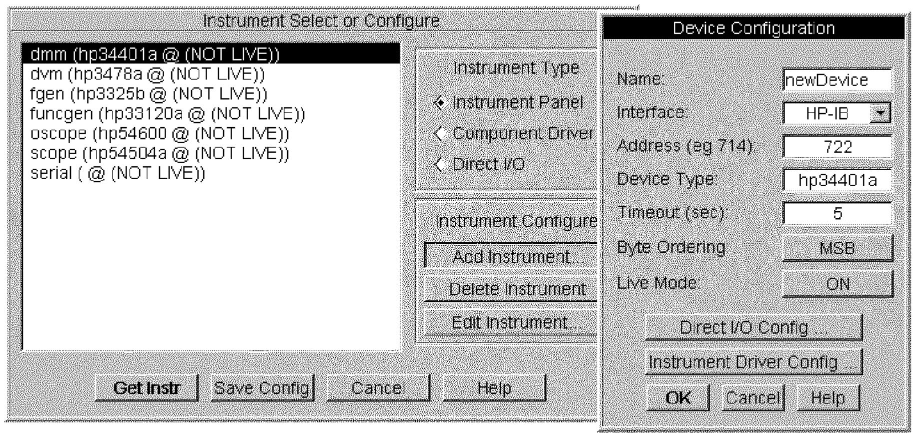
   a. Select `hp34401a.cid` in the list of file names. Click on the file name and then click on `OK` to exit the `Read from what Instrument Driver?` dialog box.

   b. Click on `OK` to exit the `Instrument Driver Configuration` dialog box and return to the `Device Configuration` dialog box.

NOTE

If no instrument driver file is listed for the instrument you want to add, you must load the appropriate file. Refer to "To Load an Instrument Driver", earlier in this chapter, for further information.

6. Now enter an HP-IB address for the instrument in the `Address` field, for example `722`. Click on the field, enter the address, and then tab to the next field. Note that the `Live Mode` button automatically toggles to `ON` when you enter an address.


Instrument Select or Configure

dmm (hp34401a @ (NOT LIVE))
dvm (hp3478a @ (NOT LIVE))
fgen (hp3325b @ (NOT LIVE))
funcgen (hp33120a @ (NOT LIVE))
oscope (hp54600 @ (NOT LIVE))
scope (hp54504a @ (NOT LIVE))
serial (@ (NOT LIVE))

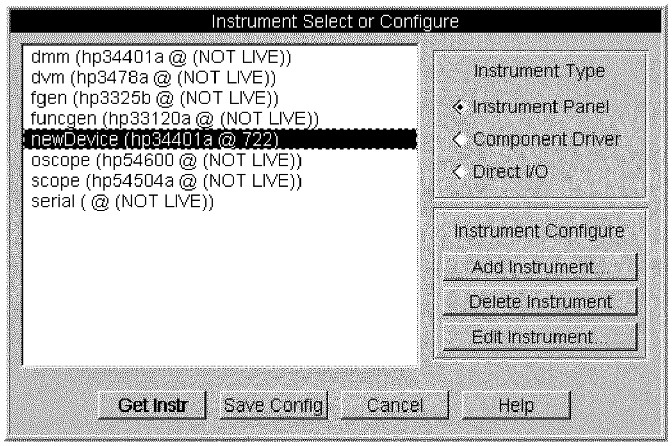
Device Configuration

Name:	newDevice
Interface:	HP-IB
Address (eg 714):	722
Device Type:	hp34401a
Timeout (sec):	5
Byte Ordering	MSB
Live Mode:	ON

NOTE

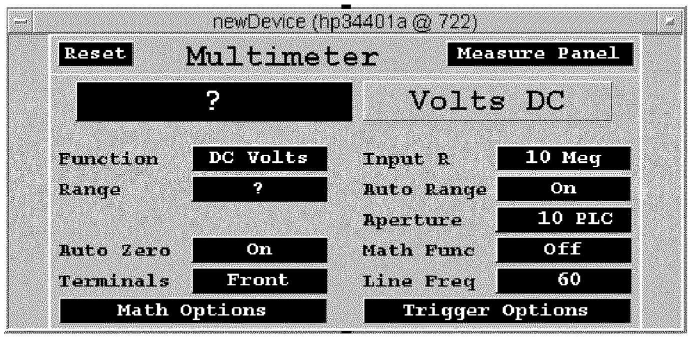
For information about determining what address to use for a particular instrument, refer to the "Using Instruments" chapter in *HP VEE Advanced Programming Techniques*, and to the owner's manual for the instrument.

7. Click on `OK` to exit the `Device Configuration` dialog box. The new selection, `newDevice (hp34401a @ 722)`, is highlighted in the `Instrument Select or Configure` dialog box:



dmm (hp34401a @ (NOT LIVE))
dvm (hp3478a @ (NOT LIVE))
fgen (hp3325b @ (NOT LIVE))
funcgen (hp33120a @ (NOT LIVE))
newDevice (hp34401a @ 722)
oscope (hp54600 @ (NOT LIVE))
scope (hp54504a @ (NOT LIVE))
serial (@ (NOT LIVE))

8. If the `Instrument Panel` checkbox is not already selected, click on it.
9. Click on the `Get Instr` button to add the instrument panel object to the work area:



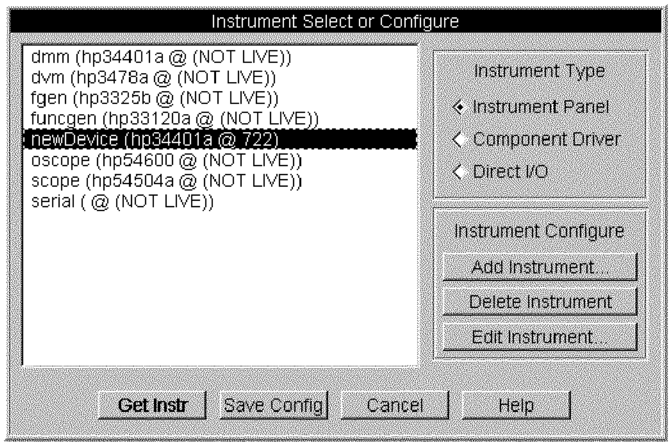
? Volts DC

Function	DC Volts	Input R	10 Meg
Range	?	Auto Range	On
		Aperture	10 PLC
Auto Zero	On	Math Func	Off
Terminals	Front	Line Freq	60

To Delete an Instrument Configuration

In the previous sections we've seen how to add an additional instrument to the **`Instrument Select or Configure`** dialog box, and how to edit an instrument configuration. You can also delete an instrument from the dialog box. To delete the **`newDevice`** configuration of the previous example:

1. Select **`I/O`** $\Longrightarrow$ **`Instrument`** . . . . Then select the instrument configuration **`newDevice (hp34401a @ 722)`** by clicking on it:



Instrument Select or Configure

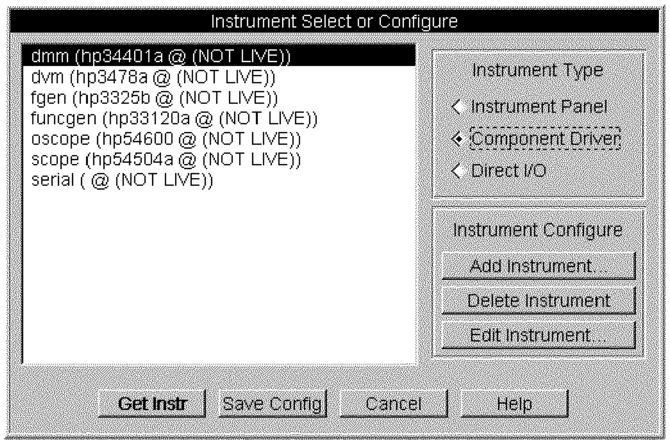
```
dmm (hp34401a @ (NOT LIVE))
dvm (hp3478a @ (NOT LIVE))
fgen (hp3325b @ (NOT LIVE))
funcgen (hp33120a @ (NOT LIVE))
newDevice (hp34401a @ 722)
oscope (hp54600 @ (NOT LIVE))
scope (hp54504a @ (NOT LIVE))
serial ( @ (NOT LIVE))
```

2. Click on the **`Delete Instrument`** button. The selection is removed from the list.
3. Click on the **`Save Config`** button to close the dialog box and save the new configuration.

To Use a Component Driver Object

Although instrument panel objects give you direct control over an instrument, sometimes you don't need such direct control. If so, you can gain a slight increase in performance by using a component driver object. A component driver object uses the same instrument driver file as an instrument panel object for the same instrument. Let's add a component driver object for the HP 34401A Multimeter.

1. Select `I/O` ⟹ `Instrument` . . . . The `Instrument Select or Configure` dialog box appears.
2. Select the **`dmm (hp34401a @ (NOT LIVE))`** configuration (if it is not already highlighted). Then click on the checkbox for `Component Driver`, as shown below:

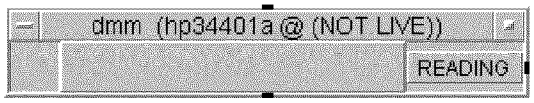


Instrument Select or Configure

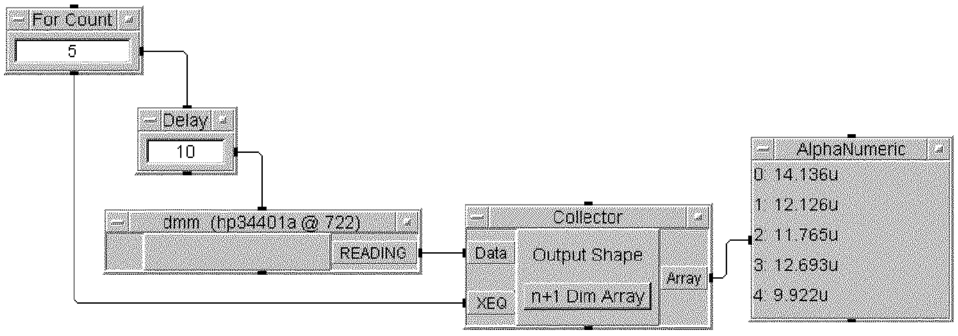
dmm (hp34401a @ (NOT LIVE))
dvm (hp3478a @ (NOT LIVE))
fgen (hp3325b @ (NOT LIVE))
funcgen (hp33120a @ (NOT LIVE))
oscope (hp54600 @ (NOT LIVE))
scope (hp54504a @ (NOT LIVE))
serial (@ (NOT LIVE))

3. Click on the `Get Instr` button and place the component driver object in the work area.
4. The component driver object appears as an icon. Double-click on the icon to display the open view.

5. Add a **Reading** terminal to the component driver object (object menu ⟹ **Add Terminal** ⟹ **Data Output**).



The following program (**manual22.vee** in your manual examples directory) uses a component driver to collect a series of data points. (With live mode **OFF**, this program collects a series of **0** values. If you have an HP 34401A Multimeter connected to your system, you can use this program to collect real data.)



```
0 14.136u
1 12.126u
2 11.765u
3 12.693u
4 9.922u
```

In the program, the **For Count** and **Delay** objects cause the component driver object to input a reading approximately every 10 seconds until 5 readings are taken. The **Collector** object collects the readings until its **XEQ** terminal is activated, and then outputs an array. This is a useful technique for taking readings on a timed basis, as long as you don't need to take the readings at an exact time.

Using Direct I/O

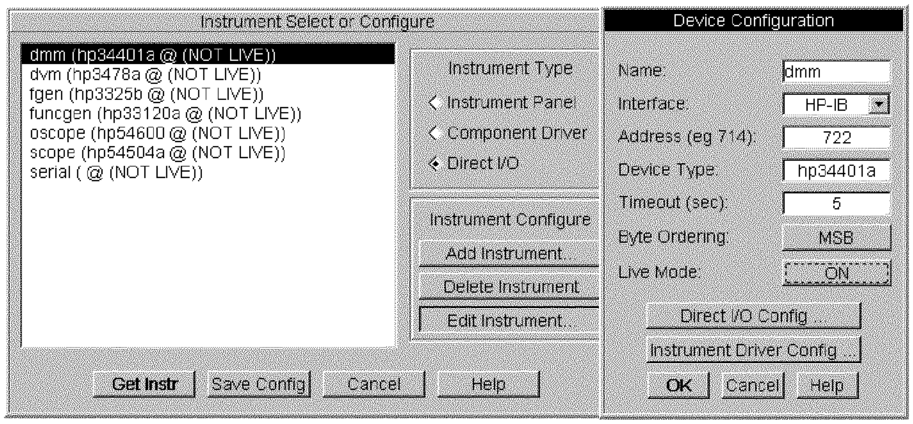
Direct I/O allows you to directly specify the messages to be sent an instrument, and to directly read the information sent back by an instrument. To do this, use I/O **transactions** in the `Direct I/O` object. Direct I/O provides the following advantages:

1. Direct I/O allows you to communicate with *any* instrument connected to one of the supported interfaces. You don't need an instrument driver to use direct I/O.
2. Direct I/O provides the fastest instrument I/O communication.

To Use a Direct I/O Object

To add a direct I/O object, configured to to communicate with an HP 34401A Multimeter at HP-IB address `722`, to the work area:

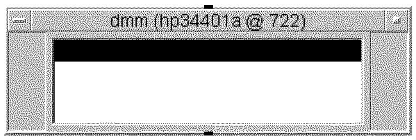
1. Select `I/O` $\Longrightarrow$ `Instrument` . . . . The `Instrument Select or Configure` dialog box appears.
2. Click on the checkbox for `Direct I/O` and select `dmm (hp34401a @ NOT LIVE))`.
3. Click on `Edit Instrument` . . . , and then turn on live mode by clicking on the `Live Mode` button.



Name: dmm
Interface: HP-IB
Address (eg 714): 722
Device Type: hp34401a
Timeout (sec): 5
Byte Ordering: MSB
Live Mode: ON

You don't need to configure an instrument driver, but you can configure parameters for direct I/O (such as the EOL sequence) using **Direct I/O Config**. Refer to the "Using Instruments" chapter in *HP VEE Advanced Programming Techniques* for further information.

4. Click on **OK**, and then click on **Get Instr**. Place the direct I/O object in the work area:

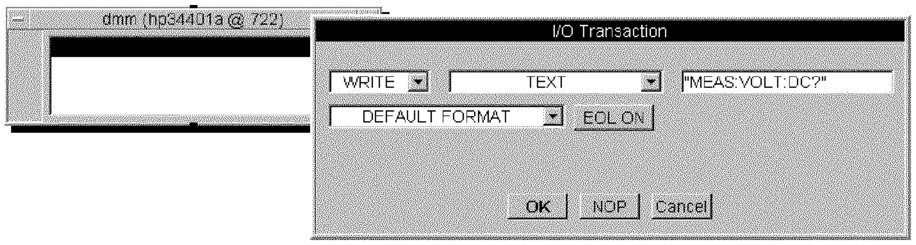


The next section shows how to use the direct I/O object in a simple program to read data from an HP 34401A Multimeter.

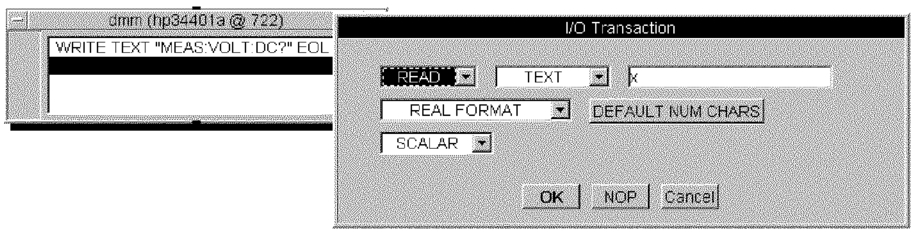
To Read Data Using Direct I/O

As an example, let's use the direct I/O object to read data from an HP 34401A Multimeter.

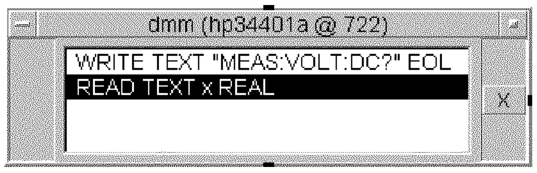
1. Add a direct I/O object, configured for address **722**, to the work area as shown in the previous section.
2. The direct I/O object uses **transactions** to communicate with an instrument. Double-click on the "blank" transaction (the highlighted field) in the object. The **I/O Transaction** dialog box appears with the variable **a** highlighted. Enter the string **"MEAS:VOLT:DC?"** in its place:



3. Click on **OK** to create the transaction **WRITE TEXT "MEAS:VOLT:DC?"**. Then select **Add Trans** from the object menu to add a second transaction. Change this transaction to a **READ TEXT** transaction as shown:



4. Click on **OK**. Now add a data output terminal to the object. (Use the object menu or the shortcut, Ctrl+A.)



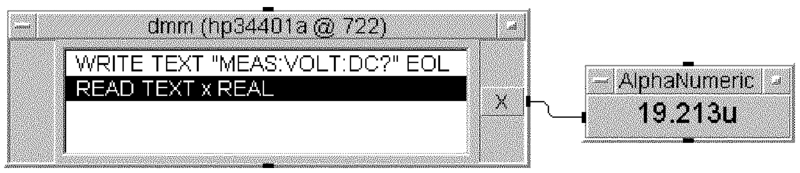
```
WRITE TEXT "MEAS:VOLT:DC?" EOL
READ TEXT x REAL
```

The direct I/O object now has two transactions:

a. **WRITE TEXT "MEAS:VOLT:DC?" EOL** writes the command string **"MEAS:VOLT:DC?"** to the instrument at address 722.

b. **READ TEXT x REAL** reads text data in the real format from the instrument at address 722, and outputs the data on the **X** data output terminal.

5. To complete the program, add an **AlphaNumeric** display object to the work area, and connect it as shown below. (You can run this program if you have an HP 34401A Multimeter connected to the HP-IB at address 722.)



```
WRITE TEXT "MEAS:VOLT:DC?" EOL
READ TEXT x REAL
```

```
19.213u
```

To the HP 34401A Multimeter, the command string **MEAS:VOLT:DC?** means "output one dc voltage reading." The reading is read by the direct I/O object and output to the **AlphaNumeric** display.

The commands to control an instrument depend upon the particular instrument. Refer to the owner's manual for your instrument for further information. For further information on transaction I/O, refer to the "Using Transaction I/O" chapter in *HP VEE Advanced Programming Techniques*.

Handling Service Requests and Bus Messages

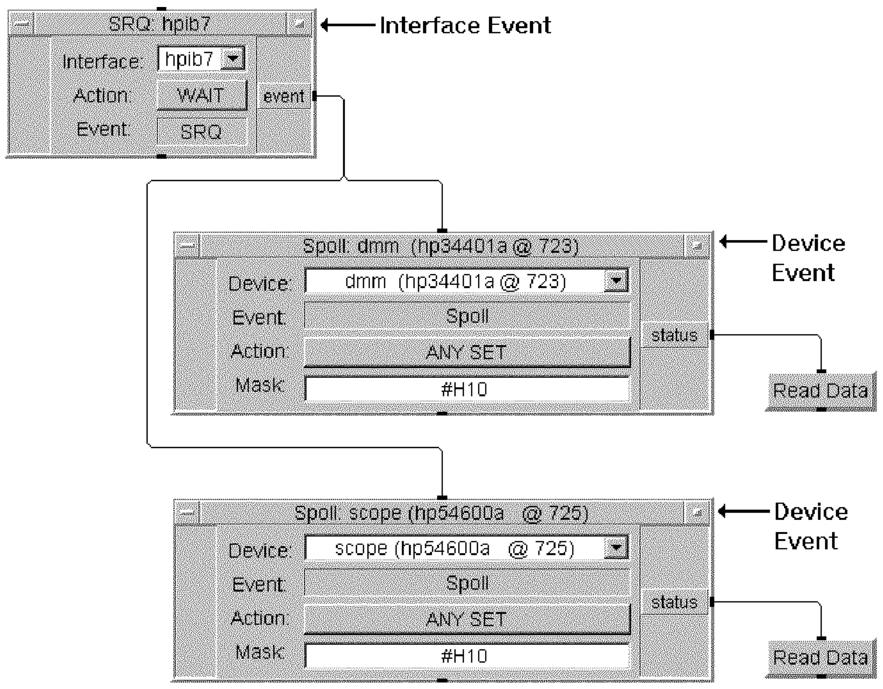
HP VEE provides objects to handle service requests and to monitor the instrument bus for messages. This section shows how to use these objects. For further information, refer to the "Using Instruments" chapter in *HP VEE Advanced Programming Techniques*.

To Poll HP-IB Instruments

From time to time an HP-IB instrument sends a service request ("SRQ") on the bus. HP VEE provides two devices that help you determine what device has requested service, and what the problem is. Both of these objects are found under `I/O` ⟹ `Advanced I/O`.

- The `Interface Event` object reports an event for the entire bus. For example, if `Interface Event` is set to report an SRQ for `hpib7`, any device on the interface at select code 7 can cause an event. You can also use the `Interface Event` object to detect events on the serial and VXI interfaces, if such instruments are configured.
- The `Device Event` object returns the status byte from a particular instrument. On the HP-IB, use `Device Event` objects to poll the individual devices on the bus.

The following program shows how you can use the `Interface Event` and `Device Event` objects to process HP-IB service requests:



In the program the `Interface Event` object waits for an event on `hpib7`, that is, on the HP-IB interface at select code 7. However, the `Interface Event` object does not indicate which device on the bus caused the event.

The `event` output of the `Interface Event` object activates both of the `Device Event` objects, which individually poll two HP-IB instruments (the HP 34401A and HP 54600A). Each `Device Event` object has its action set to `Any Set` and its mask set to `#H10`. `Any Set` causes the object to wait for an event to occur, and then perform a logical `AND` of the returned value with the mask `#H10`, which corresponds to bit 4 in the status byte (`DAV`, or "data available"). The result is returned on the `status` output terminal. For further information, refer to the `Device Event` section in the *HP VEE Reference* manual.

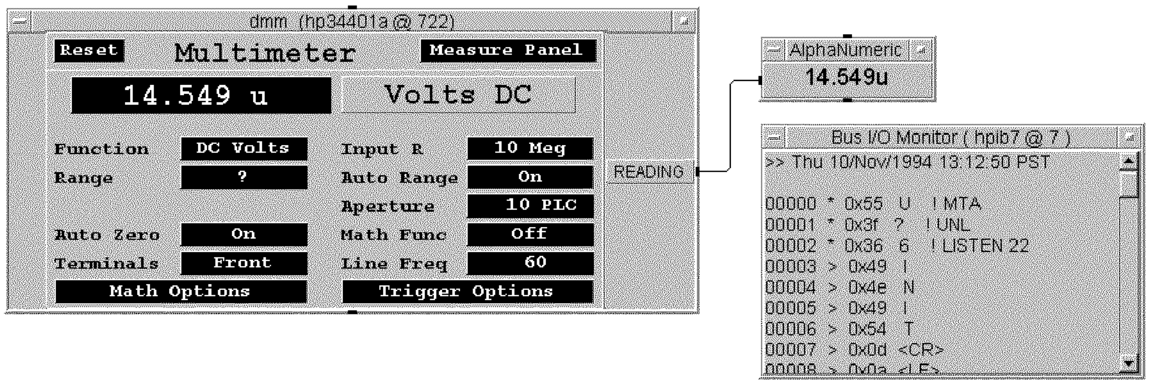
To Monitor Bus Messages

HP VEE communicates with an instrument by means of a series of messages. On the HP-IB, the bus messages follow the IEEE-488 standard. Normally, these messages are not seen. However, when you debug your I/O program you may want to monitor the messages on the bus. To do this, add a `Bus I/O Monitor` object to your program. You can also use this object to monitor VXI or serial interface messages.

For example, to add a `Bus I/O Monitor` object to monitor the HP-IB at select code 7:

1. Select `I/O` ⟹ `Bus I/O Monitor`. The `Select I/O Channel` dialog appears.
2. Select `hpib7` and click on `OK`.
3. Place the object in the work area.

Here is an example showing the `Bus I/O Monitor` object monitoring the bus:



```
>> Thu 10/Nov/1994 13:12:50 PST

00000 * 0x55  U   ! MTA
00001 * 0x3f  ?   ! UNL
00002 * 0x36  6   ! LISTEN 22
00003 > 0x49  I
00004 > 0x4e  N
00005 > 0x49  I
00006 > 0x54  T
00007 > 0x0d  <CR>
00008 > 0x0a  <LF>
```

Each time you run the program, the sequence of bus messages is recorded. When the program stops, you can scroll through the messages. By default, the `Bus I/O Monitor` object can save 256 lines of information in its buffer. However, you can change the buffer size. Refer to the `Bus I/O Monitor` section in the *HP VEE Reference* manual for further information.

The `Bus I/O Monitor` object slows down your program, and is generally used for diagnostic purposes. Many programmers create a program using an instrument panel or component driver to control an instrument, and then use `Bus I/O Monitor` to “capture” the commands and messages used to communicate with the instrument. You can then use these commands and messages to communicate with the instrument using direct I/O, which provides increased I/O performance.

6

Displaying Data

Displaying Data

HP VEE provides several types of display objects, found under the `Display` menu. This chapter shows how to use several of these objects to display various types of data.

NOTE

For information on using dialog boxes, `Picture` objects, and panel views to create an operator interface for your program, refer to *Building an Operator Interface with HP VEE*.

Displaying Alphanumeric Information

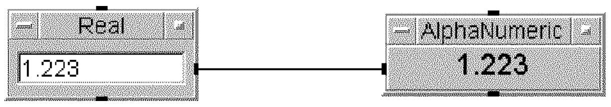
You can use the **AlphaNumeric** and **Logging AlphaNumeric** objects to display alphanumeric data in HP VEE programs.

To Display a Scalar Value

To display a scalar value:

1. Add an **AlphaNumeric** display to the work area (**Display** $\Longrightarrow$ **AlphaNumeric**).
2. Connect the data input pin of the **AlphaNumeric** object to a source of scalar data (for example, a **Real** constant object).
3. Run the program to display the data.

In the example below, the **AlphaNumeric** object displays the scalar real value `1.223` from the **Real** constant object.

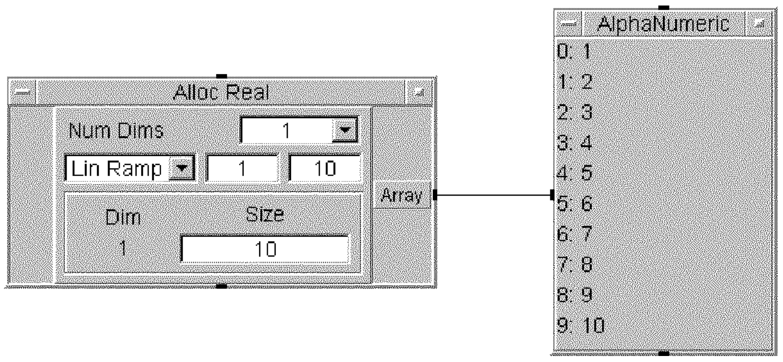


To Display an Array of Values

You can also use the **AlphaNumeric** object to display an array.

1. Add an **AlphaNumeric** display to the work area (**Display** ⟹ **AlphaNumeric**).
2. Connect the data input pin of the **AlphaNumeric** object to a source of array data (for example, an **Alloc Real** object).
3. Run the program to display the data.
4. Resize the **AlphaNumeric** object to show the entire array.

In the example below the **AlphaNumeric** object displays a one-dimensional array output by the **Alloc Real** object:



```
0: 1
1: 2
2: 3
3: 4
4: 5
5: 6
6: 7
7: 8
8: 9
9: 10
```

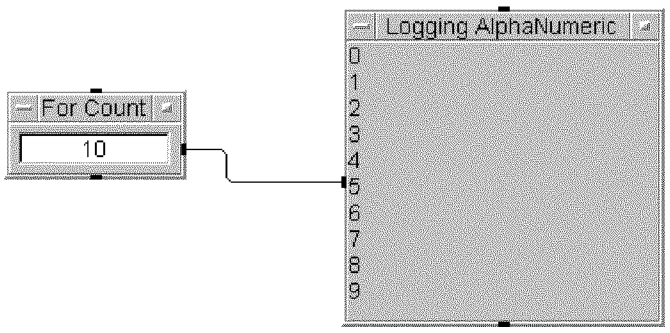
This array is a single data container consisting of 10 elements.

To Display a Series of Values in a Log

You can use the **Logging AlphaNumeric** object to display a series of values in a log.

1. Add a **Logging AlphaNumeric** display to the work area (**Display** ⟹ **Logging AlphaNumeric**).
2. Connect the data input pin of the **Logging AlphaNumeric** object to a repeating source of data (for example, a **For Count** object).
3. Run the program to display the log of data.

In the following example, the **Logging AlphaNumeric** object displays a log of values as the **For Count** object repeats 10 times.



```
0
1
2
3
4
5
6
7
8
9
```

This log of values consists of 10 scalar values from the **For Count** object. Note the difference between this log and the array displayed in the previous section:

- The **AlphaNumeric** object clears itself and displays only the new data received each time it executes.
- The **Logging AlphaNumeric** object keeps a “history log” of the data it receives.

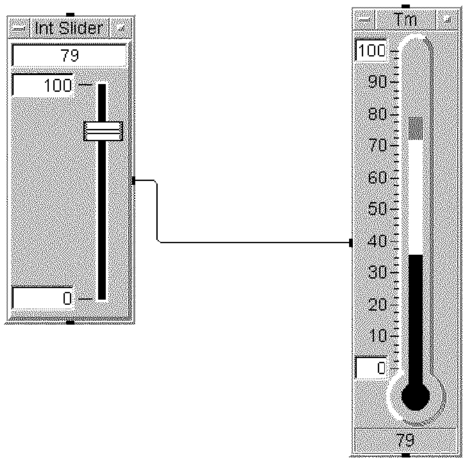
Using the Indicator Displays

HP VEE provides five indicator display objects under **`Display`** ⟹ **`Indicator`** in the menu. You can use these objects to display an analog indication of data, or to display color alarms.

To Show an Analog Display of Data

You can use a **`Meter`**, **`Thermometer`**, **`Fill Bar`**, or **`Tank`** display object to display an analog indication of data. For example, the **`Thermometer`** object displays an analog indication that resembles the "mercury" in a conventional thermometer. Let's look at how to use the **`Thermometer`** object:

1. Add a **`Thermometer`** object to the work area (**`Display`** ⟹ **`Indicator`** ⟹ **`Thermometer`**).
2. Connect an **`Integer Slider`** object to the data input pin of the **`Thermometer`**. Move the slider to select a value, and run the program:

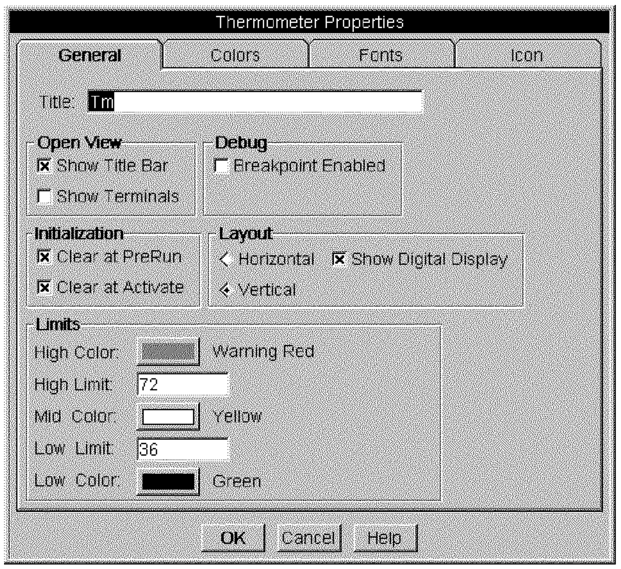


Note how the “mercury” rises to show the value set by the slider. (Press **Run** each time you move the slider.) The value is also shown by the digital display at the bottom of the **Thermometer**. Note that the “mercury” changes color as you move up the scale (green, yellow, and then red).

3. Try changing the range of the **Thermometer** scale. Click on the lower range field (0) and enter a new value. Do the same for the upper range field (100). Note that the thermometer scale changes to conform to the new range.

Now let’s try changing the configuration of the **Thermometer** object:

1. Click on **Edit Properties** from the object menu of the **Thermometer**. The **Thermometer Properties** dialog box appears:



Title: Tm

Open View
[✓] Show Title Bar
[ ] Show Terminals

Debug
[ ] Breakpoint Enabled

Initialization
[✓] Clear at PreRun
[✓] Clear at Activate

Layout
Horizontal [✓ Vertical]
[✓] Show Digital Display

Limits
High Color: Warning Red
High Limit: 72
Mid Color: Yellow
Low Limit: 36
Low Color: Green

2. Try changing a few parameters:
   - From the **Layout** section of the dialog box you can select either a vertical (the default) or a horizontal layout for the **Thermometer**. You can also turn the digital display on or off.
   - From the **Limits** section of the dialog box you can select a color for each range (low, mid, and high). Click on a color button to show the

possible colors. You can also set the lower and upper limit values to determine where each color range begins.

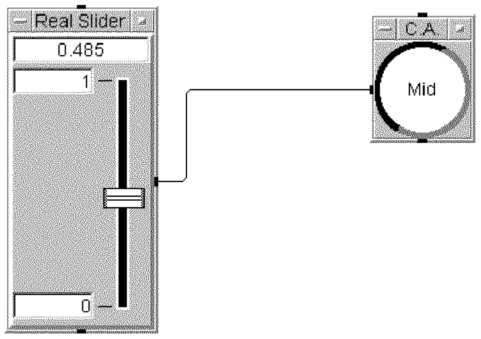
3. Click on **OK** to see the effect of your changes.

The **Tank** and **Fill Bar** objects work very much like the **Thermometer**. The **Meter** object also provides an analog indication, but with a needle resembling an analog panel meter. All of these objects provide three-color range indication.

To Show a Color Alarm

The **Color Alarm** object displays one of three pre-set color indications (green=**Low**, yellow=**Mid**, and red=**High**) depending on the value on its data input pin. Let's look at how to use the **Color Alarm**.

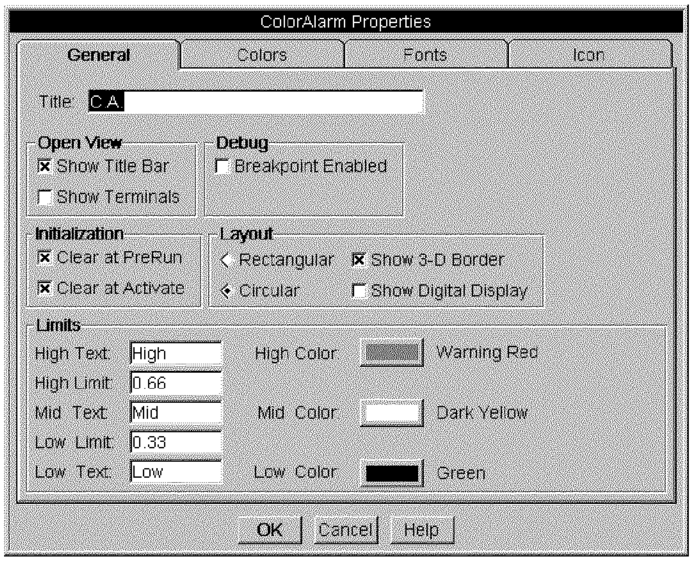
1. Add a **Color Alarm** object to the work area (**Display** ⟹ **Indicator** ⟹ **Color Alarm**).
2. Connect a **Real Slider** object to the data input pin of the **Color Alarm**. Move the slider to select a value, and run the program:



Note how the color and text displayed by the **Color Alarm** change as you change the slider position. (Press **Run** each time you move the slider.)

Now let's try changing the configuration of the `Color Alarm` object:

1. Click on `Edit Properties` from the object menu of the `Color Alarm` object. The `Color Alarm Properties` dialog box appears:



Title: C A

Open View
[✓ Show Title Bar]
Show Terminals

Debug
Breakpoint Enabled

Initialization
[✓ Clear at PreRun]
[✓ Clear at Activate]

Layout
Rectangular [✓ Circular]
[✓ Show 3-D Border]
Show Digital Display

Limits

High Text:	High	High Color:	Warning Red
High Limit:	0.66		
Mid Text:	Mid	Mid Color:	Dark Yellow
Low Limit:	0.33		
Low Text:	Low	Low Color:	Green

2. Try changing a few parameters:

   - From the `Layout` section of the dialog box you can select either a circular (the default) or a rectangular layout for the `Color Alarm`. You can also turn the digital display and the "three dimensional" border on or off.

   - From the `Limits` section of the dialog box you can select a color for each range (low, mid, and high). Click on a color button to show the possible colors. You can also change the text for each range, and you can set the lower and upper limit values to determine where each range begins.

3. Click on `OK` to see the effect of your changes.

Graphically Plotting Data

HP VEE provides 10 display objects that allow you to plot data graphically. These include:

- `XY Trace`—Displays a two-dimensional Cartesian plot of `Y` input values versus either *points* or, for a waveform, *time*.
- `Strip Chart`—Displays continuously-generated data in a "strip-chart" format.
- `Complex Plane`—Displays complex data as a Cartesian plot.
- `X vs Y Plot`—Displays a two-dimensional Cartesian plot of `Y` input values versus `X` input values.
- `Polar Plot`—Displays a polar coordinate graph.
- `Waveform (Time)`—Displays waveform (time-domain) data in a two-dimensional plot.
- `Magnitude Spectrum`—Displays the magnitude of a frequency-domain spectrum.
- `Phase Spectrum`—Displays the phase of a frequency-domain spectrum.
- `Magnitude vs Phase (Polar)`—Displays a polar plot of magnitude against phase for a complex spectrum.
- `Magnitude vs Phase (Smith)`—Displays a Smith plot of magnitude against phase for a complex spectrum.

For a detailed description of each of these objects, refer to the *HP VEE Reference* manual. The rest of this chapter gives some examples showing how to use the graphical display objects.

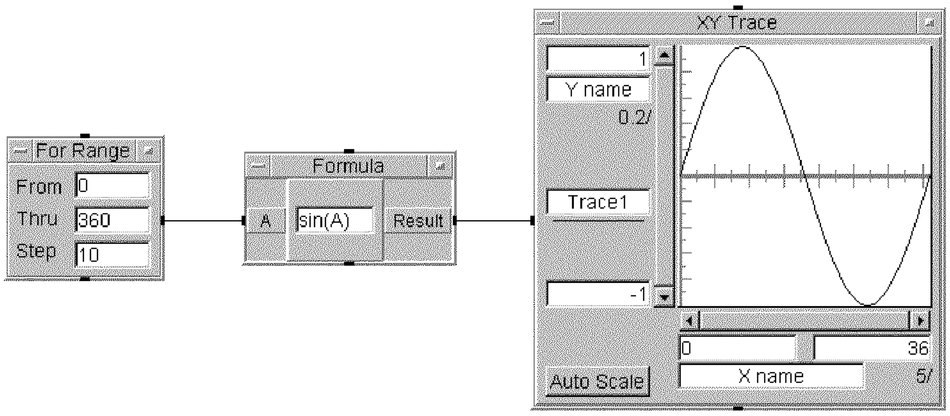
To Plot XY Values

The `XY Trace` and `X vs Y Plot` objects are probably the most generally applicable of the graphical displays.

If you want to plot a series of values, use the `XY Trace` object:

1. Add an `XY Trace` object to the work area (`Display` ⟹ `XY Trace`).
2. Connect a source of data to the data input pin on the `XY Trace` object. The data can either be a continuously-generated series of points, or a waveform.
3. Run the program. If you aren't sure of the appropriate scale, press the `Auto Scale` button to automatically re-scale the display and show the trace.

Here is an example of a program that generates a series of points, and then plots them on the `XY Trace` object.



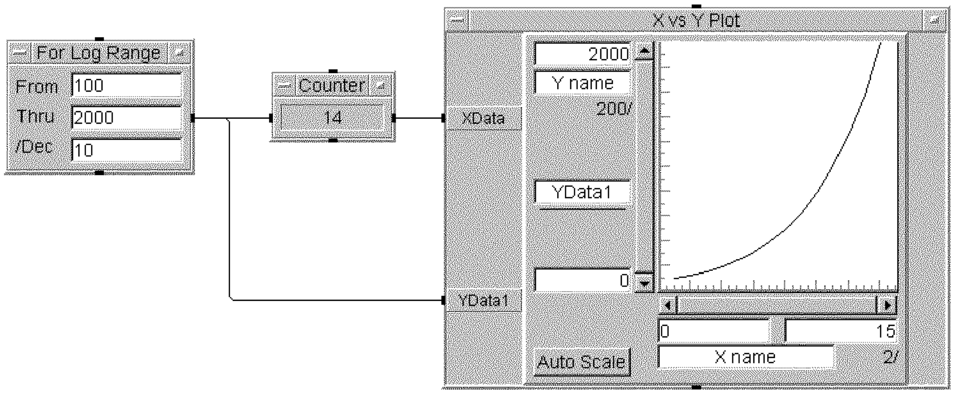
In this program, the `For Range` object repeats 37 times, generating the series of values `0`, `10`, `20`, and so forth, up to `360`. The `Formula` object calculates the sine of each value and outputs the result to the `XY Trace` object, which plots the resulting curve.

If an `XY Trace` object receives *waveform* data on its data input pin, it automatically displays the waveform in the time domain. See "To Display a Waveform" for an example.

If you want to plot a series of `Y` values against `X` values, use the `X vs Y Plot` object:

1. Add an `X vs Y Plot` object to the work area (`Display` ⟹ `X vs Y Plot`).
2. Connect the `XData` and `YData1` data input terminals to the appropriate sources of data. You can plot a continuously-generated series of `Y` points against `X` points, or you can plot a waveform against a waveform. However, you cannot mix data shapes.
3. Run the program. If you aren't sure of the appropriate scale, press the `Auto Scale` button to automatically re-scale the display and show the trace.

Here is an example of a program that plots `Y` points against `X` points:



In this program, the `For Log Range` object repeats 14 times, generating a logarithmic series of `Y` data points. The `Counter` outputs the current count to the `XData` input terminal, while the `For Log Range` object outputs the logarithmic series to the `YData1` input terminal. The `X vs Y Plot` object displays the resulting curve.

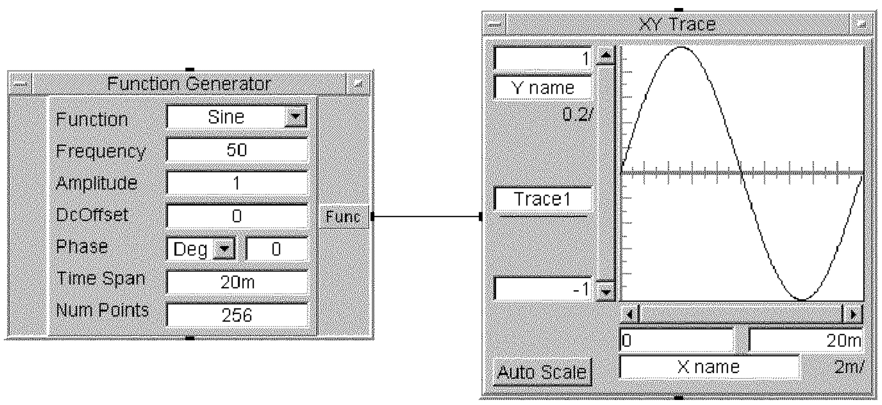
The `X vs Y Plot` object is also capable of displaying waveforms on both axes in the time domain. See "To Display a Waveform" for further information.

To Display a Waveform

To display waveform data you can use the `Waveform (Time)` display object or the `XY Trace` object. To display one waveform against another you can use the `X vs Y Plot` object.

For example, to display a sine wave from a `Function Generator` object on an `XY Trace` object:

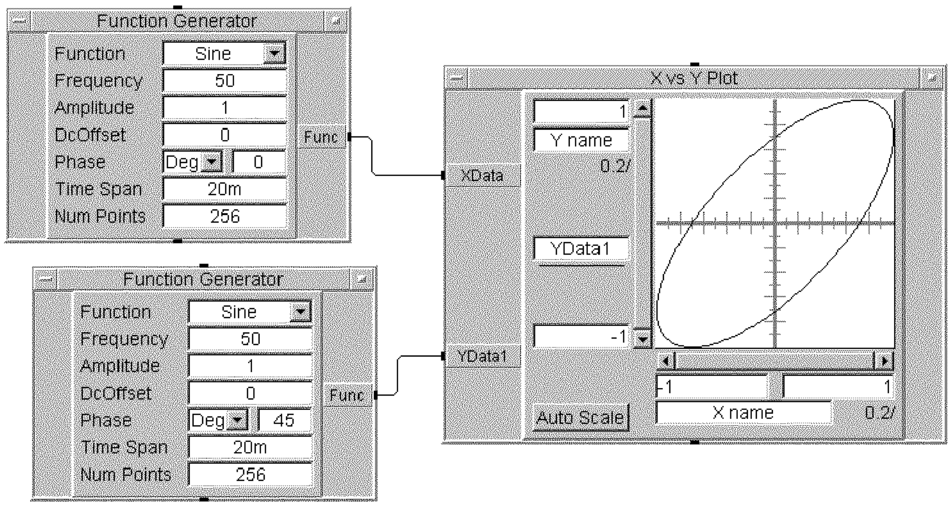
1. Add an `XY Trace` object to the work area (`Display` $\Longrightarrow$ `XY Trace`).
2. Add a `Function Generator` object to work area. Connect the `Func` output of the `Function Generator` to the data input pin on the `XY Trace` object.
3. Change the settings on the `Function Generator` to output a 50 Hz sine wave, as shown below. Run the program and click on `Autoscale` to show the waveform.



Function Sine
Frequency 50
Amplitude 1
DcOffset 0
Phase Deg 0
Time Span 20m
Num Points 256

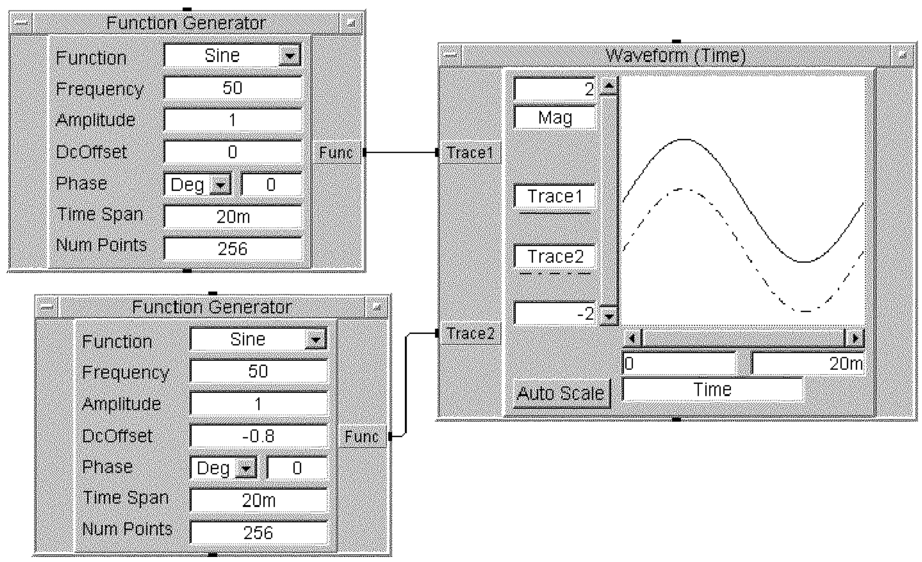
Note that in this program the `XY Trace` object displays the waveform in the time domain. The magnitude of the waveform is plotted against a time scale of from `0` to `20m` (20 milliseconds) to match the time span of the waveform. The `XY Trace` object does this automatically if the input data is a waveform. Otherwise, the `X` axis is simply the number of points plotted, as shown in the previous section.

Here is another example, using the **`X vs Y Plot`** object to display one waveform against another:



In the above program, the **`Function Generator`** objects generate two 50 Hz sine waves, which are 45 degrees out of phase. These waveforms are plotted against each other, displaying a classic Lissajous figure.

Here is a third example, showing two waveforms plotted on separate traces using the **Waveform (Time)** display object.



Function Generator

Function: Sine
Frequency: 50
Amplitude: 1
DcOffset: 0
Phase: Deg 0
Time Span: 20m
Num Points: 256

Function Generator

Function: Sine
Frequency: 50
Amplitude: 1
DcOffset: -0.8
Phase: Deg 0
Time Span: 20m
Num Points: 256

Waveform (Time)

In this example, each **Function Generator** object generates a 50 Hz sine wave, but the second has a d.c. offset of **-0.8**. The **Waveform (Time)** object displays the two waveforms as **Trace 1** and **Trace 2**. (The **Waveform (Time)** object has only one trace by default, but you can add a trace from the object menu. Refer to "To Add a Trace" for further information.)

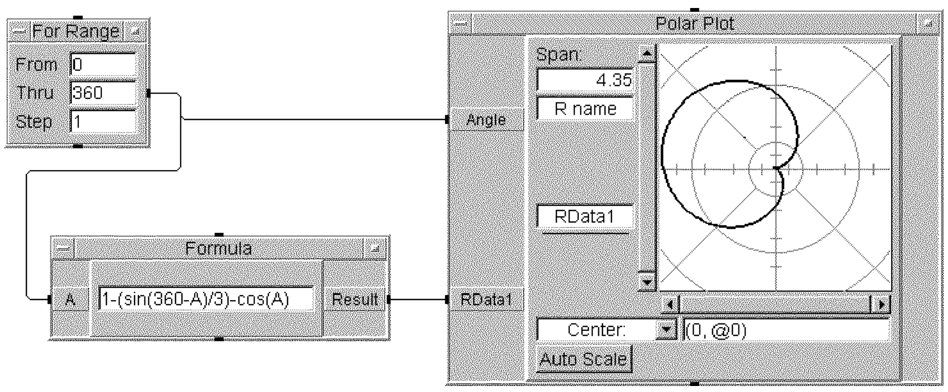
NOTE

You can also use the **XY Trace**, **X vs Y Plot**, and **Waveform (Time)** objects to display waveform data from an instrument. To do this, use an instrument panel, component driver, or **Direct I/O** object (refer to Chapter 5).

To Plot Polar Data

You can plot polar data with the **Polar Plot** display object. Let's create a simple program that plots a trigonometric formula against angular data.

1. Add a **Polar Plot** display object to the work area (**Display** ⟹ **Polar Plot**).
2. Add a **For Range** object and a **Formula** object. Connect the program as shown below. Enter the desired trigonometric formula in the **Formula** object and run the program. Click on **Autoscale** to see the result:

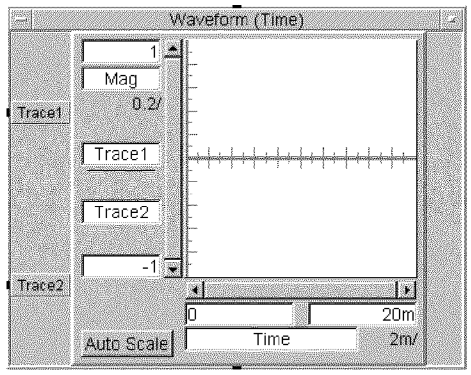


In the example shown above the trigonometric formula is **1-(sin(360-A)/3)-cos(A)**, which creates the "heart-shaped" figure. However, you can enter any formula that you want. The **For Range** object repeats, stepping the angle one degree at a time from **0** to **360** degrees. The **Polar Plot** object plots the trigonometric data against the angle.

To Add a Trace

You can add a trace to any of the graphical display objects. For example, suppose you want to add a trace to a **`Waveform (Time)`** object:

1. Select **`Add Terminal`** $\Longrightarrow$ **`Data Input`** from the object menu of the **`Waveform (Time)`** object.
2. The **`Trace2`** input is added to the object. If you want to see the terminals, select **`Edit Properties`** from the object menu and click on the checkbox for **`Show Terminals`** in the dialog box. Then click on **`OK`**. The object should show two terminals, **`Trace1`** and **`Trace2`**, as shown below:

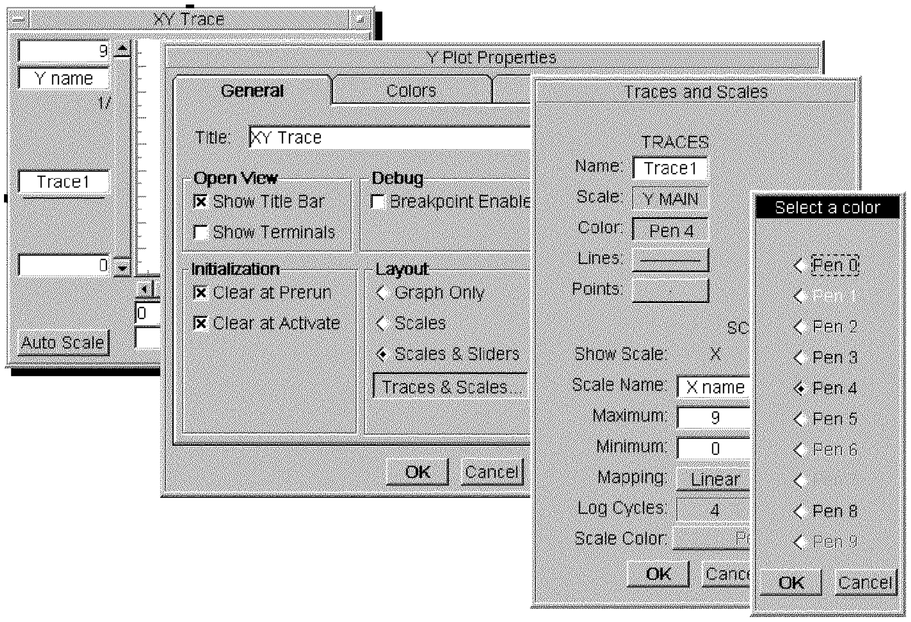


Once you've added the terminals that you want, connect a source of data to each.

To Change the Color of a Trace

The procedure for changing the color of a trace is the same for all of the graphical display objects. For the **XY Trace** object:

1. Select **Edit Properties** from the object menu.
2. Click on the **Traces & Scales** button in the **Y Plot Properties** dialog box.
3. The default pen color is **Pen 4**. Click on the **Pen 4** button to the right of **Color** in the **Traces and Scales** dialog box:



Title: XY Trace
[✓ Show Title Bar]
Show Terminals
Breakpoint Enable
[✓ Clear at Prerun]
[✓ Clear at Activate]
Graph Only
Scales
[✓ Scales & Sliders]

Name: Trace1
Scale: Y MAIN
Color: Pen 4
Lines:
Points:
Show Scale: X
Scale Name: X name
Maximum: 9
Minimum: 0
Mapping: Linear
Log Cycles: 4
Scale Color:

[✓ Pen 4]

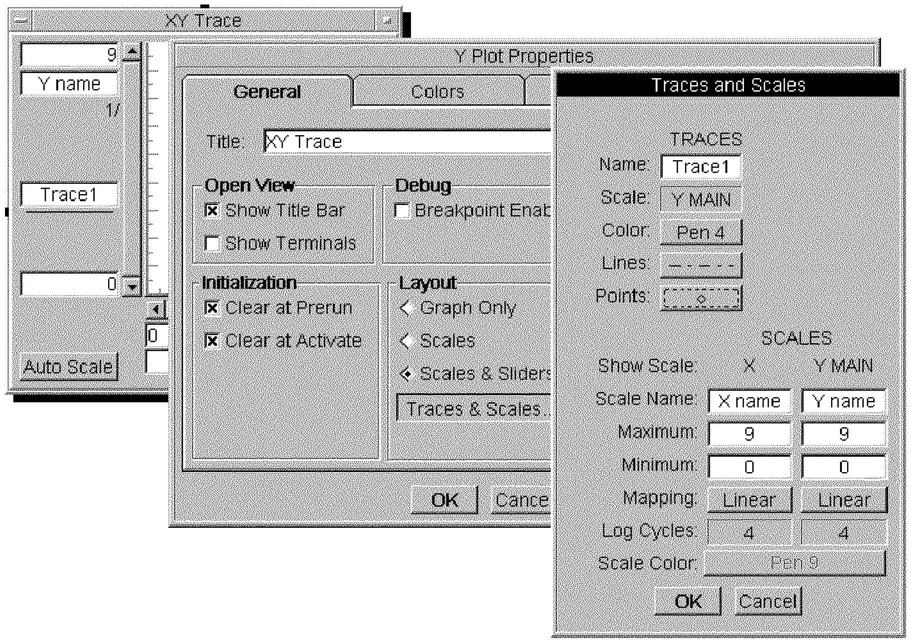
4. The list of pen colors is displayed, as shown. Click on the checkbox for the color you want. Then click on OK in each dialog box to get back to the work area.

The new pen color is now active. Connect a source of data to the display object and run the program. The trace should be the color you selected.

To Change the Line Type or Point Symbol of a Trace

The procedure for changing the line type or point symbol of a trace is the same for all of the graphical display objects. For the **`XY Trace`** object:

1. Select **`Edit Properties`** from the object menu.
2. Click on the **`Traces & Scales`** button in the **`Y Plot Properties`** dialog box.
3. You can change both the line type and the point symbol from the **`Traces and Scales`** dialog box:



Title: XY Trace

Open View: [x] Show Title Bar, [ ] Show Terminals

Debug: [ ] Breakpoint Enab

Initialization: [x] Clear at Prerun, [x] Clear at Activate

Layout: Graph Only, Scales, [✓ Scales & Sliders], Traces & Scales...

Traces and Scales

TRACES

Name: Trace1
Scale: Y MAIN
Color: Pen 4
Lines: – - – - -
Points: ○

SCALES

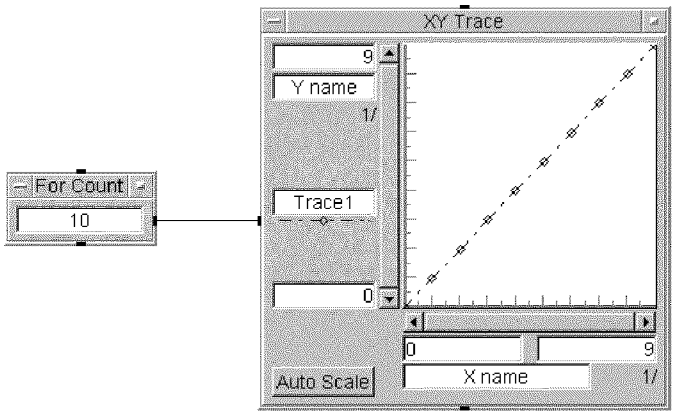
Show Scale:	X	Y MAIN
Scale Name:	X name	Y name
Maximum:	9	9
Minimum:	0	0
Mapping:	Linear	Linear
Log Cycles:	4	4
Scale Color:	Pen 9	

a. To change the line type, click on the button to the right of **`Lines:`** — each time you click, the line type changes. The default is a solid line. You can choose one of several types of dashed lines. When you have cycled through all of the selections, you start over with a solid line.

b. To change the point symbol, click on the button to the right of **`Points:`** — each time you click, the point symbol changes. The default is a simple point (**`.`**). You can choose one of several point symbols, including the diamond (shown), square, **`x`**, **`+`**, and so forth. When you have cycled through all of the selections, you start over with a simple point.

4. Click on **`OK`** in each dialog box to get back to the work area.

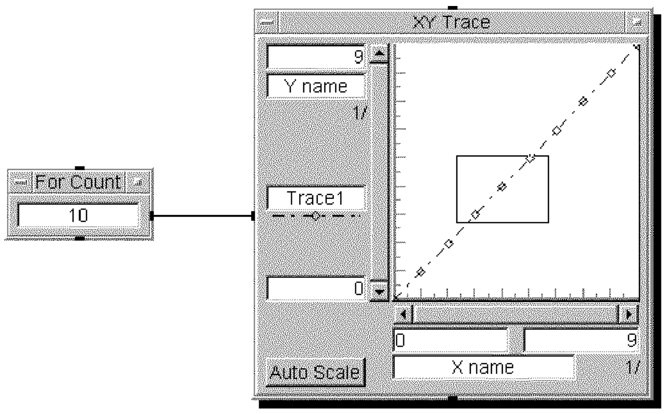
The following is an example showing an **`XY Trace`** object displaying a dashed line with a diamond point symbol:



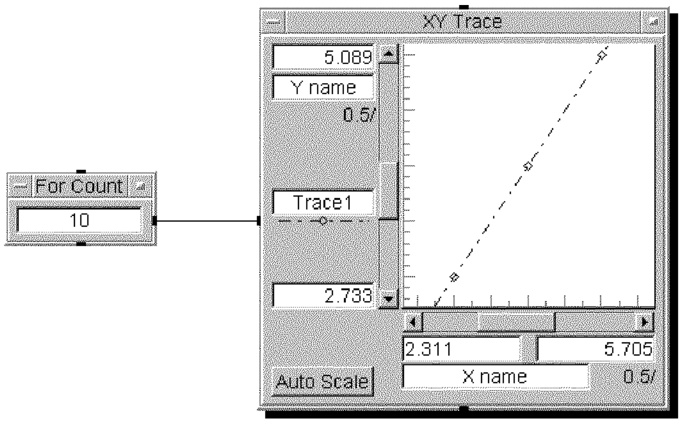
To Zoom into or out of the Display

You can zoom into or out of the display in any of the graphical (XY) display objects. To see how to do this, open the program saved as `manual23.vee` in your manual examples directory, and try these steps:

1. Run the program to display a trace.
2. Select **`Zoom`** $\Longrightarrow$ **`In`** from the object menu of the **`XY Trace`** object.
3. Move the pointer to the upper right-hand corner of the area on which you want to zoom. Then click and drag the pointer to define the area, as shown in the following figure.



4. Release the mouse button. The zoomed area is displayed:

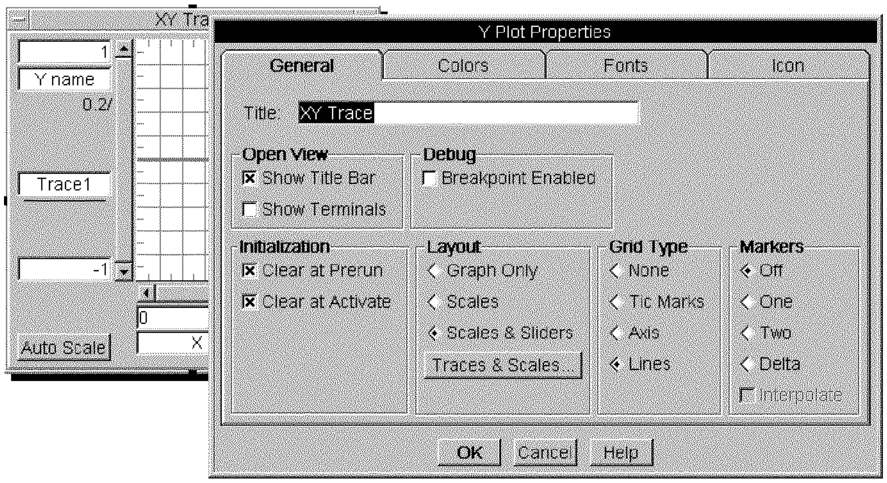


5. Click on `Auto Scale` to return to the original display.
6. Now select `Zoom` $\Longrightarrow$ `Out 2 X` from the object menu of the `XY Trace` object.
7. Note how the display zooms out to twice its original size. Click on `Auto Scale` at any time to return to the original display.

To Change the Grid Type

You can change the grid type of any of the graphical display objects using `Edit Properties` from the object menu. For the `XY Trace` object:

1. Select `Edit Properties` from the object menu. The `Y Plot Properties` dialog box appears:



Y Plot Properties

General | Colors | Fonts | Icon

Title: XY Trace

Open View
- [✓] Show Title Bar
- [ ] Show Terminals

Debug
- [ ] Breakpoint Enabled

Initialization
- [✓] Clear at Prerun
- [✓] Clear at Activate

Layout
- Graph Only
- Scales
- [✓ Scales & Sliders]
- Traces & Scales...

Grid Type
- None
- Tic Marks
- Axis
- [✓ Lines]

Markers
- [✓ Off]
- One
- Two
- Delta
- [ ] Interpolate

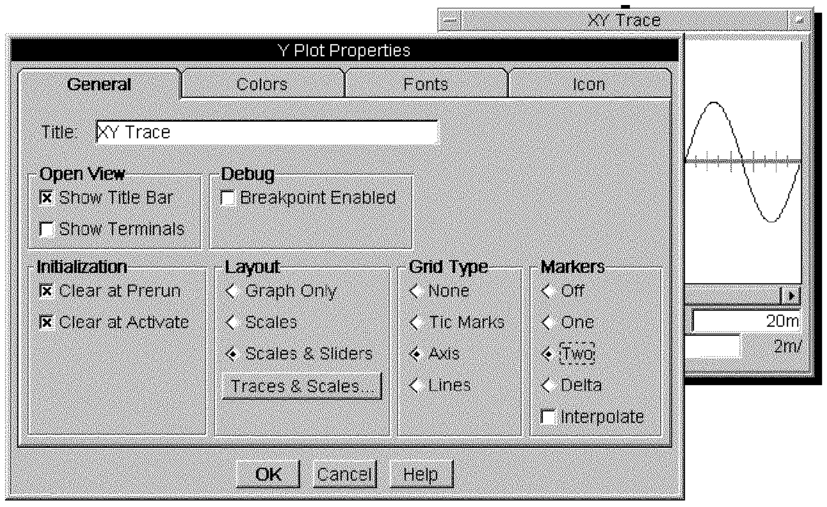
OK | Cancel | Help

2. Under `Grid Type`, click on the checkbox for the type of grid that you want (the default is `Lines`).
3. Click on `OK` to exit the dialog box and see the changes.

To Set Markers in the Display

You can set markers in any of the graphical display objects using `Edit Properties`. For the `XY Trace` object:

1. Select `Edit Properties` from the object menu. The `Y Plot Properties` dialog box appears:



Title: XY Trace

Open View: [✓ Show Title Bar] Show Terminals

Debug: Breakpoint Enabled

Initialization: [✓ Clear at Prerun] [✓ Clear at Activate]

Layout: Graph Only, Scales, [✓ Scales & Sliders]

Grid Type: None, Tic Marks, [✓ Axis], Lines

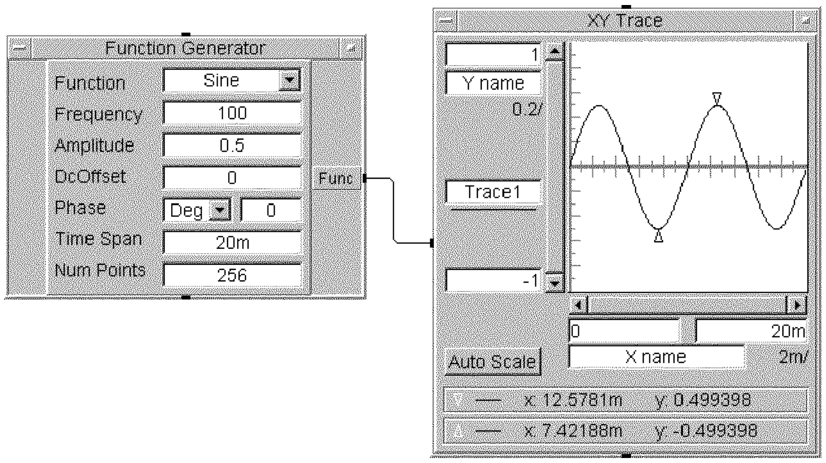
Markers: Off, One, [✓ Two], Delta; Interpolate

2. Under `Markers`, click on the desired checkbox:

- `Off` turns markers off (the default).
- `One` turns on one marker. The marker position is displayed at the bottom of the object.
- `Two` turns on two markers. The position of each is displayed at the bottom of the object.
- `Delta` turns on two markers. The position of each, and the difference between the two positions, is displayed at the bottom of the object.

In addition, if you click on the **Interpolate** checkbox, you can place the markers between the displayed data points. (The marker positions are calculated by linear interpolation.)

3. Click on **OK** to exit the dialog box. The following example shows what an **XY Trace** object looks like with two markers set:



```
x: 12.5781m   y: 0.499398
x: 7.42188m   y: -0.499398
```

Once you have set a marker, you can move it along a trace by clicking and dragging on it with the mouse.

Let's try an example. The program shown above is saved as **manual24.vee** in your manual examples directory.

1. Load and run the program to display a waveform on the **XY Trace** object.
2. Try moving the markers. Note the position data at the bottom of the **XY Trace** object. You can use the markers to identify the positions of peaks and valleys.

In the figure shown above, the markers report the following data:

`X = 12.5781m and Y = 0.499398` (The peak.)

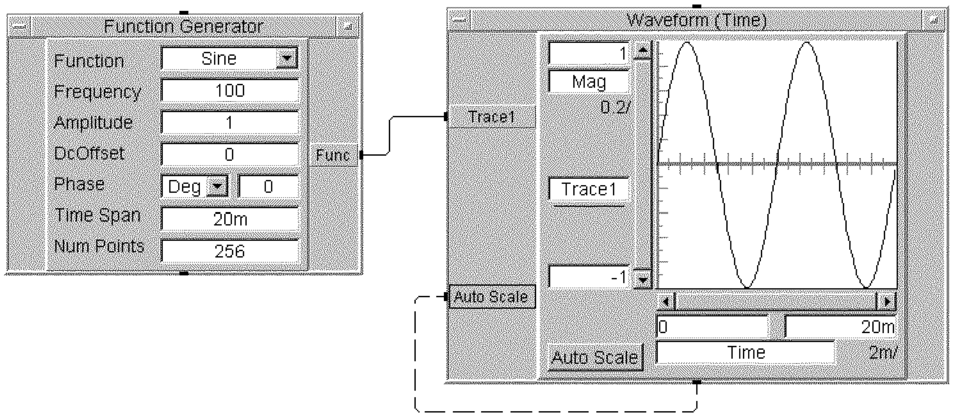
`X = 7.42188m and Y = -0.499398` (The valley.)

To Automatically Re-Scale the Display

You can re-scale any of the graphical displays at any time by clicking on the **Auto Scale** button. However, it is often more convenient to have the display automatically re-scale itself each time it executes. To do this, add a control pin to the display object:

1. Select **Add Terminal** $\Longrightarrow$ **Control Input** from the object menu of the graphical display object.
2. Select the appropriate control input to add:
   - **Auto Scale** re-scales both axes.
   - **Auto Scale X** re-scales the **X** axis.
   - **Auto Scale Y** re-scales the **Y** axis.
3. Connect the sequence output pin of the graphical display object to the control terminal.

Here is an example showing how this works:



Whenever the **Waveform (Time)** object displays a trace, it automatically re-scales the display to show the entire trace.

To Clear the Display

Use either of the following ways to clear the display on any graphical display object:

- Select `Clear Control` $\Longrightarrow$ `Display` from the object menu.

  or

- Add a `Clear` control input terminal to the object. The display is cleared whenever the `Clear` terminal is activated.

7

Printing and Plotting Techniques

Printing and Plotting Techniques

This chapter shows how to use printers and plotting devices with HP VEE. HP VEE supports the HP LaserJet, HP DeskJet, and most PostScript™ printers. These printers can be used for HP VEE printing operations, and for plotting operations as well. In addition, HP VEE supports most HP-GL compatible plotting devices. For further information about supported printers and plotters, refer to the installation guide for your version of HP VEE.

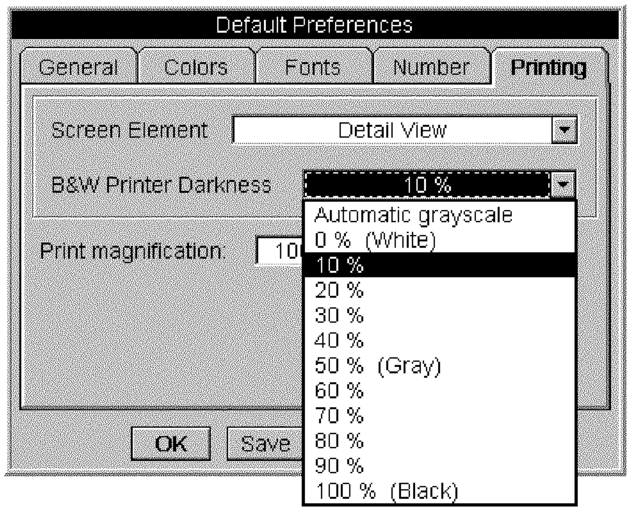
Printing from HP VEE

You can use your system printer to print HP VEE data, to print the HP VEE work area, to print the objects in an HP VEE program, or to print a visual representation of an entire HP VEE program.

To Change the Printer Palette

You can change the gray-scale palette that HP VEE uses for printing by means of the **Default Preferences** dialog box:

1. Select **File** $\Longrightarrow$ **Edit Default Preferences** to display the **Default Preferences** dialog box. Then click on the **Printing** tab.
2. The **Screen Element** and **B&W Printer Darkness** fields both provide drop-down lists. To change the darkness (gray-scale value) for the **Detail View**, click on **B&W Printer Darkness** (or the arrow):



Screen Element: Detail View
B&W Printer Darkness: 10 %
Print magnification: 10

Automatic grayscale
0 % (White)
10 %
20 %
30 %
40 %
50 % (Gray)
60 %
70 %
80 %
90 %
100 % (Black)

3. Now click on the desired darkness. If you want the `Detail View` background to be white, click on `0 % (White)`. (Most HP VEE elements are set to `Automatic grayscale` by default. The colors in your screen palette are automatically converted to appropriate shades of gray.)
4. If you want to change the `Print Magnification` value, click on the field and edit the value. (The default is `100` percent. You can set values greater or less than `100` to change the size of the printed image.)
5. Click on `Save` if you want the changes to be saved as new defaults (in `VEE.RC` or `.veerc`). Or click on `OK` if you want to use the changes for this HP VEE session only (until you select `File` ⟹ `New`, `Open`, or `Exit`).

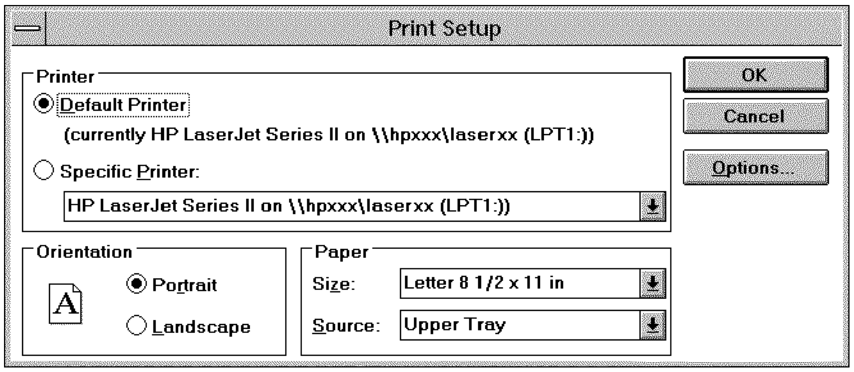
To Set Up a Printer

The procedure for setting up a printer is different for HP VEE for Windows and HP VEE for UNIX.

HP VEE for Windows:

To change the printer configuration for HP VEE for Windows, follow these steps. (In most cases the default printer configuration will work.)

1. Select `File` ⟹ `Print Setup`. The following dialog box appears:



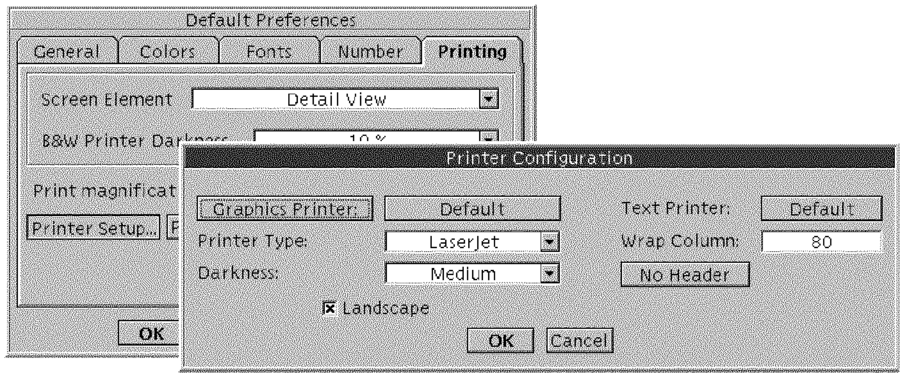
Printer
[✓ Default Printer]
(currently HP LaserJet Series II on \\hpxxx\laserxx (LPT1:))
Specific Printer: HP LaserJet Series II on \\hpxxx\laserxx (LPT1:))
Orientation: [✓ Portrait] Landscape
Paper
Size: Letter 8 1/2 x 11 in
Source: Upper Tray

2. Select the printer configuration that you want, using the standard Microsoft Windows techniques for the `Print Setup` dialog box.
3. Click on `Options` for more printing options, or click on `OK` to confirm your selections.

HP VEE for UNIX:

To change the printer configuration for HP VEE on a UNIX workstation, follow these steps. (In most cases the default printer configuration will work.)

1. Select `File` ⟹ `Edit Default Preferences` to display the `Default Preferences` dialog box. Then click on the `Printing` tab.
2. Click on the `Printer Setup` button to display the `Printer Configuration` dialog box:



Screen Element: Detail View
B&W Printer Darkness: 10 %
Graphics Printer: Default
Text Printer: Default
Printer Type: LaserJet
Wrap Column: 80
Darkness: Medium
☒ Landscape

3. Select the desired printer configuration:
   - Click on the `Default` button to the right of `Graphics Printer` and use the drop-down menu to select a graphics printer. Or click on the `Graphics Printer` button to toggle to `Graphics Directory`.
   - Click on the `Default` button to the right of `Text Printer` and use the drop-down menu to select a text printer.
   - Select other options by editing the appropriate fields.
4. Click on `OK` to confirm your new printer configuration and return to the `Default Preferences` dialog box.

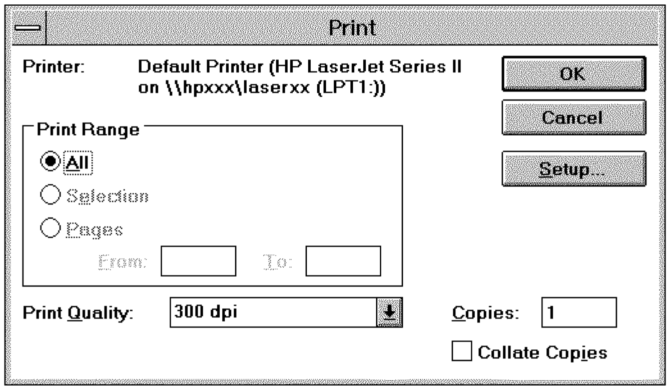
5. Click on `Save` if you want the changes to be saved as new defaults (in the `.veerc` file). Or click on `OK` if you want to use the changes for this HP VEE session only (until you select `File` ⟹ `New`, `Open`, or `Exit`).

To Print the HP VEE Work Area

You can print the HP VEE work area (actually, the HP VEE Window) using the `Print Screen` feature.

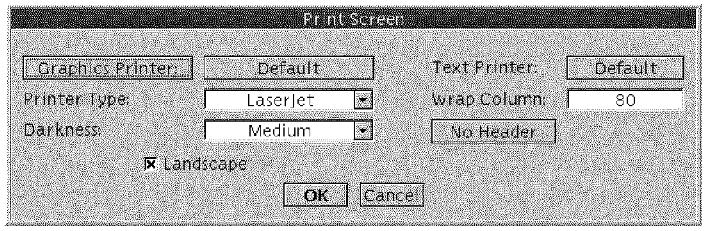
1. Select `File` ⟹ `Print Screen`. The print dialog box appears (the dialog box is different for Windows and UNIX):

HP VEE for Windows Dialog Box



Printer: Default Printer (HP LaserJet Series II on \\hpxxx\laserxx (LPT1:))
Print Range: All
Print Quality: 300 dpi
Copies: 1

HP VEE for UNIX Dialog Box



2. If the printing configuration is acceptable, click on **OK** to print the HP VEE work area. Or change the parameters and then click on **OK**. The work area is printed, complete with the window border.

Shortcuts

If you are using HP VEE on a UNIX workstation, you can print the work area by pressing Shift+Print at any time. (If a program is running, it will pause several seconds for the printout.)

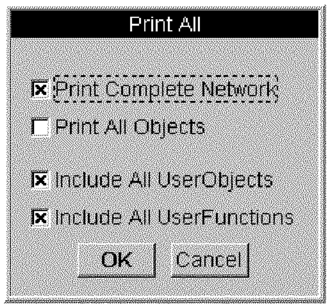
If you are using HP VEE for Windows, you can capture the HP VEE window in the Microsoft Windows clipboard by pressing Alt+Print Screen at any time. (If a program is running, it will pause momentarily.) You can then paste the image into Microsoft Paintbrush or other Windows graphics applications.

These shortcuts allow you to capture the HP VEE screen even when a dialog box is displayed and the menu is inaccessible.

To Print an Entire HP VEE Program

You can print a representation of an entire HP VEE program using the `Print All` feature.

1. Select `File` ⟹ `Print All`. The `Print All` dialog box appears:



Print All

[✓] Print Complete Network
[ ] Print All Objects
[✓] Include All UserObjects
[✓] Include All UserFunctions

2. Now click on `OK`. The print dialog box appears. Change the printing configuration if desired.
3. Click on `OK` again to print a representation of the entire program.

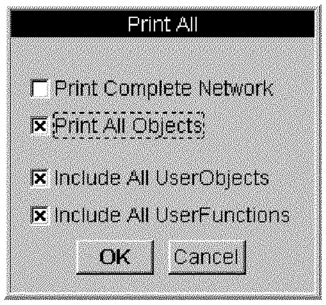
By default, the `Print All` dialog box has `Print Complete Network` turned on, which causes the entire program to be printed (even if it doesn't fit in the visible screen). Multiple pages are printed if necessary to show the entire program, with objects connected together with lines to show the complete network.

With `Include All UserObjects` and `Include All UserFunctions` turned on, the objects inside of UserObjects and UserFunctions are included in the printout.

To Print the Objects in an HP VEE Program

You can print all of the objects in an HP VEE program using the **`Print All`** feature.

1. Select **`File`** $\Longrightarrow$ **`Print All`**. The **`Print All`** dialog box appears.
2. By clicking on the checkboxes, turn **`Print All Objects`** on, and turn **`Print Complete Network`** off, as shown below:



Print All

☐ Print Complete Network
☒ Print All Objects

☒ Include All UserObjects
☒ Include All UserFunctions

3. Now click on **`OK`**. The print dialog box appears. Change the printing configuration if desired.
4. Click on **`OK`** again to print all of the objects in the program.

With **`Print All Objects`** turned on, **`Print All`** causes all of the objects in the program to be printed out sequentially, not showing the connections in the program. Multiple pages are printed if necessary.

With **`Include All UserObjects`** and **`Include All UserFunctions`** turned on, the objects inside of UserObjects and UserFunctions are included in the printout.

To Send Data to a Printer

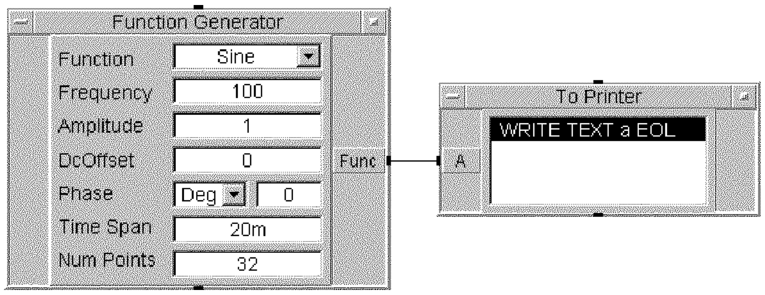
You can send data from your HP VEE program directly to a printer by using the `To Printer` object.

1. Add a `To Printer` object to your program (`I/O` $\Longrightarrow$ `To` $\Longrightarrow$ `Printer`) and connect the data input pin to the source of the data that you want to print.
2. Double-click on the `To Printer` icon to display the open view. Note that the `To Printer` object uses I/O transactions to output data, just like the `To File` object described in Chapter 4.
3. The default output transaction is `WRITE TEXT a EOL`. If desired, change the transaction to output the data in some other form. (Refer to "To Edit a File I/O Transaction" in Chapter 4. The procedure is the same.)
4. Run the program. When the `To Printer` object operates, the data is output to the printer, which prints the data using the current printing configuration.

Tip for Windows Users

On UNIX workstations, the `To Printer` object outputs straight ASCII text to the printer, including carriage-return/line-feed characters, tab characters, and escape characters. However, if you are using HP VEE for Windows, the `To Printer` object outputs data through the Windows printer driver, which adds a header and changes the fonts. If you want to output straight ASCII text to a printer from HP VEE for Windows, use the `To File` object with the file name set to `LPT1` (or whichever printer port you are using). Refer to Chapter 4 for further information about using `To File`.

Let's look at an example:



Function: Sine
Frequency: 100
Amplitude: 1
DcOffset: 0
Phase: Deg 0
Time Span: 20m
Num Points: 32

WRITE TEXT a EOL

In this example the `Function Generator` is configured to output a 100 Hz sine wave with only 32 points. The waveform is output to the `To Printer` object (shown in its open view), which outputs the data to the system printer using the `WRITE TEXT a EOL` transaction. The resulting printout is a list of the 32 points:

```
0
0.3826834323650898
0.7071067811865475
0.9238795325112867
1
0.9238795325112867
0.7071067811865477
0.3826834323650894
1.224646752988016E-16
-0.3826834323650897
-0.7071067811865477
-0.9238795325112868
-1
-0.9238795325112866
-0.707106781186547
-0.3826834323650896
-2.449293505976033E-16
0.3826834323650899
0.7071067811865474
0.9238795325112868
1
0.9238795325112867
0.7071067811865471
0.3826834323650889
3.673940362910631E-16
-0.3826834323650898
-0.7071067811865479
-0.9238795325112871
-1
-0.9238795325112867
-0.7071067811865472
-0.3826834323650889
```

Plotting XY Displays on a Printer or Plotter

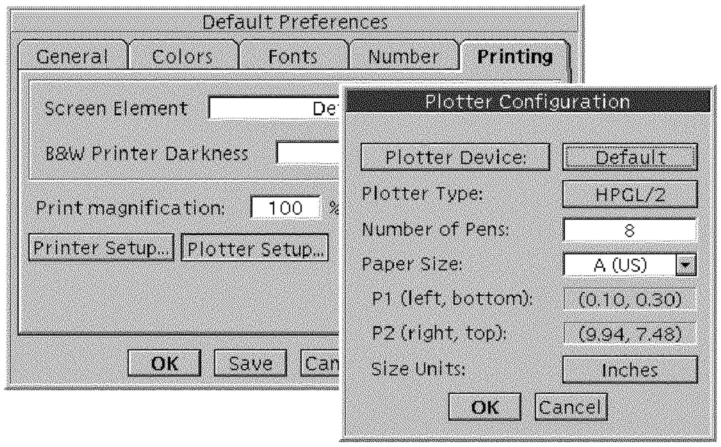
You can output a plot from an XY display object (for example, **XY Trace**) to an external plotting device, which may be a printer or an HP-GL plotter.

To Set Up a Plotting Device

If you are using HP VEE for Windows, the **Plot** function uses the standard Windows printer drivers for plotting. So you don't need to do anything special to set up a plotting device.

If you are using HP VEE on a UNIX workstation, you can set up a plotting device as follows:

1. Select **File** ⟹ **Edit Default Preferences** to display the **Default Preferences** dialog box. Then click on the **Printing** tab.
2. Click on the **Plotter Setup** button to display the **Plotter Configuration** dialog box:



Screen Element: De
B&W Printer Darkness:
Print magnification: 100 %

Plotter Device: Default
Plotter Type: HPGL/2
Number of Pens: 8
Paper Size: A (US)
P1 (left, bottom): (0.10, 0.30)
P2 (right, top): (9.94, 7.48)
Size Units: Inches

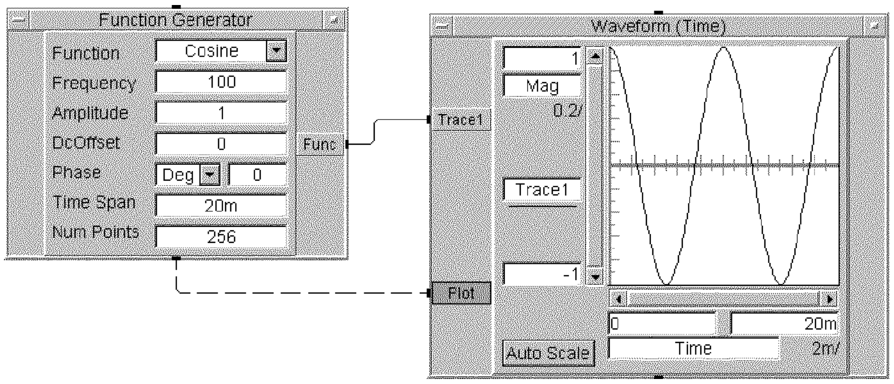
3. Select the desired plotter configuration:
   - Click on the `Default` button to the right of `Plotter Device` and use the drop-down menu to select a device. Or click on the `Plotter Device` button to toggle to `Plot to File`.
   - Select a `Plotter Type`, `Number of Pens`, `Paper Size`, and so forth by editing the appropriate fields.
4. Click on `OK` to confirm your new plotter configuration and return to the `Default Preferences` dialog box.
5. Click on `Save` if you want the changes to be saved as new defaults (in the `.veerc` file). Or click on `OK` if you want to use the changes for this HP VEE session only (until you select `File` ⟹ `New`, `Open`, or `Exit`).

To Plot a Graph

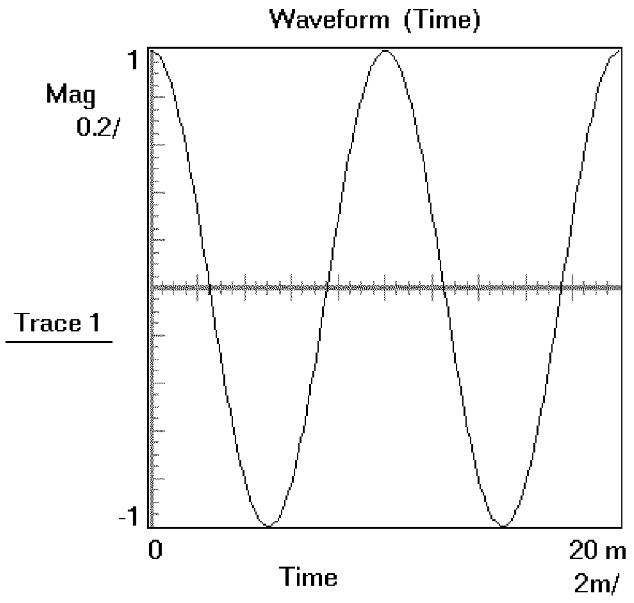
There are two ways that you can plot a graph from an XY display (for example, an `XY Plot` or `Waveform (Time)` display) to your plotting device.

- Select the `Plot` command from the object menu of the XY display.
- Add a `Plot` control input terminal to the XY display. (Object Menu ⟹ `Add Terminal` ⟹ `Control Input` ⟹ `Plot`.) Use the `Plot` terminal to programmatically plot the graph.

Let's look at an example of the latter technique. In the following program, the `Function Generator` object outputs a 100 Hz cosine wave to the `Waveform (Time)` display object.



The **`Waveform (Time)`** object has a **`Plot`** control input terminal, which is connected to the sequence output pin of the **`Function Generator`**. Thus, when the **`Function Generator`** finishes operating, it activates the **`Plot`** terminal, and the graph is output to the plotting device. The resulting graph is shown below:



8

Creating Custom Objects and Functions

Creating Custom Objects and Functions

This chapter shows how to create your own custom objects and functions. You can encapsulate a group of HP VEE objects, which together perform a particular task, into a `UserObject`. You can also create a custom function by converting a `UserObject` into a UserFunction. Let's begin by looking at `UserObject`s.

Creating and Using UserObjects

In this section we will look at some examples showing how to create and use `UserObjects`. You can use `UserObjects` to add structure to your program. Let's begin by looking at some `UserObject` concepts.

Some UserObject Concepts

A `UserObject` provides the means for you to encapsulate a group of objects that perform a particular task into a single, custom object. This encapsulation allows you to:

- Use modular design techniques in building your HP VEE program. This allows you to solve a complex problem through an organized approach.

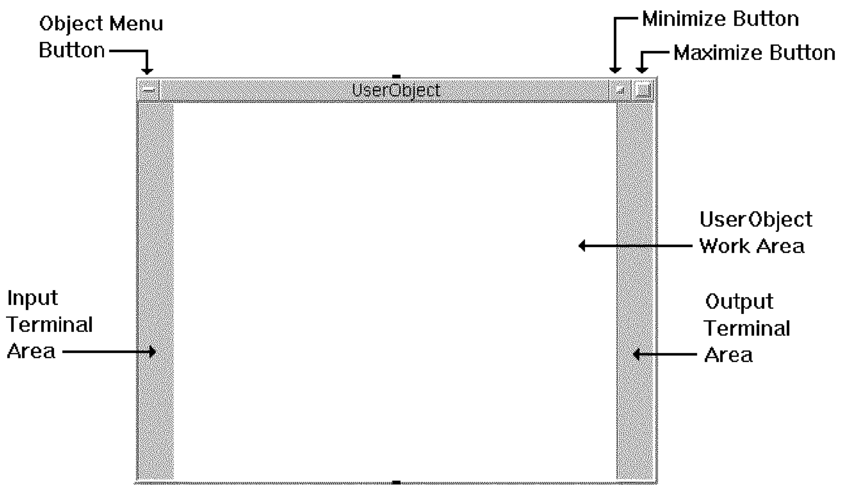
  `UserObjects` *allow you to use top-down design techniques to create a more flexible and maintainable program.*

- Build user-defined objects that you can save in a library for later re-use.

  Once a `UserObject` *is created and saved, you can* `Merge` *it in other programs.*

UserObject Features

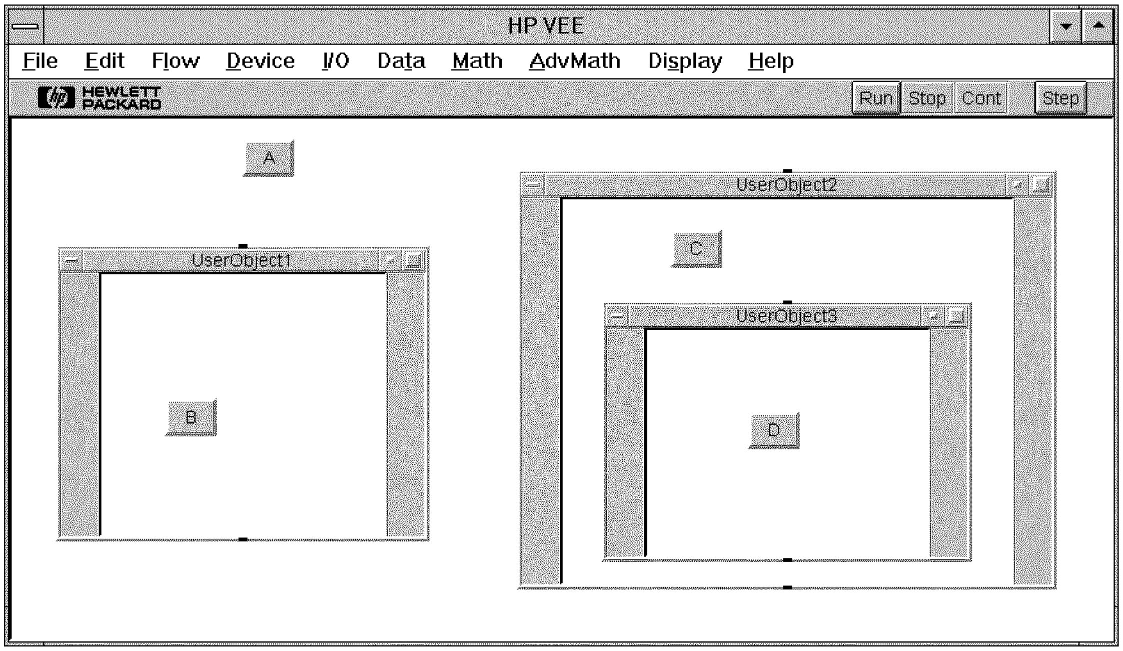
The following figure shows an "empty" `UserObject` and identifies its features:



- Like all other HP VEE objects, each `UserObject` has an **object menu button**, which raises the object menu, and a **minimize button**, which minimizes the `UserObject`. In addition, the `UserObject` has a **maximize button**, which maximizes the `UserObject` to occupy the available space.
- The `UserObject` **work area** is similar to the main HP VEE work area. You can create an HP VEE program segment within the `UserObject` by adding objects to the `UserObject` work area and connecting them.
- The `UserObject` **input terminal area** and **output terminal area** allow you to add data terminals and control terminals, so that the `UserObject` can communicate with the rest of your HP VEE program.

Contexts and UserObjects

A `UserObject` represents a separate **context** within an HP VEE program, just as a subprogram represents a separate context within a C or BASIC program. You can nest `UserObjects` in an HP VEE program, as shown below:



Each `UserObject` has a different context depending on its location in the program. There are four contexts in the above example:

1. Object `A`, `UserObject1`, and `UserObject2` are in the **root context** of the program. That is, they are in the main work area.
2. Object `B` is in the context of `UserObject1`. That is, it is in the work area of `UserObject1`.
3. Object `C` and `UserObject3` are in the context of `UserObject2`.
4. Object `D` is in the context of `UserObject3`.

Propagation and UserObjects

Propagation in a program containing `UserObject`s is affected by the fact that a `UserObject` is a separate context. The `UserObject` propagation rules are as follows:

- All data input terminals (and the sequence input terminal if connected) of the `UserObject` must be activated before any objects within the `UserObject` operate.
- Once the data input terminals (and the sequence input terminal if connected) of the `UserObject` have been activated, the `UserObject` operates. The objects within the `UserObject` operate following the rules of propagation listed under "Propagation Summary" in Chapter 1. However, the objects within the `UserObject` time-share in operation with external objects on different subthreads. The `UserObject` does *not* block the operation of external objects on different subthreads.
- If the optional `XEQ` input terminal is activated, the `UserObject` immediately begins operation of the objects within its work area, using whatever "old" data may be on the unactivated input terminals of the `UserObject`. In most cases, you need not use the `XEQ` terminal for a `UserObject`.
- The `UserObject` data output terminals are not activated until all objects within the `UserObject` finish operating (unless the `UserObject` is exited prematurely by an error or an `Exit UserObject`). Only those output terminals activated from inside the `UserObject` pass data to objects outside the `UserObject`. When activated, each data output terminal outputs only one data container.

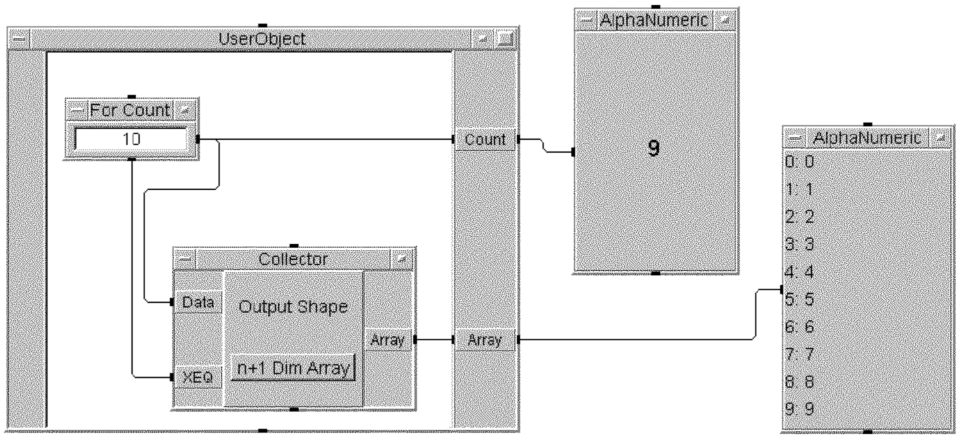
For more information about the propagation rules, refer to "Propagation Summary" in Chapter 1.

NOTE

If there is a `Start` object in a `UserObject` (to handle feedback), pressing `Start` causes only the objects within the `UserObject` to run. No data will be read from the input terminals of the `UserObject`, nor will its output terminals be activated. Therefore, no propagation outside the `UserObject` takes place.

Data Output from a UserObject

As stated in the propagation rules, each data output terminal of a `UserObject` outputs only one data container (the last received by the terminal) to the context outside the `UserObject` when the objects within the `UserObject` finish propagating. This can lead to unexpected results in your program if you neglect to account for it. The following example illustrates this situation:



Although the `For Count` object sends 10 data containers (the numbers 0 through 9) to the `Count` output terminal, only one data container (the last number) is output from the `UserObject`. However, you can use a `Collector` object to collect the data from the `For Count` object into an array. The `Array` output terminal also outputs only one data container, but that container is a one-dimensional *array* of 10 values (0 through 9).

To Add Objects to the UserObject Work Area

Any object that you place inside the `UserObject` work area becomes part of the `UserObject`. Use any of the following methods:

- Select the appropriate object from the main menu and place the object in the `UserObject` work area.
- Move an object from the main HP VEE work area to the `UserObject` work area.
- Copy (use **`Edit`** $\Longrightarrow$ **`Copy`**) or clone (use **`Edit`** $\Longrightarrow$ **`Clone`**) an object from the main work area and place the copy in the `UserObject` work area.

To Add Input and Output Terminals to a UserObject

In general, you can add input and output terminals to a `UserObject` using the same techniques that you would for any other object.

- Select **`Add Terminal`** from the object menu, and then select the type of terminal you want to add: **`Data Input`**, **`Control Input`**, **`XEQ Input`**, **`Data Output`**, or **`Error Output`**.
- Place the mouse cursor in the data input terminal area and press Ctrl+A. A data input terminal is added to the `UserObject` each time you press Ctrl+A.
- Place the mouse cursor in the data output terminal area and press Ctrl+A. A data output terminal is added to the `UserObject` each time you press Ctrl+A.

Shortcut

There is a shortcut for adding input and output terminals to a `UserObject`:

- Connect an output terminal of an object within the `UserObject` to an input terminal of an external object. A data output terminal automatically appears on the `UserObject`.
- Connect an output terminal of an external object to an input terminal of an object within the `UserObject`. A data input terminal automatically appears on the `UserObject`.

The appropriate connections through the new `UserObject` terminals are automatically made. Also, the terminals are automatically given appropriate names.

In addition to data terminals, you can connect to other types of terminals on objects inside a `UserObject` (sequence, control, `XEQ`, and `Error` terminals, for example). However, the corresponding terminals created for the `UserObject` will be data terminals. For example, if you connect the `Error` output terminal of an object within a `UserObject` to an external object, a data output terminal named `Error` appears on the `UserObject`.

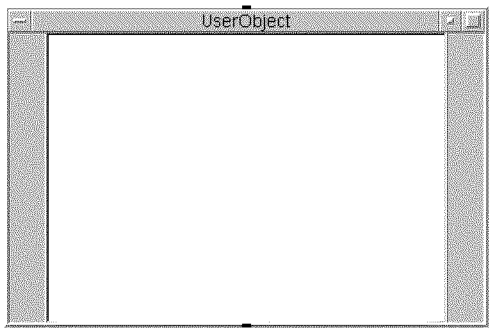
You can connect an object within one `UserObject` to an object that is in another `UserObject`.

To Create a UserObject in a Program

The following example shows, step-by-step, how to create a simple `UserObject` and connect it in an HP VEE program.

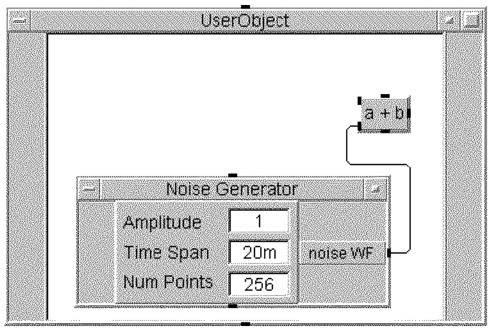
1. Add a UserObject to the main HP VEE work area:

To add a `UserObject` to the main HP VEE work area, select `Device` ⟹ `UserObject` from the menu, and place the `UserObject` in the work area. An "empty" `UserObject` appears:



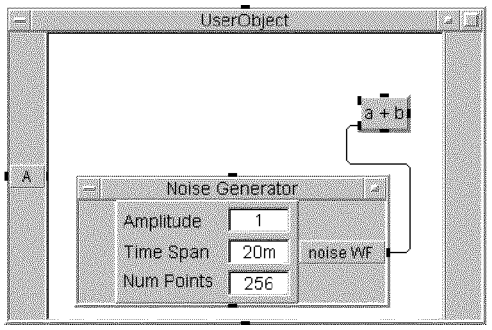
2. Add objects to the UserObject work area:

For our example, select the `+` object from the `Math` menu and the `Noise Generator` object from the `Device` menu, and place each of them in the `UserObject` work area. Then connect them as follows:

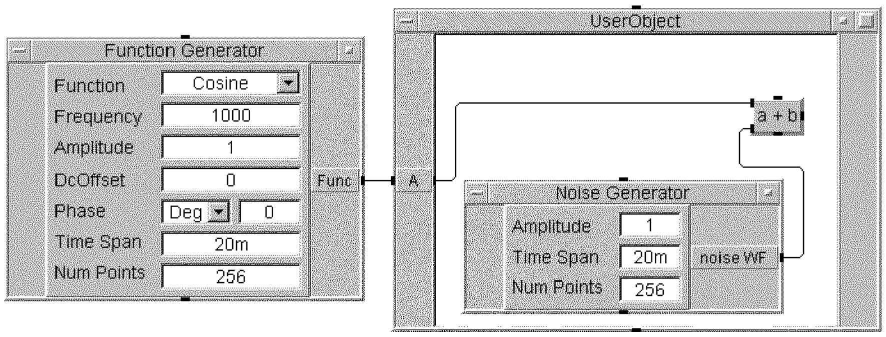


3. Add input and output terminals to the UserObject:

Place the mouse cursor in the input terminal area of the `UserObject` and press (Ctrl)+(A). The data input terminal `A` appears:



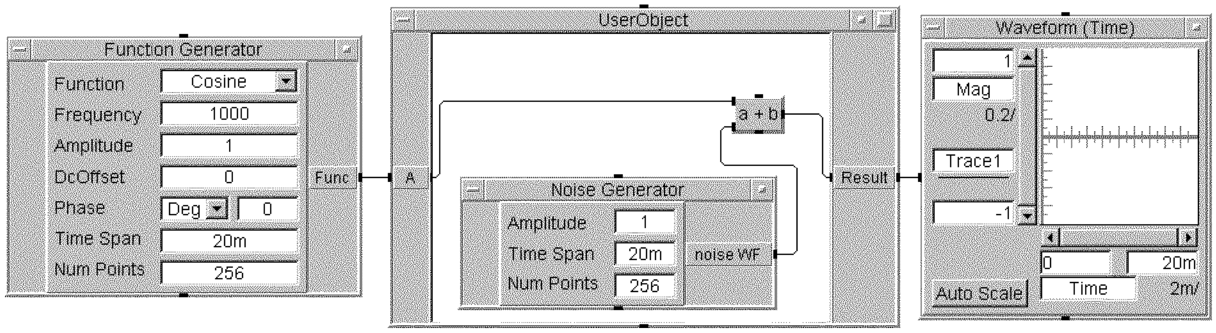
Now you can use the data input terminal **A** to connect an external object to the **UserObject**. In this case, add a **Function Generator** object (**Device** ⟹ **Virtual Source** ⟹ **Function Generator**) to the main work area and connect terminal **A** as shown in the following figure:



Function Generator
Function: Cosine
Frequency: 1000
Amplitude: 1
DcOffset: 0
Phase: Deg 0
Time Span: 20m
Num Points: 256

Noise Generator
Amplitude: 1
Time Span: 20m
Num Points: 256

There is a shortcut for adding data input terminals and data output terminals to a `UserObject`. You can connect objects within a `UserObject` to objects outside the `UserObject` directly. For example, add a **Waveform** display to our main program. Then connect the data output terminal of the **+** object inside the `UserObject` to the data input terminal of the **Waveform** object. The **Result** data output terminal automatically appears on the `UserObject`:

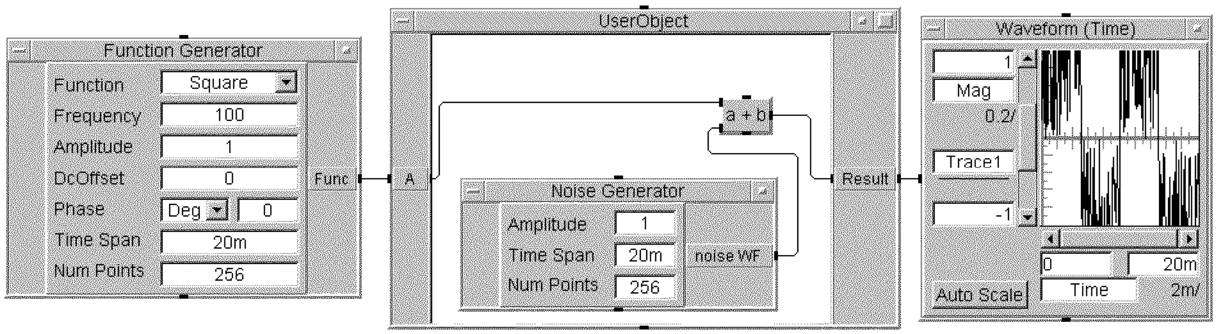


Function Generator
Function: Cosine
Frequency: 1000
Amplitude: 1
DcOffset: 0
Phase: Deg 0
Time Span: 20m
Num Points: 256

Noise Generator
Amplitude: 1
Time Span: 20m
Num Points: 256

4. Test the UserObject in the program:

Let's see what the program does. First, change the **`Function`** setting on the **`Function Generator`** object to **`Square`**. Also, change the **`Frequency`** setting to **`100`**. Then run the program. You should see something like the following:



Function Generator

Function: Square
Frequency: 100
Amplitude: 1
DcOffset: 0
Phase: Deg 0
Time Span: 20m
Num Points: 256

Noise Generator

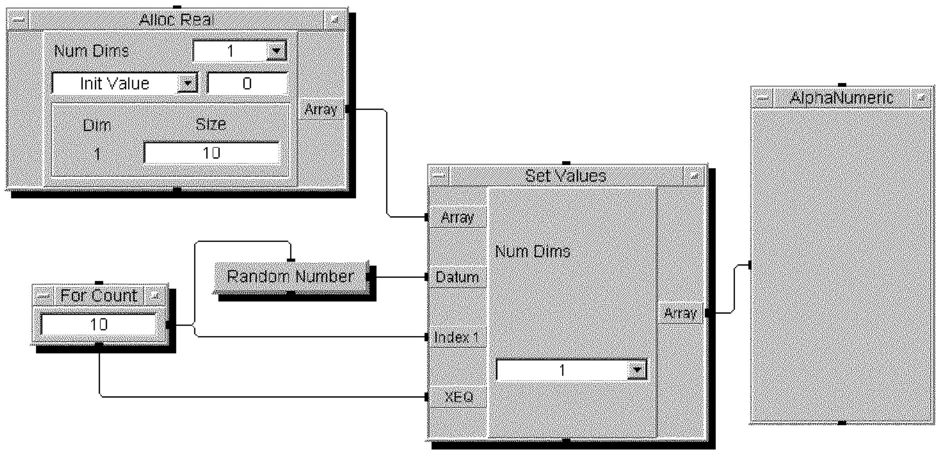
Amplitude: 1
Time Span: 20m
Num Points: 256

The **`UserObject`** adds a noise component to the waveform on input **`A`**. You can save screen space in your program by minimizing a **`UserObject`**. However, change the **`UserObject`** title to something descriptive first, so you can identify the **`UserObject`** by its function in the program.

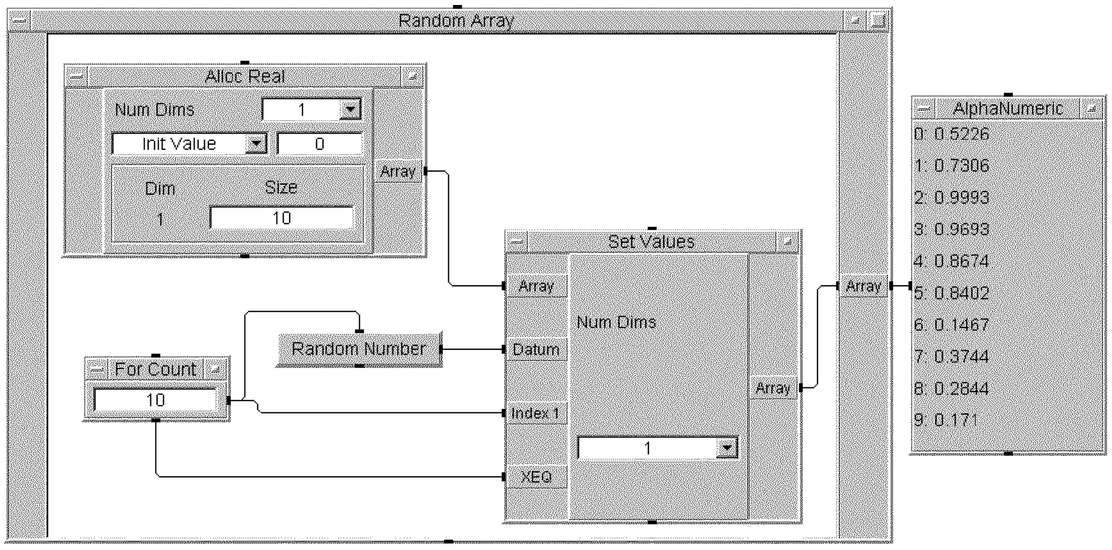
To Create a UserObject from Objects in a Program

Once you have written an HP VEE program you may decide that you want to make part of the program into a **`UserObject`**. In Chapter 3 we showed how to create a program to generate a one-dimensional array of random numbers. Let's look at how to convert several objects in this program into a **`UserObject`**.

1. Open the program (`manual06.vee` in your manual examples directory).
2. Click on `Edit` ⟹ `Select Objects`. The message `Select Desired Objects` is displayed on the tool bar.
3. Click on each of the following objects to select them: `Alloc Real`, `For Count`, `Random Number`, and `Set Values`. The selected objects are highlighted with shadows, as shown below:



4. When you are done selecting objects, click in an empty space in the work area to turn off `Select Desired Objects` mode, and then select `Edit` ⟹ `Create UserObject`.
5. Change the title of the `UserObject` to `Random Array` and run the program. You should see something like this:

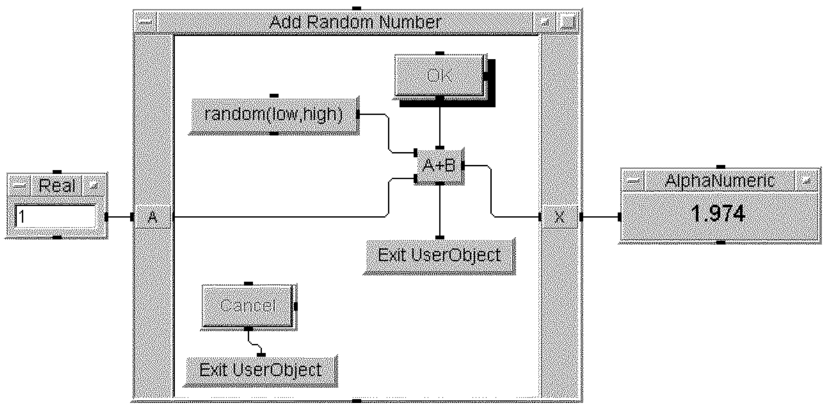


Note that when the `UserObject` is created, the connection to the `AlphaNumeric` object is retained and an `Array` output terminal is automatically created. (This program is saved as `manual25.vee` in your manual examples directory.)

To Exit a UserObject Early

There are two objects that force propagation to exit a `UserObject` before all of the objects within the `UserObject` have operated. These are `Exit UserObject` and `Raise Error`. The `Exit UserObject` object, when activated, causes an immediate exit from the `UserObject` context to the context outside it (for example, the main program). The `Raise Error` object generates a user-defined error message box, and also causes propagation to exit the `UserObject`. (No data is output from the `UserObject` unless there is an `Error` output pin.)

Let's look at an example using **Exit UserObject**:



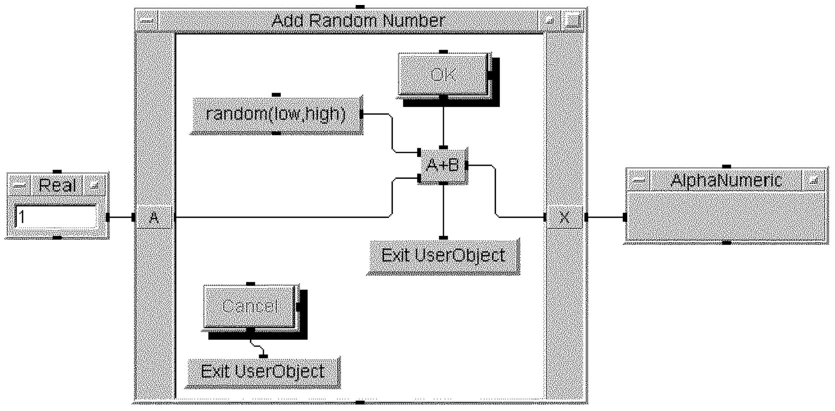
To see how **Exit UserObject** works, open and run this program (saved as **manual26.vee** in your manual examples directory). Turn on animation by selecting **Edit** ⟹ **Animate** to see the order of propagation. The **OK** and **Cancel** buttons both become active when the program is run, pausing execution in the **UserObject**. (The **Cancel** button is just an **OK** button with the title changed.)

- If you press the **OK** button, the upper thread in the **UserObject** finishes propagating. The **A+B** object adds its two inputs, the result is propagated to output terminal **X**, and then the **Exit UserObject** is activated. The result on terminal **X** is output to the **AlphaNumeric** display as the **UserObject** exits.
- If you press **Cancel**, the **Exit UserObject** in the lower thread operates. This causes an exit from the **UserObject** without finishing propagation of the upper thread. No data is output.

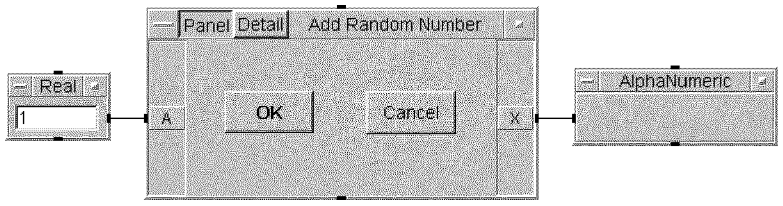
To Add a Panel View to a UserObject

A **panel view** is a view of an HP VEE program or `UserObject` that shows only those objects needed by an operator to run the program and view the resulting data. Refer to *Getting Started with HP VEE* for an example of adding a panel view to an HP VEE program. The following example shows how to add a panel view to a `UserObject`:

1. Open the program `manual26.vee` in your manual examples directory.
2. Raise the `Edit` menu for the `UserObject` (click the right mouse button within the `UserObject` work area) and click on `Select Objects`.
3. Click on the `OK` and `Cancel` buttons to select them, highlighting them as shown below:



4. Click on any empty area to turn off `Select Desired Objects` mode, and then select `Add To Panel` from the `Edit` menu of the `UserObject`.
5. The panel view appears, showing just the `OK` and `Cancel` buttons. You can move the buttons and re-size the `UserObject` to obtain a panel view similar to the one that follows.



In the panel view, only the objects that you added to the panel show. However, the `UserObject` functions just as before. Press `Detail` if you want to see the detail view, which is unchanged. You can change the colors and fonts, select a background bitmap, and otherwise enhance the panel view without affecting the detail view. For further information about using panel views to create an operator interface, refer to your *Building an Operator Interface with HP VEE* manual.

To Create a Library of UserObjects

You can save one or more `UserObjects` in a file by using `Save Objects`:

1. Create the `UserObject`(s) in your work area.
2. Select the `UserObject`(s).
3. Select `File` ⟹ `Save Objects`. You'll be prompted for a file name in which to save the `UserObject`(s).

You can merge the `UserObject`(s) from your library into the work area by using `Merge`:

1. Select `File` ⟹ `Merge`. You'll be prompted for a file name.
2. Select the file name for your library. The `UserObject`(s) in the file are merged into the work area. Place them where you want them in your program.

To Secure a UserObject

After the design of a `UserObject` is final, you may want to secure it. This prevents others from viewing the contents or changing the `UserObject`.

Caution

Don't secure your only copy of a `UserObject`. *Once you secure a UserObject, it cannot be unsecured!*

To secure a `UserObject`:

1. Select `Secure` from the object menu of the `UserObject`.
2. You will be prompted for a file name in which to save an unsecured version of the `UserObject`. Enter a descriptive file name.
3. Click on `Save, then secure`. (Don't select `Secure without saving` unless you already have made an unsecured copy.)

If the `UserObject` does not have a panel view, the secured version is displayed only as an icon. The open view is not available, so no one can view or edit the contents.

If the `UserObject` has a panel view, the secured version is displayed either as an icon or open, showing the panel view. The detail view is not available, and the panel view cannot be modified.

Creating and Using UserFunctions

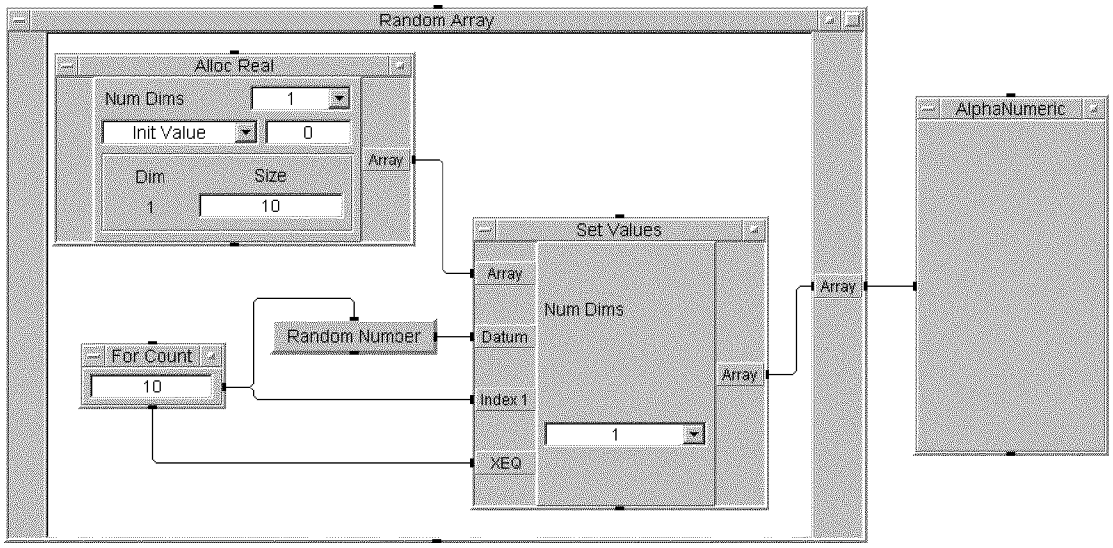
A UserFunction is a user-defined function created from a `UserObject` by selecting `Make UserFunction` from the object menu. The UserFunction exists in the background within the HP VEE process, but provides the same functionality as the original `UserObject`. You can call a UserFunction with the `Call Function` object, or from an expression. The major advantage of creating a UserFunction is that you can call the same UserFunction several times in your program. Thus, there is only one UserFunction to maintain, rather than several copies of a `UserObject`.

We'll just look at the basics of creating and using UserFunctions here. For a complete discussion of UserFunctions, as well as Compiled Functions and Remote Functions, refer to the "Creating User-Defined Functions" chapter in the *HP VEE Advanced Programming Techniques* manual.

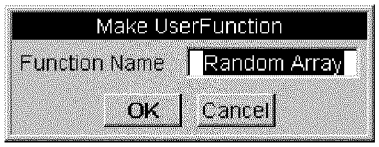
To Create a UserFunction

The first step in creating a UserFunction is to create a `UserObject`. We'll start with one of the examples from earlier in this chapter.

1. Open the program named `manual25.vee` (saved in your manual examples directory).

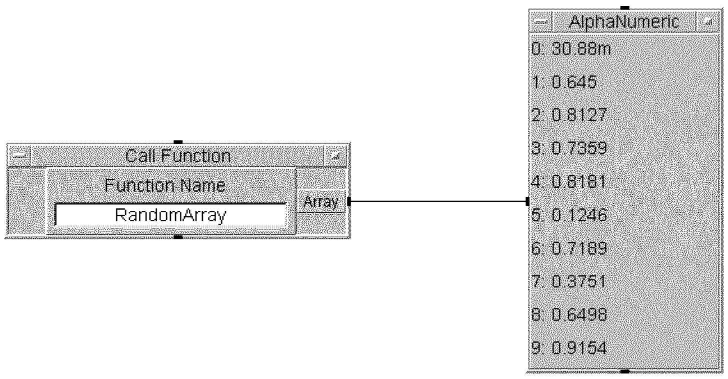


2. Select `Make UserFunction` from the object menu of the `Random Array UserObject`. If the `UserObject` name isn't valid as a function name, you'll be prompted for a function name:



3. Assign a function name (for example, `RandomArray`) that conforms to the following:
   - Only alphanumeric characters and underscores are allowed (no hyphens, spaces, commas, or other punctuation).
   - The name must be unique (not used for another function).

4. The **`UserObject`** is transformed into a UserFunction, and a **`Call Function`** object is automatically created as an example of how to call the new UserFunction. When you run the program, a set of 10 random numbers is generated:

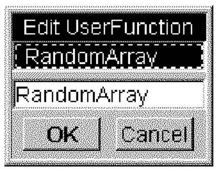


```
0: 30.88m
1: 0.645
2: 0.8127
3: 0.7359
4: 0.8181
5: 0.1246
6: 0.7189
7: 0.3751
8: 0.6498
9: 0.9154
```

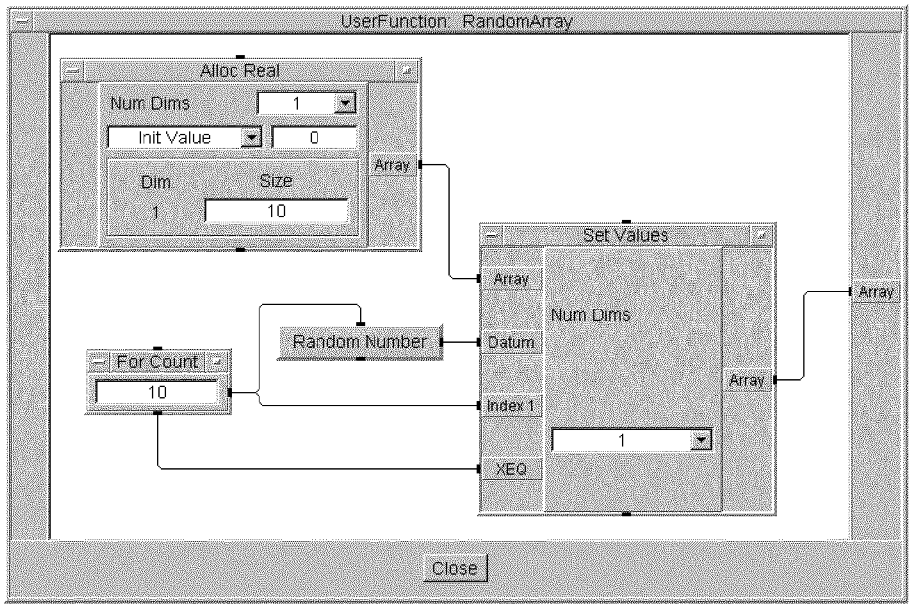
To Edit a UserFunction

To edit a UserFunction, the HP VEE program containing the UserFunction must be open. Let's edit the UserFunction you created in the last section:

1. With the program of the previous section still open, select **`Edit`** $\Longrightarrow$ **`Edit UserFunction`**. The **`Edit UserFunction`** dialog box appears:



2. Select the UserFunction named `RandomArray` (or whatever name you assigned), and click on `OK`. The `Edit UserFunction` work area appears:



3. Make your changes. For example, change both the `Size` parameter in the `Alloc Real` object and the value in the `For Count` object to 9. This changes the UserFunction to generate 9, rather than 10, random numbers.
4. Click on the `Close` button. Any changes that you have made will affect *all* calls to the UserFunction (see the next section).

Shortcut

You can also select `Edit UserFunction` from the object menu of the `Call Function` object, which opens the called function for editing.

For further information about editing a UserFunction, refer to the "Creating User-Defined Functions" chapter in *HP VEE Advanced Programming Techniques*.

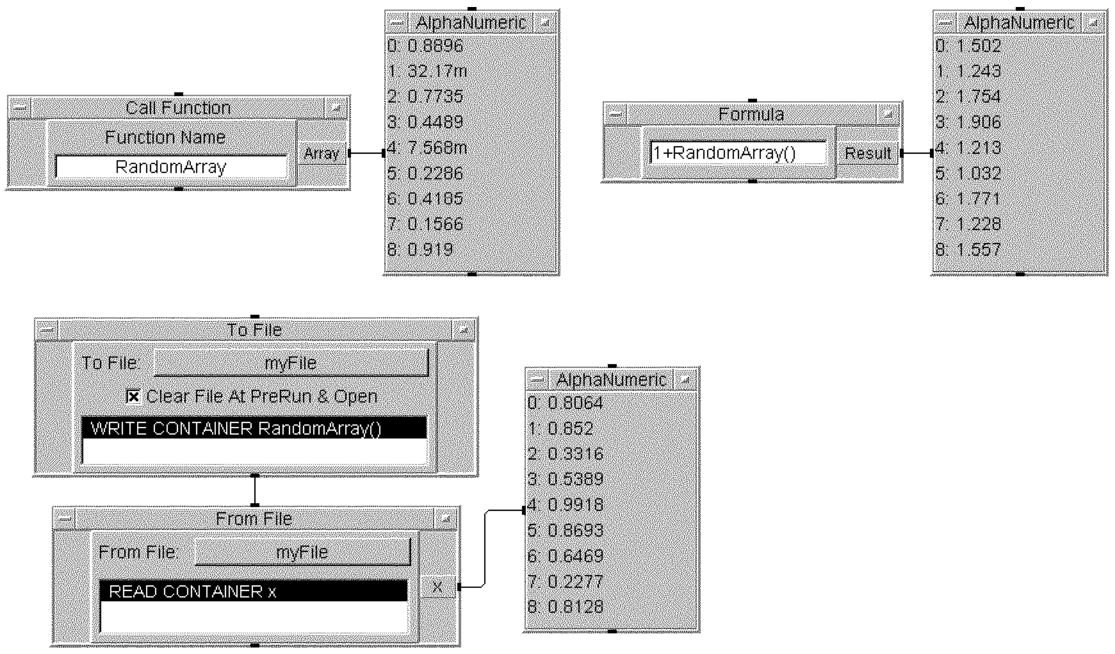
To Call a UserFunction in a Program

Once you have created a UserFunction, you can call it multiple times within an HP VEE program. You can call the UserFunction by using the `Call Function` object, or by including the UserFunction name in an expression. In the following example (`manual51.vee` in your manual examples directory), the `RandomArray` UserFunction (created and edited in the previous two sections) is called three times:

- The `Call Function` object calls `RandomArray` and outputs the returned array, as we've seen before.
- The expression `1+RandomArray()` in the `Formula` object adds `1` to each element of the returned array.
- The transaction `WRITE CONTAINER RandomArray()` in the `To File` object writes the array returned by `RandomArray` to the file `myFile` as a container.

NOTE

You can also include a UserFunction name in an expression in the `Sequencer` object. Refer to the "Using the Sequencer Object" chapter in the *HP VEE Advanced Programming Techniques* manual for further information.



Here are some key points:

- You can call the **`RandomArray`** UserFunction as many times as you like within a program.
- Each call of a UserFunction is independent. (Each call to **`RandomArray`** generates a different set of random numbers.)
- In the **`Call Function`** object, just specify the name of the function to call (for example: **`RandomArray`**). However, if you call the UserFunction from an expression, you must use the expression syntax, for example: **`RandomArray()`**.
- UserFunction names are case-insensitive (like global variable names).
- UserFunction names must be unique.

For a complete discussion of calling and using UserFunctions, refer to the "Creating User-Defined Functions" chapter in *HP VEE Advanced Programming Techniques*.

9

Debugging and Optimizing an HP VEE Program

Debugging and Optimizing an HP VEE Program

In this chapter we'll look at some techniques for debugging your program. Also, we'll look at how to optimize your program, both to make your program easy to understand and maintain, and to improve performance.

Debugging an HP VEE Program

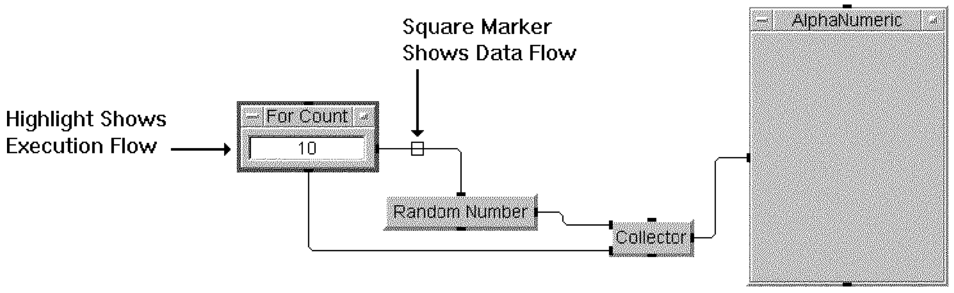
The following are some useful techniques for debugging your program.

To View Data Flow and Program Execution

Use the **Animate** feature to view data flow and program execution in a running HP VEE program. To turn animation on:

1. Select the **Edit** menu and look at the **Animate** selection. If there is a checkmark in front of it, animation is already on.
2. If there is no checkmark, click on **Animate** to turn it on.

The following program (**manual36.vee** in your manual examples directory) shows how animation works. Open the program, turn animation on as described above, and run the program.



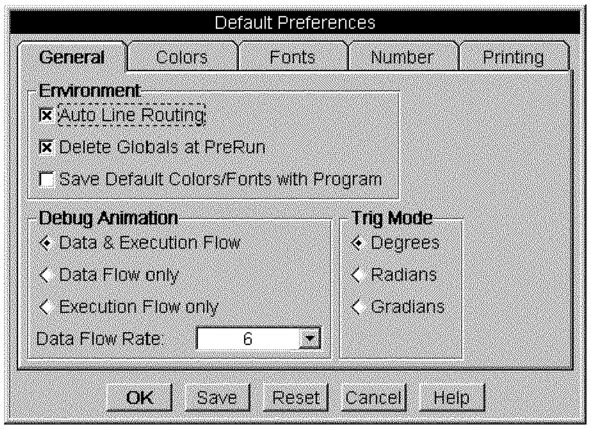
By default, **Data & Execution Flow** is shown by **Animate**:

- Data containers are indicated by small square markers (commonly called "torpedoes" by HP VEE programmers). For example, in the above figure a data container is traveling from the **For Count** object to the **Random Number** object.

- Execution flow is shown by highlighting each object as it operates. The highlight is a yellow outline around the object. For example, in the figure, the `For Count` object is highlighted, showing that it is operating.

You can change the animate feature's operation from the `Default Preferences` dialog box:

1. Select `File` $\Longrightarrow$ `Edit Default Preferences` to show the dialog box:



Environment
[✓] Auto Line Routing
[✓] Delete Globals at PreRun
[ ] Save Default Colors/Fonts with Program

Debug Animation
[✓ Data & Execution Flow]
Data Flow only
Execution Flow only
Data Flow Rate: 6

Trig Mode
[✓ Degrees]
Radians
Gradians

2. Under `Debug Animation` you can select one of the following by clicking on its radio button:

   - `Data & Execution Flow` (the default)
   - `Data Flow Only`
   - `Execution Flow Only`

   You can also change the animation speed. Click on the `Data Flow Rate` arrow to show the drop-down list. You can set a rate from `1` (the slowest) to `10` (the fastest).

3. Click on `Save` if you want the changes to be saved as new defaults (in `VEE.RC` or `.veerc`). Or click on `OK` if you want to use the changes for this HP VEE session only (until you select `File` $\Longrightarrow$ `New`, `Open`, or `Exit`).

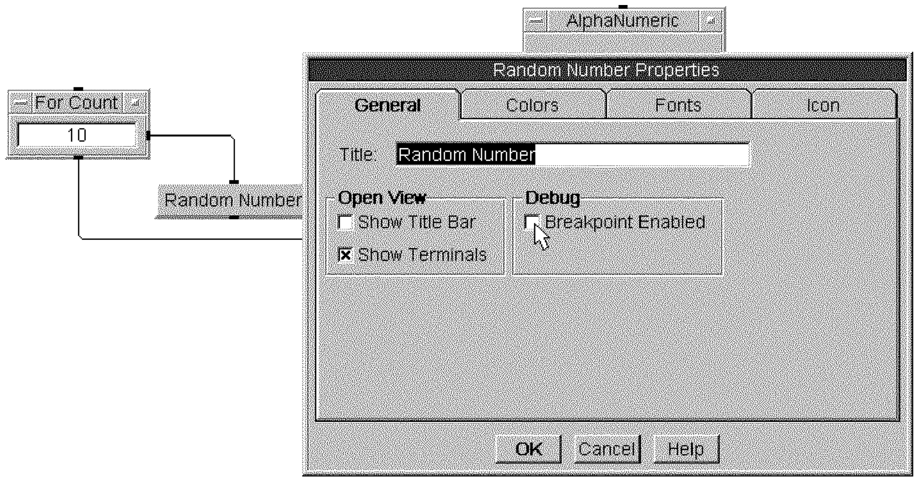
Be sure to turn `Animate` off when you don't need it. It slows down your program considerably.

To Set and Delete Breakpoints

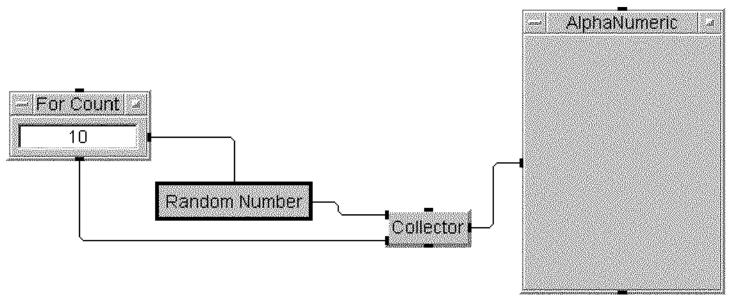
If you set a **breakpoint** on an object in your program, the program will pause just before that object operates. You can then use **`Line Probe`** to see the data containers on each line, and you can use **`Step`** or **`Cont`** to continue program execution.

Let's add a breakpoint to the **`Random Number`** object in the program of the previous section:

1. Open the program (**`manual36.vee`** in your manual examples directory).
2. Select **`Edit Properties`** from the object menu of the **`Random Number`** object. Then, click on the checkbox for **`Breakpoint Enabled`**:



3. Click on **`OK`**. The **`Random Number`** object is now highlighted with a black outline, showing that it has a breakpoint set.



4. Run the program.

Note that the program pauses just before the **Random Number** object operates. Click on the **Cont** button to continue the program. The program again pauses just before the **Random Number** object operates. Use a breakpoint to pause the program at a desired point, then press **Cont** to resume execution or **Step** to proceed to the next object.

To delete a breakpoint:

1. Select **Edit Properties** from the object menu of the object that has the breakpoint set.
2. Click on the **Breakpoint Enabled** checkbox in the dialog box to turn off the breakpoint, and then click on **OK**.

Shortcuts:

Use the following features (found under **Edit** $\Longrightarrow$ **Breakpoints** $\Longrightarrow$) to enable and disable breakpoints:

- Use **Set Breakpoints** to set breakpoints on multiple objects. Select each desired object using **Select Objects**, and then select **Set Breakpoints**.
- Use **Clear Breakpoints** to clear breakpoints on multiple objects. Select each desired object using **Select Objects**, and then select **Clear Breakpoints**.
- Use **Clear All Breakpoints** to clear all breakpoints from a program once you have finished debugging it.

- When `Activate Breakpoints` is checked, breakpoints are active and you can debug your program. When `Activate Breakpoints` is not checked, breakpoints are disabled.

To Step through a Program

Use the `Step` button in the HP VEE tool bar to run your program one object at a time.

- From a stopped program, the first time you press `Step` the program PreRuns. Press `Step` to operate each object in turn. A green highlight indicates the object that will operate next.
- If you run your program (by pressing `Run`) with a breakpoint enabled, the program will PreRun and then execute up to the breakpoint. From there, you can use `Step` to operate each object in turn.
- At any point, you can press `Cont` to continue running the program. To pause the program, press `Stop` once. If you press `Stop` twice, the program will stop, and will go through PreRun when restarted.

When you encounter a `UserObject` when stepping through a program, the following occurs:

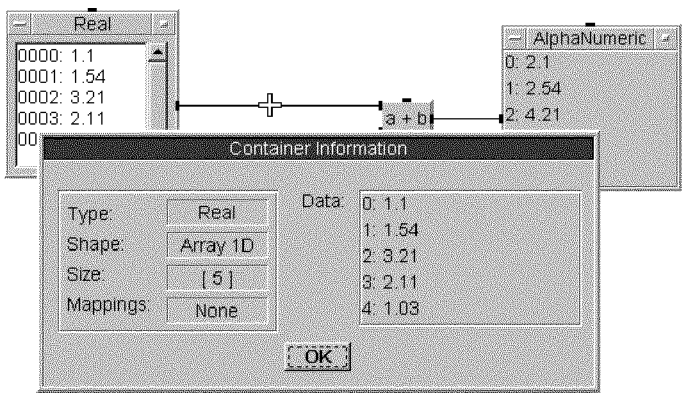
- If the `UserObject` is in its open view, the objects in the `UserObject` operate one at a time. You'll need to press `Step` for each object executed in the `UserObject`.
- If the `UserObject` is an icon, is showing a panel view, or has `Show Panel on Exec` set, the entire `UserObject` operates when you press `Step`.

To View the Data on a Line

Use **Line Probe** to view the data container that has been passed on any HP VEE data line. With the program paused or stopped:

1. Select **Edit** $\Longrightarrow$ **Line Probe**.
2. Move the mouse pointer near the line to be probed and click the left mouse button.

The **Container Information** dialog box appears. In the example below, the data output line from the **Real** constant object has been probed.



Type: Real
Shape: Array 1D
Size: [5]
Mappings: None
Data: 0: 1.1
1: 1.54
2: 3.21
3: 2.11
4: 1.03

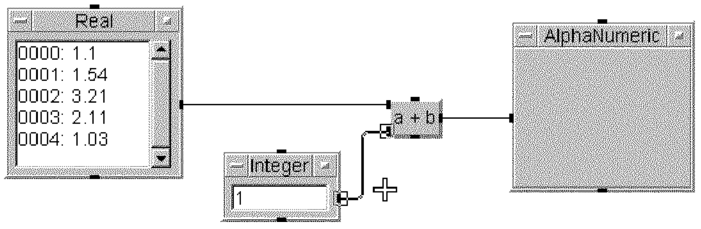
If you probe a line on which no data has been transmitted (or on which a sequence pin "ping" has been transmitted), the **Container Information** dialog box reports **Nil (empty) container**.

Shortcut

To probe a line, move the mouse pointer near the line, press and hold the Shift key, and click the left mouse button.

To Find the Endpoints of a Line

Use **Line Probe** to trace a line and show its endpoints. Select **Line Probe** (Shift+left mouse button) and *drag* the mouse pointer across the line. The line endpoints are indicated with small square markers, as shown:



If you release the mouse button over a line, the **Container Information** dialog box appears.

To Trap an Error

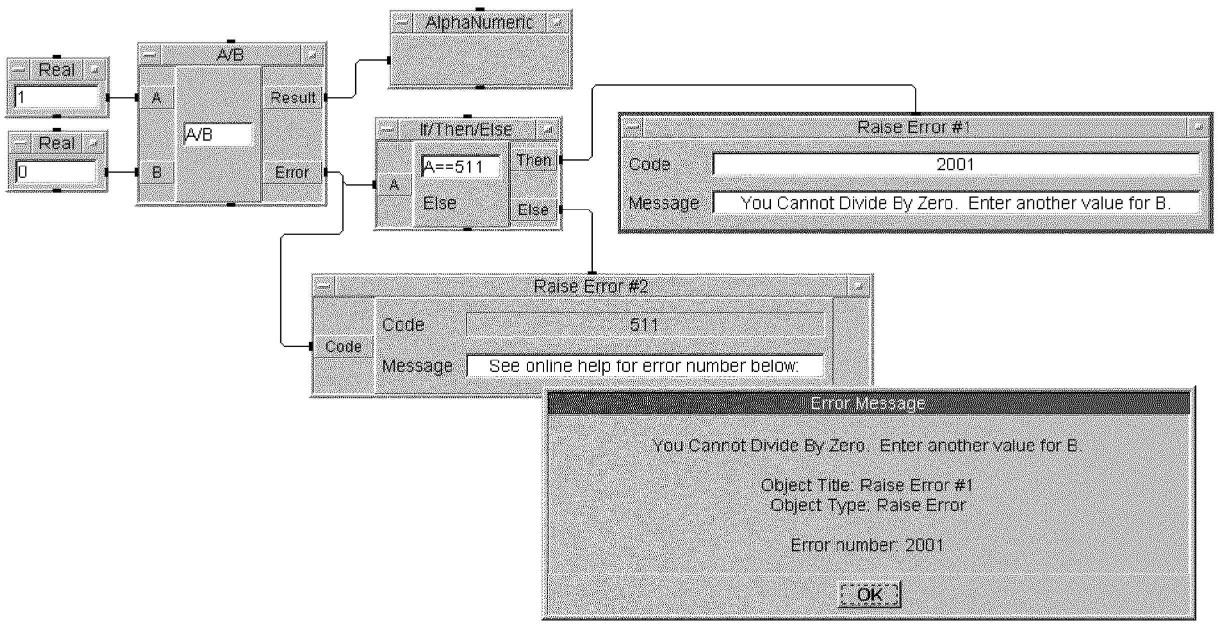
Normally, all HP VEE errors are reported by the **Error Message** dialog box. However, you can trap an error generated by a particular object by adding an **Error** output terminal to that object.

To add an **Error** output terminal to an object, select **Add Terminal** ⟹ **Error Output** from the object menu.

If an object that has an **Error** output terminal generates an error, the **Error Message** dialog does not appear. Instead, the **Error** terminal outputs the error number.

There are various ways to use the **Error** output terminal. You can use it to activate another object, such as **Raise Error**. Or you can use **If/Then/Else** to determine an action based on the error number.

The following program (`manual37.vee` in your manual examples directory) illustrates some useful techniques for trapping errors. In this program, we expect error number 511 (**`Division/Modulo by zero`**) to occur, and we want to trap that error.



Code 2001
Message You Cannot Divide By Zero. Enter another value for B.

Code 511
Message See online help for error number below:

Error Message

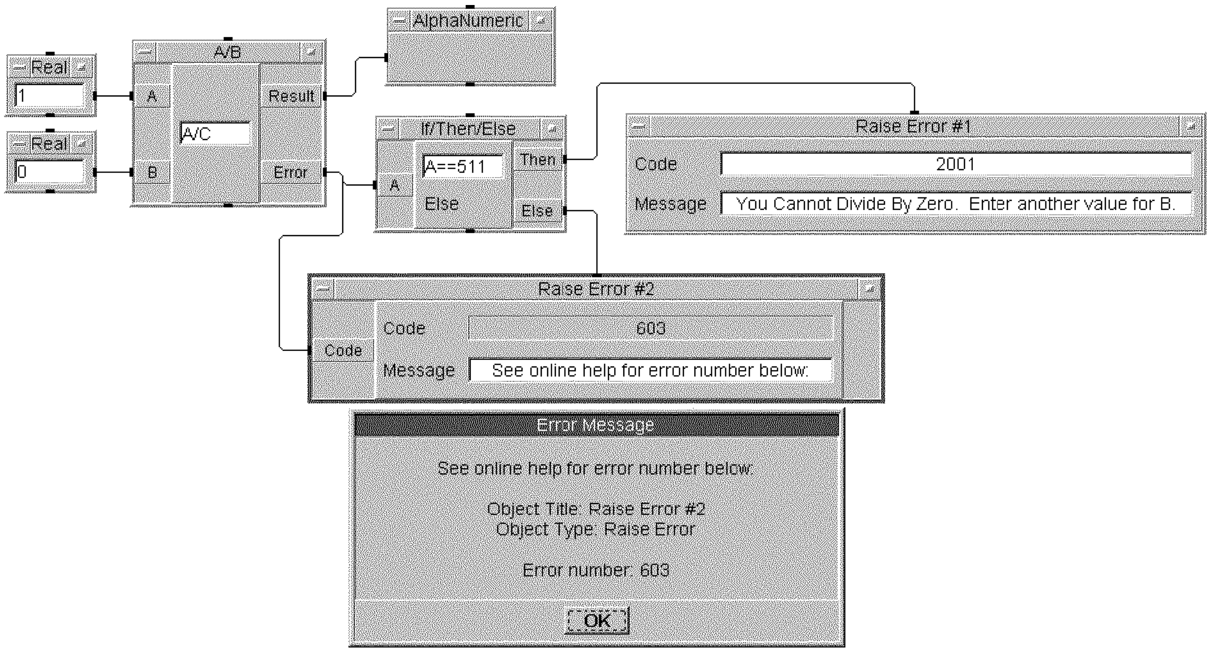
You Cannot Divide By Zero. Enter another value for B.

Object Title: Raise Error #1
Object Type: Raise Error

Error number: 2001

When a division-by-zero condition occurs, the **`Error`** terminal on the **`A/B`** object outputs error number **`511`**. The **`If/Then/Else`** object tests the error number for equality to **`511`** and activates its **`Then`** terminal, causing the **`Raise Error #1`** object to operate. **`Raise Error #1`** generates the error dialog shown, displaying the message **`You Cannot Divide By Zero. Enter another value for B.`** and the error code **`2001`**.

But, we also need to provide for other errors that the **`A/B`** object might generate. (Remember, with an **`Error`** terminal present, the normal HP VEE error message dialog won't appear.) For this, the program has a second **`Raise Error`** object connected to the **`Else`** terminal of the **`If/Then/Else`** object:



See online help for error number below:

Object Title: Raise Error #2
Object Type: Raise Error

Error number: 603

In the above figure, we've forced the `A/B` object to generate error number 603 (`Variable was not found: C`) by changing the expression to `A/C`. The `If/Then/Else` object tests for equality to `511`, and activates its `Else` terminal, which activates the sequence input pin on `Raise Error #2`.

`Raise Error #2` passes through the error number (`603`) on its `Code` data input terminal, and generates the error dialog shown. It displays `See online help for error number below:`, along with the error number `603`.

This technique is useful for trapping one or more specific errors, and providing an "escape" to pass through other errors generated by an object. You can use as many `If/Then/Else` and `Raise Error` objects as are necessary.

Making an HP VEE Program Easier to Understand

The following are some techniques for making your program easier to understand and maintain.

To Add Structure to Your Program

Just as you can add structure to a program in a conventional programming language, you can add structure to an HP VEE program.

1. Analyze what your HP VEE program does. Can you identify a group of objects that together perform a specific sub-task? Create a `UserObject` from those objects.
2. Are there sub-tasks that appear multiple times in your program? If so, create a `UserObject` to do the sub-task, convert the `UserObject` into a UserFunction, and call the UserFunction each time it is needed.
3. Proceed through your program, building a hierarchical structure using `UserObjects` and UserFunctions.
4. If you pass a single data container to several objects, consider using a global variable to pass that data.

For further information on `UserObjects` and UserFunctions, refer to Chapter 8.

To Document Your Program

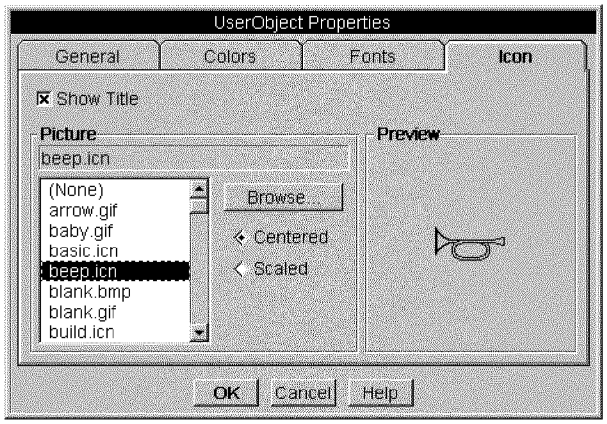
You can make your program easier to use, debug, and maintain by using the following techniques:

- Give your program a descriptive name. You can name your program in the HP VEE title bar. (Double-click on the title area and change the title in the dialog box.)
- Rename objects to names that are more meaningful to you. For example, `Repeat 100x` may be more useful than `For Count`. (Double-click on the title area and change the title in the dialog box.)
- Rename input and output terminals to descriptive names. For example, if terminal `A` inputs a flow rate, you may want to change the name to `Flow_Rate`. (Double-click on the terminal and change the title in the dialog box.) Note that you cannot change the title of some terminals.
- Add descriptions to key objects by using `Edit Description` in the object menu. The information may include why you used that particular object, details about the inputs and outputs, and the options that you used on the object.
- Add notes to yourself or others using the `Note Pad` display object (`Display` $\Longrightarrow$ `Note Pad`). The information could include:
  - □ Your name, phone number, and the date you created the program.
  - □ What this program does.
  - □ The dates that you made changes and what the changes were.
- Customize icons, such as `UserObject` icons, by adding a descriptive bitmap. Refer to "To Add a Bitmap to an Icon" for further information.

To Add a Bitmap to an Icon

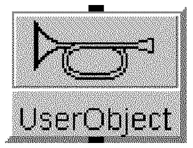
You can add a bitmap to any iconified HP VEE object. For example, to add a bitmap to a `UserObject` icon, follow these steps:

1. Select `Edit Properties` from the object menu of the `UserObject`. Then select the `Icon` tab.
2. The available bitmaps in your `bitmaps` directory are displayed. Scroll through the list and click on the file name that you want. For example, if you click on `BEEP.ICN`, the `Beep` icon is shown in the preview area:



(If you want to browse through other directories, click on `Browse`. The `Select File` dialog box appears and you can navigate through the directories to select a bitmap file.)

3. Once you have selected an icon, click on `OK`. The following is a `UserObject` icon with the `Beep` bitmap:



The procedure for adding a custom bitmap is the same as for adding one of the standard bitmaps. However, you must first create the bitmap. To create a custom bitmap, use an appropriate bitmap editor for your computer (for example, the Microsoft Windows PaintBrush program for a PC). The following file formats are supported:

- Windows: BMP, ICN, or GIF
- UNIX: BMP, ICN, GIF, or XWD

The default bitmap directories are as follows:

- Windows: `C:\VEE\BITMAPS`
- UNIX: `/usr/lib/veetest/bitmaps`

Improving the Performance of an HP VEE Program

The following sections describe some techniques that you can use to optimize the performance of your HP VEE program.

To Remove Barriers to Program Performance

The following are some general techniques that you may be able to use to improve program execution speed.

- Leave data input terminals set to `Any` for data type and shape. HP VEE will convert data types only when necessary, saving execution time.
- Turn off `Clear at PreRun` and `Clear at Activate` on displays where not needed.
- Use `Initialize at PreRun` and `Initialize at Activate` on data objects instead of setting defaults with control pins.
- Collect data for graphical displays into an array, and plot the *entire* array at once rather than plotting each individual scalar point. If the `X` values of a plot are regularly spaced, use an `XY Trace` object rather than an `X vs Y Plot` object.
- Run the program from the panel view. If the panel view contains fewer objects than the detail view, the program will run faster.
- Set graphical displays to be as plain as possible. The settings that allow the fastest update times are `Grid Type` ⟹ `None` and `Layout` ⟹ `Graph Only`.
- Iconify those objects that continuously update their displays (such as the `Timer`, `Counter`, `Accumulator`, `AlphaNumeric`, and graphical display objects) if you don't need to see the displayed information.
- If using a `Strip Chart` display, set the buffer size as small as possible.
- Collect data for processing into an array, and then process the data as an array.

- Use Complex data in expressions rather than PComplex. Most of the math libraries will convert PComplex to Complex, calculate the answer, and convert Complex back to PComplex.
- To display PComplex data, set **`Trig Mode`** (under **`File`** ⟹ **`Edit Properties`**) to **`Radians`**. HP VEE stores PComplex values internally as radians.
- In general, the fewer objects that need to operate, the faster the program will run. Perform as many functions as possible in each object. For further information about objects performing multiple functions, refer to the following section.

To Reduce the Number of Objects in a Program

Many HP VEE objects can perform a set of functions. Your programs will be more compact, easier to maintain, and will run faster if you use the following techniques:

- Enter equations in a **`Formula`** object instead of using multiple constant and single-function math objects. You can nest functions in a **`Formula`** object. For example, you can enter the expression **`(sin(ramp(100, 0, 360)))`**.
- Use one **`If/Then/Else`** object with multiple conditions (for example, `A == B OR A<=C AND A>D`) instead of using multiple **`If/Then/Else`** or **`Conditional`** objects.
- To input a one-dimensional array of data, use a **`Constant`** object configured as an array instead of using one **`Constant`** object for each value and then building an array.
- To read all of “the rest” of the data in a file, use the `ARRAY 1D TO END:(*)` transaction in a **`From File`** object. This is simpler and faster than looping on single-element reads and collecting the result into an array.
- Use the **`Sequencer`** to control the flow of execution of several UserFunctions. Refer to the chapter “Using the Sequencer Object” in *HP VEE Advanced Programming Techniques* for further information.

- When using the **Sequencer**, only enable logging for transactions where the **Log** record is required. If the **Log** output terminal is not used, delete it to speed up execution slightly.

To Increase I/O Performance

In general, direct I/O provides the fastest I/O performance. To optimize the I/O performance of your HP VEE program:

1. Create your program using an instrument panel object (or component driver object) to communicate with the instrument. Test and debug the program and instrument configuration.
2. Add a **Bus I/O Monitor** object to the program. Run the program and "capture" the messages sent to the instrument, and received from it, by the instrument panel or component driver.
3. Rewrite the program so that it uses a **Direct I/O** object to communicate with the instrument using the "captured" messages. This will speed up I/O performance in the final program.

For further information, refer to Chapter 5.

In rare cases where extremely fast I/O is required, you can create a compiled function to perform the instrument I/O function. Refer to the "User-Defined Functions" chapter in *HP VEE Advanced Programming Techniques* for further information.

A

Data Type Conversion and Array Mappings

Data Type Conversion and Array Mappings

This appendix provides information about how HP VEE converts data types internally, and how arrays are mapped.

Data Type Conversion

The following sections describe what HP VEE does, and when, to convert between data types.

Converting Data Types on Input Terminals

In conventional programming languages, you manually convert between data types. HP VEE automatically converts between most data types.

NOTE

Data shapes are not converted on input terminals, but data types and shapes may be automatically converted when used in math functions. Refer to the "Mathematically Processing Data" section in Chapter 3 of the *HP VEE Reference* manual for further information.

Most objects accept any data type on their data input terminals, but a few objects require a particular data type or shape. For these objects, the data input terminal automatically attempts to convert the input container to the desired data type.

For example, a **`Magnitude Spectrum`** display needs Spectrum data. If the output of a **`Function Generator`** (a Waveform) is connected to the **`Magnitude Spectrum`** display, the input terminal of the **`Magnitude Spectrum`** automatically does an FFT to convert time-domain data to frequency-domain data (Waveform to a Spectrum).

The conversion can be a promotion or demotion. A **promotion** is the conversion from a data type with less information to one with more. For example, a conversion from an Int32 to Real is a promotion. Such promotions take place automatically as needed — you rarely if ever need to be concerned with them.

A **demotion** is a conversion that loses data. For example, the conversion from a Real to an Int32 is a demotion because the fractional part of the Real number is lost. A demotion of data type occurs *only* if you force it by specifying a certain data type for an input on an object. Once you have specified a data type, the demotion will occur automatically if it is needed and is possible.

For example, if you change the input on a **Formula** object to Int32, and another object supplies a Real number to that input (such as 28.2), the value will be demoted to an Int32 (28).

To change the data type on the **Formula** input from **Any** to **Int32**, just double-click on the input terminal's information area (not the pin), and then click on the **Required Type** field. Click on **Int32** in the drop-down list to change types.

NOTE

The conversion of data types for instrument I/O transactions is a special case. Refer to "Instrument I/O Data Type Conversions" for further information.

When the conversion can't be done, HP VEE returns an error. The following table shows when conversion is automatic (yes) or when HP VEE returns an error (no). Demotions are indicated by shading.

NOTE

The Record data type has the highest priority. However, HP VEE does not automatically promote to or demote from the Record data type. To convert between Record and non-Record data, use **Build Record** and **Unbuild Record**. For further information, refer to "Using Records and DataSets" in the *HP VEE Advanced Programming Techniques* manual.

Table A-1. Promotion and Demotion of Types in Input Terminals

To ▶ ▼ From	Int32	Real	Complex	PComplex	Waveform	Spectrum	Coord	Enum	Text
Int32	n/a	yes	yes(1)	yes(1)	no	no	yes(2)	no	yes
Real	yes(3)	n/a	yes(1)	yes(1)	no	no	yes(2)	no	yes
Complex	no	no(4)	n/a	yes	no	no	no	no	yes
PComplex	no	no(4)	yes	n/a	no	no	no	no	yes
Waveform	yes(3)	yes(8)	no	no	n/a	yes(5)	yes	no	yes
Spectrum	no	no	yes(8)	yes(8)	yes(5)	n/a	yes	no	yes
Coord	no	no	no	no	no	no	n/a	no	yes
Enum	no(6)	no	no	no	no	no	no	n/a	yes
Text	yes(7)	yes(7)	yes(7)	yes(7)	no	no	yes(7)	no	n/a

Notes:

n/a = Not applicable.

(1) An Int32, or Real *value* promotes to Complex (*value*, 0) or to PComplex (*value*, @0).

(2) The independent component(s), which are the first `n-1` field(s) of an `n`-field Coord, are the array indexes of the value unless the array is mapped. If the array is mapped, the independent component(s) are derived from the mappings of each dimension. The dependent component, `y`, is the array element. If the container is a Scalar (non-array), conversion fails with an error.

(3) These demotions will cause an error if the value is out of range for the destination type.

(4) This demotion is not done automatically, but can be done with the `re(x)`, `im(x)`, `mag(x)`, and `phase(x)` objects or the `Build/UnBuild` ⟹ objects.

(5) An FFT or inverse FFT is automatically done.

(6) This demotion is not done automatically, but can be done with the `ordinal(x)` object.

(7) This demotion causes an error if the text value is not a number (such as `34` or `42.6`) or is not in an acceptable numerical format. The acceptable formats are as follows (spaces, except within each number, are ignored):

- Text that is demoted to an Int32 or Real type may also include:
  - □ A preceding sign. For example, `-34`.
  - □ A suffix of `e` or `E` followed by an optional sign or space and an integer. For example, `42.6E-3`.
- Text demoted to Complex must be in the following format: *(number, number)*.
- Text demoted to PComplex must be in the following format: *(number, @number)*. The phase (the second component) is considered to be radians for this conversion, regardless of the `Trig Mode` setting.
- Text demoted to a Coord type must be in the following format: *(number, number, . . .)*.

(8) These demotions keep the Waveform and Spectrum mappings.

Instrument I/O Data Type Conversions

On instrument I/O transactions involving integers, HP VEE performs an automatic data-type conversion according to the following rules:

NOTE

These data-type conversions are completely automatic, so you won't normally need to be concerned with them. However, the following list shows what happens.

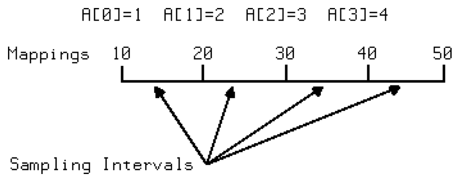
- On an input transaction (read), `Int16` or `Byte` values from an instrument are converted to `Int32` values, preserving the sign extension. Also, `Real32` values from an instrument are converted to 64-bit `Real` numbers.
- On an output transaction (write), `Int32` or `Real` values are converted to the appropriate output format for the instrument:
  - ☐ If an instrument supports the `Real32` format, HP VEE converts 64-bit `Real` values to `Real32` values, which are output to the instrument. If the `Real` value is outside of the range for `Real32` values, an error will occur.
  - ☐ If an instrument supports the `Int16` format, HP VEE truncates `Int32` values to `Int16` values, which are output to the instrument. `Real` values are first converted to `Int32` values, which are then truncated and output. However, if a `Real` value is outside the range for an `Int32`, an error will occur.
  - ☐ If an instrument supports the `Byte` format, HP VEE truncates `Int32` values to `Byte` values, which are output to the instrument. `Real` values are first converted to `Int32` values, which are then truncated and output. However, if a `Real` value is outside the range for an `Int32`, an error will occur.

Array Mappings

A mapping is a set of continuous or discrete values that express the independent variables for an array. For example, the mappings on a Waveform are the times for each amplitude value. Waveform, Spectrum, and Coord data types are mapped. Arrays of other data types can also be mapped by using **Data** $\Longrightarrow$ Access Array $\Longrightarrow$ Set Mappings''.

Mappings are either continuous (Waveform, Spectrum, or mapped arrays created with **Set Mappings**) or discrete (Coord). Continuous mappings are attached to the array and may be viewed on the terminal or by using **Line Probe**, but are not part of the array and are different than the array indices. Discrete mappings are part of the array and are displayed as the first n-1 fields in a Coord value with n fields.

The sampling interval of linear mappings is the maximum value minus the minimum value, divided by the number of points. There are the same number of points as sampling intervals; each point is at the beginning of a sampling interval. For example, if array **A** (where **A = [1, 2, 3, 4]**) is linearly mapped from **10** to **50**, the mappings and sampling intervals are as shown in the figure below. Note that the mappings are from the first element to the end of the last interval.



When mapped values (except Coord) are displayed, the **X** axis displays the mappings.

To get information about a continuous-mapped array, use **Data** $\Longrightarrow$ Access Array $\Longrightarrow$ Get Mappings''. **Get Mappings** gives you the type of mapping (log or linear) and the minimum and maximum mapping values for each dimension.

Glossary

Glossary

This Glossary defines several terms used to name or describe HP VEE features.

Activate

1. To send a container to a terminal. See also "Container" and "Terminal."

2. The action that resets the context of a `UserObject` before it operates each time. See also "Context" and "PreRun."

Allocate

To initialize an HP VEE array. Allocation sets the number of dimensions, the number of elements in each dimension, and fills the array with initial data. For example, a two-dimensional integer array is filled with `0` values by default.

Array

An HP VEE data shape corresponding to a mathematical array, which contains a grouping of data items (all of the same data type) in one or more dimensions. The data items are accessed by means of array indexes. See also "Data Shape."

Asynchronous

In asynchronous operation, a device operates without a common signal to synchronize events—the events occur at unspecified times. HP VEE control pins (for example, the `Clear` pin on a display object) are asynchronous.

Auto Execute

An option on the object menus of the data constant objects. If `Auto Execute` is set, and if the program is not already running, the object operates when its value is edited. `Auto Execute` is ignored while the program is running.

Bitmap

A bit pattern or picture. In HP VEE you can display a bitmap with a `Picture` object or on any icon. You can also display a bitmap on the background of a main program panel view, a `UserObject` panel view, or a UserFunction panel view.

Breakpoint
A tool for debugging an HP VEE program. If you set a breakpoint for an object, the program will pause *just before* that object executes.

Buffer
An area in memory where information is stored temporarily.

Button
A graphical object in HP VEE that simulates a momentary switch or selection button, and which appears to pop out from your screen. When you "press" a button in HP VEE, by clicking on it with the mouse, an action occurs. (May also refer to the left or right mouse button.)

Cascading Menu
A sub-menu on a pull-down or pop-up menu that provides additional selections.

Checkbox
A recessed square box on HP VEE menus and dialog boxes that allows you to select a setting. To select a setting, click on the box and an "x" appears in the box to indicate a selection has been made. To cancel the setting, simply click on the box again.

Click
To press and release a mouse button. Clicking usually selects a menu feature or object in the HP VEE window. See also "Double-Click" and "Drag."

Compiled Function
A user-defined function created by dynamically linking a program, written in a programming language such as C, into the HP VEE process. For UNIX systems, the user must create a shared library file and a definition file for the program to be linked. For Windows, the user must create a DLL (Dynamically Linked Library) file and a definition file. The `Import Library` object attaches the shared library or DLL to the HP VEE process and parses the definition file declarations. The Compiled Function can then be called with the `Call Function` object, or from certain expressions. See also "UserFunction" and "Remote Function."

Component
A single instrument function or measurement value in an HP VEE instrument panel or component driver. For example, a voltmeter driver contains components that record the range, trigger source, and latest

reading. See also "Component Driver," "Driver Files," "State," and "Instrument Panel."

Component Driver
An instrument control object that reads and writes values to components you specifically select. Use component drivers to control an instrument using a driver by setting the values of only a few components at a time. (Component drivers do not support coupling.)

Composite Data Type
A data type that has an associated shape. See also "Data Shape" and "Data Type."

Container
See "Data Container."

Context
A level of the work area that can contain other levels of work areas (such as nested `UserObjects`), but is independent of them.

Control Pin
An asynchronous input pin that transmits data to the object without waiting for the object's other input pins to contain data. For example, control pins in HP VEE are commonly used to clear or autoscale a display.

Coupling
The inter-relationship of certain functions in an instrument. If, in an instrument panel, functions A and B are coupled, changing the value of A may automatically change the value of B, even though you do not change B explicitly.

Cursor
A pointer (caret) in an entry field that shows where alphanumeric data will appear when you type information from the keyboard.

Cut Buffer
The buffer that holds objects that you cut or copy. You can then paste the object back into the work area with `Edit` ⟹ `Paste`

Data Container
The data package that is transmitted over lines and is processed by objects. Each data container contains data and the data type, data shape, and mappings (if any).

Data Field
The field within a transaction specification in which you specify either the expression to be written (WRITE transactions), or the variable to receive data that is read (READ transactions). See also "Transactions."

Data Flow
The flow of data through and between HP VEE objects. Data flows from left to right through objects, but an object does not execute until it has data on all of its data input pins. Data is propagated from the data output pin of one object to the data input pin of the next object. Data flow is the chief factor that determines the execution of an HP VEE program.

Data Input Pin
A connection point on the left side of an object that permits data to flow into the object.

Data Output Pin
A connection point on the right side of an object that propagates data flow to the next object and passes the results of the first object's operation on to the next object.

DataSet
A collection of "Record" containers saved into a file for later retrieval. The `To DataSet` object collects Record data on its input and writes that data to a named file (the DataSet). The `From DataSet` object retrieves Record data from the named file (the DataSet) and outputs that data as Record containers on its `Rec` output pin. See also "Record."

Data Shape
Each data container has both a shape and type. The data shape can be either a scalar or an array (Array 1D, Array 2D, and so forth).

Data Type
Each data container has both a type and shape. HP VEE supports several data types including Text, Real, and Integer.

DDE (Dynamic Data Exchange)
A communication mechanism that allows HP VEE for Windows to communicate with other Windows applications that support DDE. HP VEE can send data to, and receive data from, such applications. Also, HP VEE can execute commands in another application. Examples of Windows applications that support DDE are Microsoft Excel and Microsoft Word for Windows.

Default

A value or action that HP VEE automatically selects. For example, HP VEE uses certain default colors and fonts, which you can change using the `Default Preferences` dialog box.

Default Button

The button in a dialog box that is activated by default if (Enter) or (Return) is pressed, or the selection is double-clicked. The label of the default button is in bold text.

Demote

To convert from a data type that contains more information to one that contains less information. See also "Data Type" and "Promote."

Detail View

The view of an HP VEE program that shows all the objects and the lines between them.

Device

An instrument attached to or plugged into an HP-IB, RS-232, GPIO, or VXI interface. Specific HP VEE objects such as the `Direct I/O` object send and receive information to a device.

Device Driver

See "Interface Driver."

Dialog Box

A secondary window displayed when HP VEE requires information from you before it can continue. For example, a dialog box may contain a list of files from which you may choose.

Direct I/O Object

An instrument control object that allows HP VEE to directly control an instrument without using an instrument driver.

DLL (Dynamically Linked Library)

A collection of functions written in C that can be called from HP VEE for Windows. DLLs can be created by experienced C programmers using tools available from Microsoft and Borland. DLLs in the Windows environment are similar to shared libraries in the UNIX environment.

Double-Click

To press and release a mouse button twice in rapid succession. Double-clicking is usually a short-cut to selecting and performing an action. For example, double-clicking on a file name from `File` $\Longrightarrow$ `Open` will select the file and open it.

Drag

To press *and continue to hold down* a mouse button while moving the mouse. Dragging moves something (for example, an object or scroll bar).

Driver

Software that allows a computer to communicate with other software or hardware. See also "Component Driver," "Driver Files," "Interface Driver," and "Instrument Panel."

Driver Files

A set of files included with HP VEE that contains the information needed to create instrument panel and component driver objects for instrument control.

Drop-Down List

A list of selections obtained by clicking on the arrow to the right of a selection field.

Entry Field

A field that is typically part of a dialog box or an editable object, and which is used for data entry. An entry field is editable when its background is white.

Error Message

Information that appears in an error dialog box, explaining that a problem has occurred.

Error Pin

A pin that traps any errors that occur in an object. Instead of getting an error message, the error number is output on the error pin. When an error is generated, the data output pins are not activated.

Execute

The action of a program, or parts of a program, running.

Execution Flow
The order in which objects operate. See also "Data Flow."

Expression
An equation in an entry field that may contain input terminal names, global variable names, `Math` and `AdvMath` functions, and user-defined functions. An expression is evaluated at run-time. Expressions are allowed in `Formula`, `If/Then/Else`, `Get Values`, `Get Field`, `Set Field`, `Sequencer`, and `Dialog Box` objects, and in I/O transaction objects.

Feedback
A continuous thread path of sequence and/or data lines that uses values from the previous execution to change values in the current execution.

Flow
See "Data Flow" and "Execution Flow."

Font
HP VEE allows you to change the "font"—the size and style of type—used to display text for various HP VEE objects, titles, and so forth.

Function
The name and action of objects where the output is a function of the input. These objects are located under `Math` or `AdvMath` menus and may be used in the `Formula` object. For example `sqrt(x)` is a function; `+` is not.

Global Variable
A named variable that is set globally, and which can be used by name in any context of an HP VEE program. For example, a global variable can be set with `Set Global` in the root context of the program, and can be accessed by name with `Get Global` or from certain expressions within the context of a `UserObject`. However, a local variable with the same name as the global variable takes precedence in an expression.

Grayed Feature
A menu feature that is displayed in gray rather than black, indicating that the feature is not active or not available. Dialog box items such as buttons, checkboxes, or radio buttons may also be grayed.

Group Window

A group window in Microsoft Windows is a window that contains icons for a group of applications. Each icon starts an application in the group.

Highlight

1. The colored band or shadow around an object that provides a visual cue to the status of the object.

2. The change of color on a menu feature that indicates you are pointing to that feature.

Host

To begin a thread or subthread. For example, the subthread that is hosted by `For Count` is the subthread that iterates.

HP-UX

Hewlett-Packard Company's enhanced version of the UNIX operating system.

Hypertext

A system of linking topics so that you can jump to a related topic when you want more information. In online help systems, typically hypertext links are designated with underlined text. When you click on such text, related information is presented.

Icon

1. A small, graphical representation of an HP VEE object, such as the representation of an instrument, a control, or a display.

2. A small, graphical representation of a Microsoft Windows application within a group window. See "Group Window."

Instrument Driver

See "Driver Files," "Component Driver," and "Instrument Panel."

Instrument Panel

An instrument control object that forces all the function settings in the corresponding physical instrument to match the settings in the control panel displayed in the open view of the object.

Interface

HP-IB, RS-232, GPIO, and VXI are referred to as interfaces used for I/O. Specific HP VEE objects, such as the `Interface Event` object can only send commands to an interface.

Interface Driver
Software that allows a computer to communicate with a hardware interface, such as HP-IB or RS-232. Also called *device driver* in the UNIX operating system, interface drivers are configured into the kernel of the operating system.

Iterate
To repeat (loop) part of an HP VEE program using one of the `Repeat` objects (for example, `For Count`).

Library
A collection of often-used objects or programs grouped together for easy access.

Line
A link between two objects in HP VEE that transmits data containers to be processed. See also "Subthread" and "Thread."

Loop
To repeat part of an HP VEE program using one of the `Repeat` objects (for example, `For Count`).

Main Menu
The menus located in the HP VEE menu bar. To open a main menu, click on the appropriate menu title in the menu bar. You can also use accelerators. For example, on the PC use (Alt)+(F) to open the `File` menu. On an HP Series 300/700 workstation, use (Extend Char)+(F) to do the same thing.

Main Work Area
The area where you create a program. The main work area is the parent context of all other contexts.

Mapping
A set of continuous or discrete values that express the independent variables for an array (for example, the time span of a waveform).

Maximize
To enlarge a `UserObject` or a window to fill the available space using a maximize button. For a `UserObject`, this resizes the `UserObject` to occupy all of the HP VEE work area.

Maximize Button

A button on a `UserObject`, or the HP VEE window, that makes the `UserObject`, or the HP VEE window, occupy all of the available screen space.

Menu

A collection of features that are presented in a list. See also "Cascading Menu," "Main Menu," "Object Menu," "Pop-Up Menu," and "Pull-Down Menu."

Menu Bar

The bar at the top of the HP VEE window that displays the titles of the pull-down, main menus, from which you select features.

Menu Title

The name of a menu within the HP VEE menu bar. For example, `File` or `Edit`.

Minimize

1. To reduce an open view of an object to its smallest size—an icon.
2. To reduce a window to its smallest size—an icon.

Minimize Button

A button on an object, or the HP VEE window, that iconifies the object, or the HP VEE window.

Mouse

A pointing device that you move across a surface to move a pointer within the HP VEE window.

Mouse Button

One of the buttons on a mouse that you can click or double-click to perform a particular action with the corresponding pointer in the HP VEE window.

Network

A group of computers and peripherals linked together to allow the sharing of data and work loads.

Object

A graphical representation of an element in a program, such as an instrument, control, display, or mathematical operator. An object is placed in the work area and connected to other objects to create a program.

Object Menu
The menu associated with an object that contains features that operate on the object (for example, moving, sizing, copying, and deleting the object). To obtain the object menu, click on the object menu button at the upper-left corner of the object, or click the right mouse button with the pointer over the object.

Object Menu Button
The button at the upper-left corner of an open view object, which displays the object menu when you click on it.

Open
To start an action or begin working with a text, data, or graphics file. When you select `Open` from HP VEE, a program is loaded into the work area.

Open View
The representation of an HP VEE object that is more detailed than an icon. Most object open views have fields that allow you to modify the operation of the object.

Operate
The action of an object processing data and outputting a result. An object operates when its data and sequence input pins have been activated. See "Activate."

Operator Interface
The interface that the HP VEE programmer creates to allow the operator (end-user) to control the program. A typical operator interface involves panel views and dialog boxes.

Outline Box
A box that represents the outer edges of an object or set of objects and indicates where the object(s) will be placed in the work area.

Palette
The set of possible colors available in HP VEE.

Panel View

The view of an HP VEE program, or of a `UserObject`, that shows only those objects needed for the user to run the program and view the resulting data. You can use panel views to create an operator interface for your program.

Pin

An external connection point on an object to which you can attach a line.

Pointer

The graphical image that maps to the movement of the mouse. The pointer allows you to make selections and provides you feedback on a particular process underway. HP VEE has pointers of different shapes that correspond to process modes, such as an arrow, crosshairs, and hourglass.

Pop-Up Menu

A menu that is raised by clicking the right mouse button. For example, you can raise the `Edit` menu by clicking the right mouse button in an empty area within the work area. Or you can raise the object menu by clicking the right mouse button on an inactive area of an object.

PostRun

The set of actions that are performed when the program is stopped.

Preferences

Preferences are attributes of the HP VEE environment that you can change using `File` ⟹ `Edit Default Preferences`. For example, you can change the default colors, fonts, and number format.

PreRun

The set of actions that resets the program and checks for errors before the program starts to run.

Program

In HP VEE, a graphical program that consists of a set of objects connected with lines. The program typically represents a solution to an engineering problem.

Promote

To convert from a data type that contains less information to one that contains more information. See also “Data Type” and “Demote.”

Propagation

The rules that objects and programs follow when they operate or run. See also "Data Flow."

Properties

Object properties are attributes of HP VEE objects that you can change using *object menu* $\Longrightarrow$ `Edit Properties`. Work area properties are attributes of the HP VEE work area that you can change using `File` $\Longrightarrow$ `Edit Properties`. Properties include colors, fonts, and titles.

Pterodactyl

Any of various extinct flying reptiles of the order Pterosauria of the Jurassic and Cretaceous periods. Pterodactyl are characterized by wings consisting of a flap of skin supported by the very long fourth digit on each front leg.

Pull-Down Menu

A menu that is pulled down from the menu bar when you position the pointer over a menu title and click the left mouse button.

Radio Button

A diamond-shaped button in HP VEE dialog boxes that allows you to select a setting that is mutually exclusive with other radio buttons in that dialog box. To select a setting, click on the radio button. To remove the setting, click on another radio button in the same dialog box.

Record

An HP VEE data type that has named data fields which can contain multiple values. Records are typically used to group related articles of information. However, each record field can contain a different data type. Each field can contain another Record container, a Scalar, or an Array. The Record data type has the highest precedence of all HP VEE data types. However, data cannot be converted to and from the Record data type through the automatic promotion/demotion process. Records must be built/unbuilt using the `Build Record` and `UnBuild Record` objects.

Remote Function

A UserFunction running on a remote host computer, which is callable from the local host. (Remote functions are supported only on UNIX systems.) The `Import Library` object starts the process on the remote host and loads the Remote File into the HP VEE process on the local host. You can then call the Remote Function with the `Call Function` object, or from certain expressions. See also “UserFunction” and “Compiled Function.”

Restore
To return a minimized window or an icon to its full size as a window or open view by double-clicking on it.

Run
To start the objects on a program or thread operating.

Save
To write a file to a storage device, such as a hard disk.

Scalar
A data shape that contains a single value. See also “Data Shape.”

Schema
The structure or framework used to define a data record. This includes each field’s name, type, shape (and dimension sizes), and mapping.

Screen Dump
A graphical printout of a window or part of a window.

Scroll
The act of using a scroll bar either to move through a list of data files or other choices in a dialog box, or to pan the work area.

Scroll Arrow
An arrow that, when clicked on, scrolls through a list of data files or other choices in a dialog box, or moves the work area.

Scroll Bar
A rectangular bar that, when dragged, scrolls through a list of data files or other choices in a dialog box, or moves the work area.

Select
To choose an object, an action to be performed, or a menu item. Usually you select by clicking with your mouse.

Select Code
A number used to identify the logical address of a hardware interface. For example, the factory default select code for most HP-IB interfaces is 7.

Selection

1. A menu selection (feature).
2. An object or action you have selected in the HP VEE window.

Selection Field

A field in an object or dialog box that allows you to select choices from a drop-down list.

Sequence Input Pin

The *top* pin of an object. When connected, execution of the object is held off until the pin receives a container (is "pinged").

Sequence Output Pin

The *bottom* pin of an object. When connected, this output pin is activated when the object and all data propagation from that object finishes executing.

Sequencer

An object that controls execution flow through a series of sequence transactions, each of which may call a "UserFunction," "Compiled Function," or "Remote Function." The sequencer is normally used to perform a series of tests by specifying a series of sequence transactions.

Shared Library

A collection of functions, written in a programming language such as C, that can be called from HP VEE running on a UNIX system. Shared libraries can be created by experienced programmers. Shared libraries in the UNIX environment are similar to DLLs in the Windows environment.

Shell

In a UNIX system, the program that interfaces between the user and the operating system.

Shell Prompt

In a UNIX system, the character or characters that denote the place where you type commands while at the operating system shell level. The prompt you see displayed (for example, $) depends upon the type of shell you are running.

Sleep

An object sleeps during execution when it is waiting for an operation or time interval to complete, or for an event to occur. A sleeping object will allow other parallel threads to run concurrently. Once the event, time interval, or operation occurs, the object will execute, allowing execution to continue.

Startup Directory
The directory from which you start HP VEE on a UNIX system. This directory determines the default paths for most file actions including `Save` and `Open`. In HP VEE for Windows, this is referred to as the "working directory."

State
A particular set of values for all of the components related to an HP VEE instrument panel, which represents the measurement state of an instrument. For example, a digital multimeter uses one state for high-speed voltage readings and a different state for high-precision resistance measurements. See also "Instrument Panel."

Status Field
A field displaying information that cannot be edited. A status field looks like an entry field, but has a gray background.

Step
The action of operating an HP VEE program one object at a time (to debug the program). The object that will operate next is indicated by a green highlight.

Terminal
The internal representation of a pin that displays information about the pin and the data container held by the pin. Double-click on a terminal to view the container information.

Terminal Area
The areas on the left and right sides of an object where terminals are displayed when `Show Terminals` is active for that object. The input terminal area is on the left, and the output terminal area is on the right side of an object.

Thread
A set of objects connected by solid lines in an HP VEE program. A program with multiple threads can run all threads simultaneously.

Title Bar

The rectangular bar at the top of the open view of an object or window, which shows the title of the object or window. You can turn off an object title bar using *object menu* $\Longrightarrow$ `Edit Properties`.

Tool Bar

The rectangular bar at the top of the HP VEE window which provides the `Run`, `Stop`, `Cont`, and `Step` buttons to control HP VEE programs. The tool bar also displays the title of a program, and the `Panel` and `Detail` buttons if present.

Transaction

The specifications for input and output (I/O) used by certain objects in HP VEE. These include the `To File`, `From File`, `Direct I/O`, and `Sequencer` objects. Transactions appear as phrases listed in the open view of these objects.

Trig Mode

The `Trig Mode` is an attribute that determines whether trigonometric values are displayed in degrees, radians, or gradians. Note that HP VEE automatically converts trigonometric values to radians for calculation purposes.

User-Defined Function

A function that you can create, and then call in an HP VEE program. You can create three types of user-defined functions that can be called using the `Call Function` object, or from certain expressions. See also “UserFunction,” “Compiled Function,” and “Remote Function.”

UserFunction

A user-defined function created from a “UserObject” by executing `Make UserFunction`. The UserFunction exists in the background of the HP VEE process, but provides the same functionality as the original UserObject. You can call a UserFunction with the `Call Function` object, or from certain expressions. A UserFunction can be created and called locally, or it can be saved in a library and imported into an HP VEE program with `Import Library`. See also “Compiled Function,” “Remote Function,” and “UserObject.”

UserObject

An object that can encapsulate a group of objects to perform a particular purpose within a program. A UserObject allows you to use top-down design techniques when building a program, and to build user-defined objects that can be saved in a library and reused.

View
See "Detail View," "Icon," "Open View," and "Panel View."

Wait
See Sleep.

Window
A rectangular area on the screen that contains a particular application program, such as HP VEE.

Work Area
The area within the HP VEE window or the open view of a `UserObject` where you group objects together. When you `Open` a program, it is loaded into the main work area.

Working Directory
The directory in which HP VEE for Windows runs (`C:\VEE` is the default). On UNIX systems this corresponds to the "startup directory."

X Window System (X11)
An industry-standard windowing system used on UNIX computer systems.

X11 Resources
A file or set of files that define your X11 environment in a UNIX system.

XEQ Pin
A pin that forces the operation of the object, even if the data or sequence input pins have not been activated. See also "Control Pin," "Data Input Pin," and "Sequence Input Pin."

Index

Index