Visual Engineering Environment

An Introduction to HP VEE-Test

Customer Training Course

HP Part No. E2110+24D
Printed in USA September 1991

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard Company (HP) shall not be liable for any errors contained in this document. HP MAKES NO WARRANTIES OF ANY KIND WITH REGARD TO THIS DOCUMENT, WHETHER EXPRESS OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory, in connection with the furnishing of this document or the use of the information in this document.

Warranty Information

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013.

Use of this manual and magnetic media supplied for this product are restricted. Additional copies of the software can be made for security and backup purposes only. Resale of the software in its present form or with alterations is expressly prohibited.

Printing History

September 1991 - First Edition

0

Welcome to HP VEE

Class Overview

Welcome to
Introduction to
HP VEE

Number E2100+24D

E2100+24D V001

Welcome to *An Introduction to HP VEE*. In this class you'll learn to use the many capabilities of Hewlett Packard's visual engineering environment products.

This chapter contains a general agenda and schedule for the class. Note that if you are taking the HP VEE-Engine Class, that the class concludes at the end of Day Three.

Agenda & Schedule

HP VEE CLASS—DAY ONE

Time	Activity
8:30	Welcome & Orientation
9:00	LAB 1: *Getting Started with HP VEE*, chapter 2
9:50	BREAK
10:00	Module 1: Principles of Operation
10:50	BREAK
11:00	Module 2: Using HP VEE Objects
12:00	LUNCH
1:00	LAB 2: Using Objects
1:50	BREAK
2:00	Module 3: Virtual Devices
2:50	BREAK
3:00	LAB 3: Waveforms
3:50	BREAK
4:00	ADDITIONAL LAB TIME
5:00	Adjourn

HP VEE CLASS—DAY TWO

Time	Activity
9:00	Module 4: UserObjects
9:50	BREAK
10:00	LAB 4: UserObjects
10:50	BREAK
11:00	Module 5: User Interaction
12:00	LUNCH
1:00	LAB 5: Custom Dialog Box
1:50	BREAK
2:00	Module 6: Application Development Techniques
2:50	BREAK
3:00	LAB 6: Programming Techniques
4:00	ADDITIONAL LAB TIME
5:00	Adjourn

HP VEE CLASS—DAY THREE

Time	Activity
9:00	Module 7: I/O Transactions and Data Formatting
9:30	Module 8: Files and Standard I/O
9:50	BREAK
10:00	LAB 7: Files
10:50	BREAK
11:00	LAB 7: Files (cont.)
12:00	LUNCH
1:00	Module 9: HP-UX Escapes
1:50	BREAK
2:00	LAB 8: HP-UX Escapes
2:50	BREAK
3:00	Module 10: Install, Configure & Customize
4:00	ADDITIONAL LAB TIME/Wrap up & Evaluation (Engine Only)
5:00	Adjourn

HP VEE CLASS—DAY FOUR (HP VEE-Test Only)

Time	Activity
9:00	Module 11: Instrument Control Interfaces
9:50	BREAK
10:00	Module 12: Using Drivers for Instrument Control
10:50	BREAK
11:00	LAB 9: Measuring Frequency Response
12:00	LUNCH
1:00	LAB 9: Measuring Frequency Response (cont.)
1:50	BREAK
2:00	Module 13: Using Direct Device I/O
2:50	BREAK
3:00	LAB 10: Instrument Control
3:50	BREAK
4:00	ADDITIONAL LAB TIME
5:00	Adjourn

HP VEE CLASS—DAY FIVE (HP VEE-Test Only)

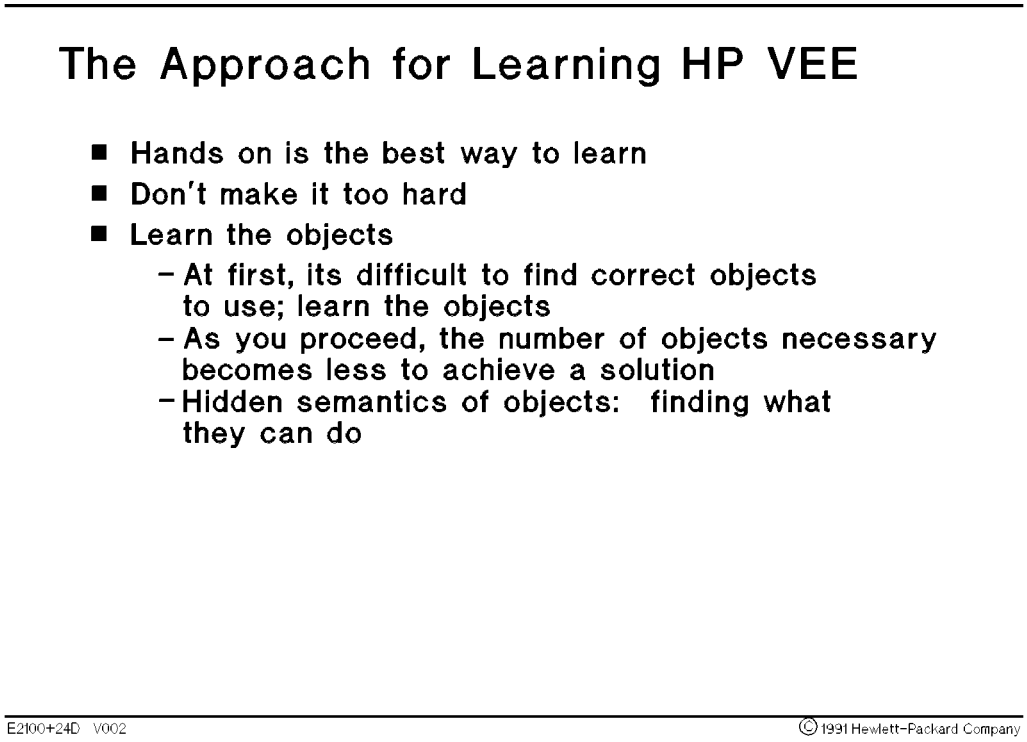
Time	Activity
9:00	Module 14: HP-UX Escapes, Named Pipes & BASIC/UX
9:50	BREAK
10:00	LAB 11: HP-UX Escapes & Named Pipes
10:50	BREAK
11:00	LAB 12: Benchmarking HP BASIC/UX Escapes
12:00	LUNCH
1:00	Module 15: Instrumentation Application Development Techniques
1:50	BREAK
2:00	LAB 13: Develop Your Application
2:50	BREAK
3:00	Wrap-Up & Evaluations
4:00	Adjourn

Contents

1

Principles of Operation

How to Learn HP VEE



The best way to learn HP VEE is experience. Think of this as an adventure game, discovering a new way to think about and use computers. It will be fun. At first it may be uncomfortable to try to think in a new way. But you will quickly find that HP VEE makes you much more productive. Don't worry if you can't find each object quickly. You'll soon learn the menus. You'll be surprised at how fast you arrive at solutions.

Lab Exercise 1 *Getting Started Guide*

Task

To familiarize yourself with the fundamental operations of HP VEE, open the *Getting Started with HP VEE* to chapter 2, "Interacting with HP VEE". Perform the exercises that it describes.

VEE Operation Fundamentals

- Synchronous Operation
- Propagation Rules
- Multiple Threads

E2100+24D V003

In this module we'll discuss the fundamental principles of operation used by HP VEE. We'll cover

- Synchronous Operation, how the HP VEE objects function
- Propogation Rules, how HP VEE decides which object operates next
- Multiple Threads, how multiple sets of objects relate and operate

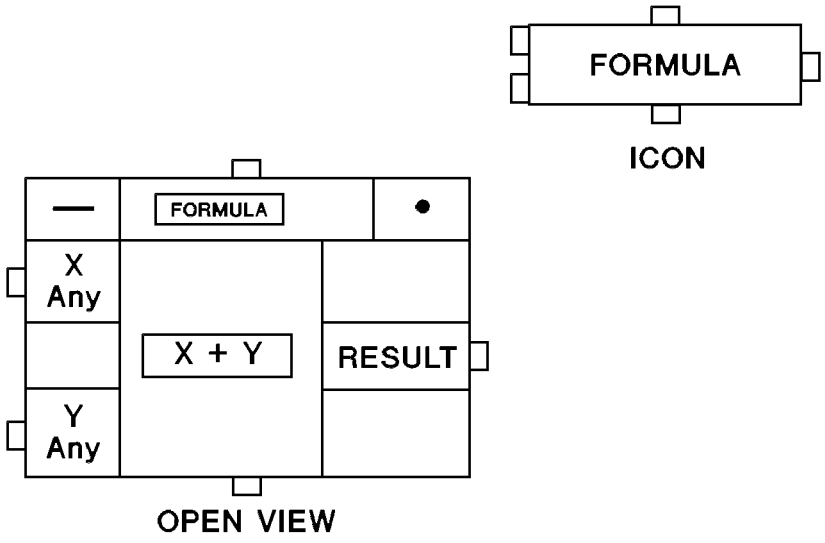
Useful Definitions For HP VEE

■ Model	The finished solutions with objects linked together (not a program – a model)
■ Work Area	"the executable block diagram" The area within the HP VEE window in which you build models
■ Object	Any item placed on the work area
■ Icon	A small, graphical representation of an object
■ Open View	The maximized view of an object

E2100+24D V004

First, lets define some words commonly used with HP VEE.

Icon vs Open View



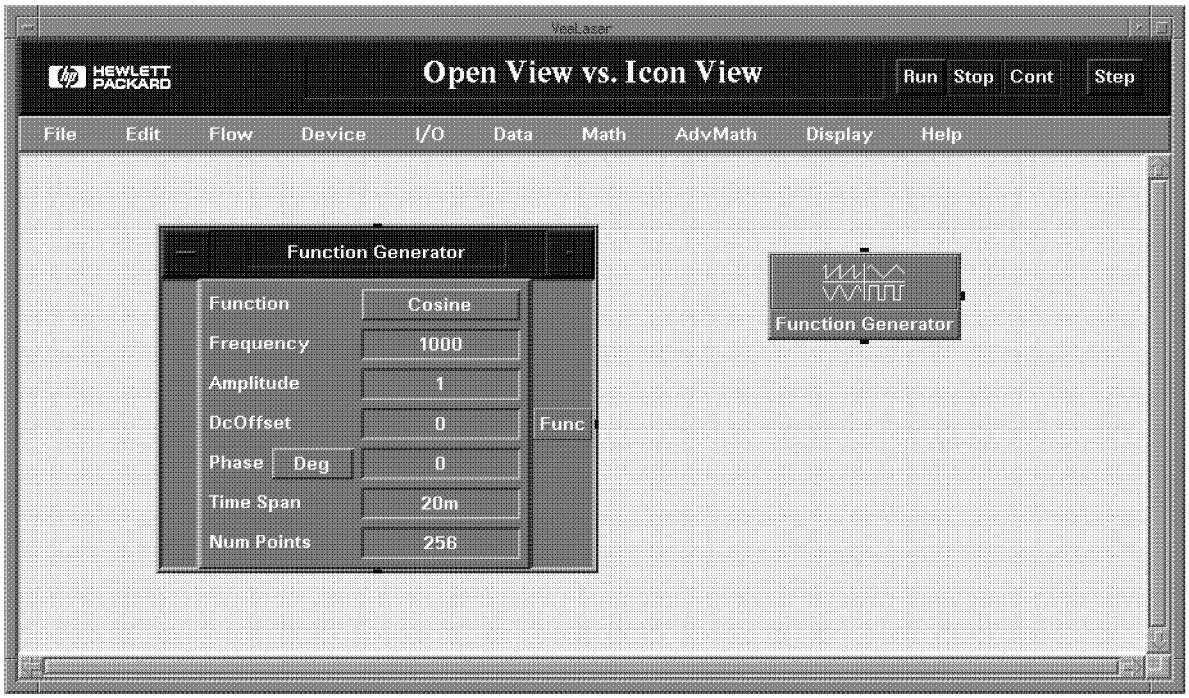
E2100+24D V005

The figure on the lower left-hand corner illustrates an "open view" of an object. In this mode, you can see the internal operation, terminals, formulas or controls that function within the object. You must be in this mode to change or modify the way an object works.

The figure in the upper right-hand corner illustrates an "icon". This mode minimizes the size of the object, saving display space. You use this mode when you have completed editing the object, or when there is no need for the user to interact with the object.

The screenshot on the following page illustrates an actual object in Open View and Icon View.

Screenshot openicon



This screenshot shows a `Function Generator` object in Open View and Icon View.

More Useful Definitions For HP VEE

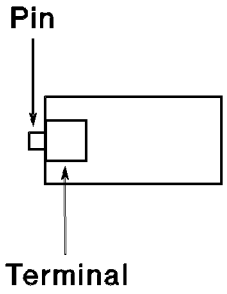
- **Thread** — A set of objects connected by solid lines
- **Activate** — Send data or sequence instructions to a terminal or pin
- **Operate** — To activate an object
- **Ping** — A message that is used to initiate execution
- **Container** — The package that is transmitted over lines and is processed by objects

E2100+24D V007

Lets define some more words commonly used with HP VEE.

Pins & Terminals

- Data Input/Output
- Sequence Input/Output
- Asynchronous Control Input
- Error Output

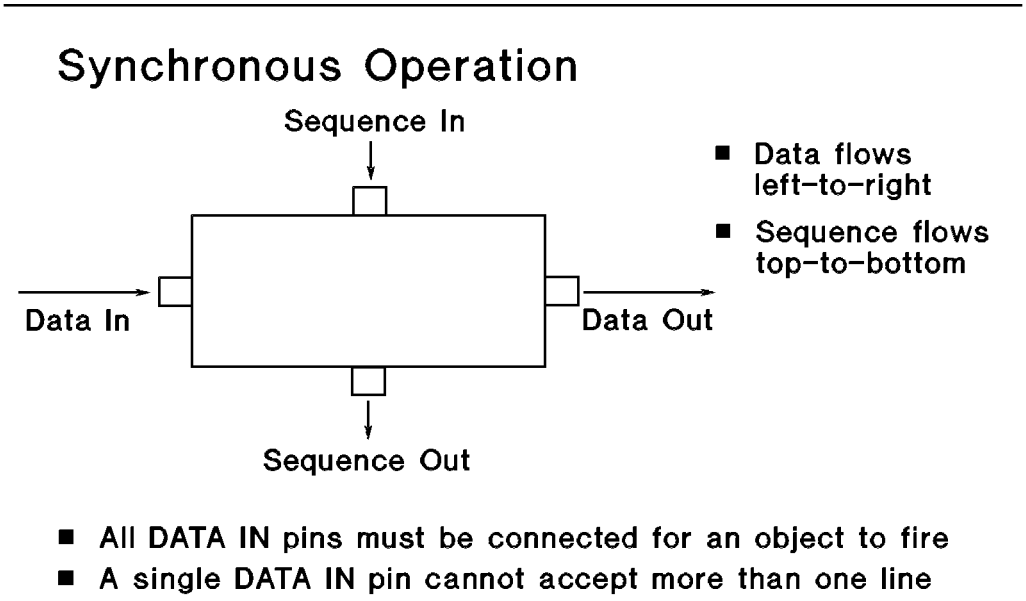


E2100+24D V008

Pins and terminals provide input and output connections for data and control on HP VEE objects.

Pins provide connecting points between objects. Data pins receive and send data. Sequence pins control the execution flow of objects. You do not have to connect sequence pins for the object to operate. Asynchronous pins control the operation of the object. The action they control activates immediately. It doesn't wait for the object to complete its operation. The Error output pin sends error information and allows you to trap errors.

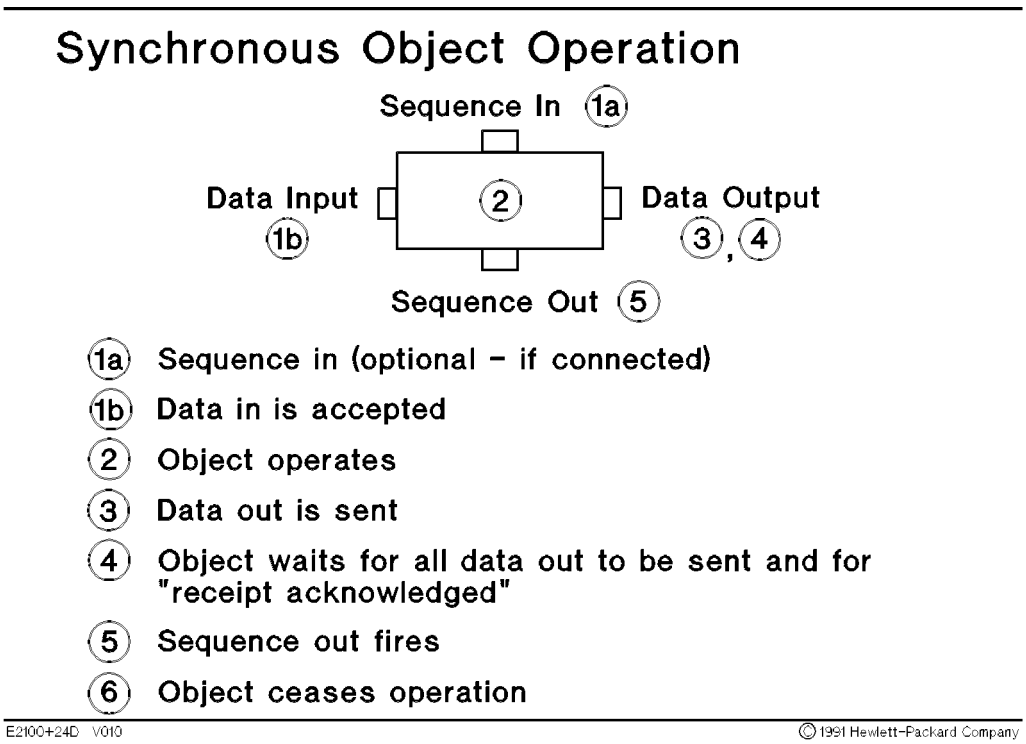
Terminals display data information for each pin. Note that sequence pins do not have a corresponding terminal.



All HP VEE objects operate in the same manner. Data flows from left to right through the object. Sequence flows from top to bottom.

All data input pins must be connected to another object. A data input pin cannot accept more than one input connection. Data output pins may connect to one or several objects.

Data output and sequence in or out pins do not need to be connected.

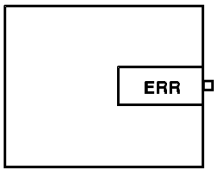


HP VEE objects operate in the follwing manner:

1. The object receives data input. Sequence input is received (if connected).
2. The object operates, performing its designated function.
3. The object sends data out to the next object.
4. The object waits for the next object in the thread to complete operation and send a "receipt acknowledged" back to it. This insures that all objects in a thread operate before HP VEE executes the next thread or subthread.
5. The object activates its sequence out pin.
6. The object ceases operating.

Optional Object Connections

- Data In, Data Out
  - Many objects allow additional data in, data out terminals
- Control Input
  - Ping causes immediate execution of object sub-function
  - Is not required for overall object execution
  - Examples: (Clear, Autoscale X, etc.)
- Error Output
  - Overrides standard object behavior
  - Activates when error occurs during object execution
  - Activates INSTEAD OF data outputs



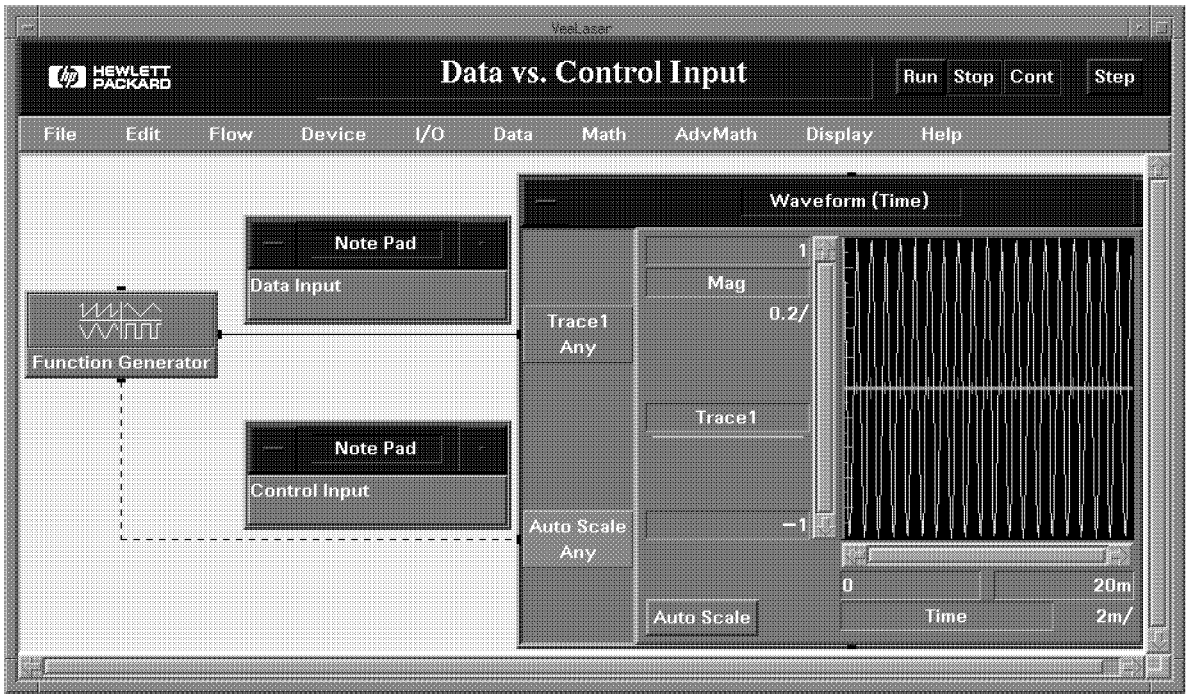
E2100+24D V011

Each object starts with a set of default input and output pins. Many devices allow additional input and output data pins as well as control and error pins.

As stated earlier, the control pins cause immediate execution of an object function, such as `clear`, `autoscale`, etc. The object does not need to receive input on a control pin in order to execute. However, it must receive input on each data pin in order to execute.

The error pin overides standard object behavior in the event of an error. It sends out an error message when an error occurs. The object does not send out data on its data pins when an error occurs.

Screenshot datactlin



Notice in this screenshot how HP VEE uses a dashed line to connect to Control inputs. This provides a visual reminder that these are control inputs and do not need to be satisfied for the object to operate.

Adding Optional Inputs

- **Object menu provides ability to add terminals to objects**
  - **Data, Control, Trigger (User Objects Only)**
  - **Inputs and Outputs**
- **Terminal can be opened to edit name**
  - **Type and shape can be modified if required**

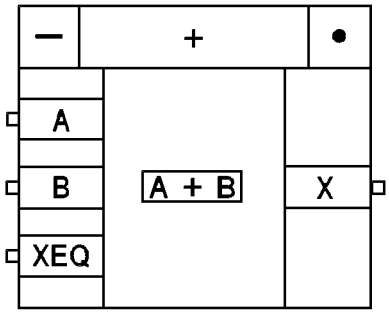
E2100+24D V013

The `Terminals` selection on the object menu allows you to add pins to an object. Selecting `Show Terminals` displays the terminal names on the object.

You can also open and edit terminal characteristics such as name, type and shape. Note that some characteristics are unalterable.

Overriding Constraints

- XEQ control causes immediate object operation
  - Available data used
- Useful for continuing after error
- Required by some data building objects



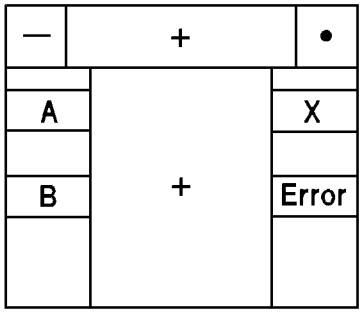
E2100+24D V014

The `XEQ` pin causes the object to operate immediately. The object uses whatever data is available on its input pins. This is useful when continuing after an error. Some data building objects require the use of this pin to "finish" their operation. Note that `XEQ` pins are only available on the following objects:

- `UserObject`
- `Confirm (Okay)`
- `Set Values`
- `Collector`

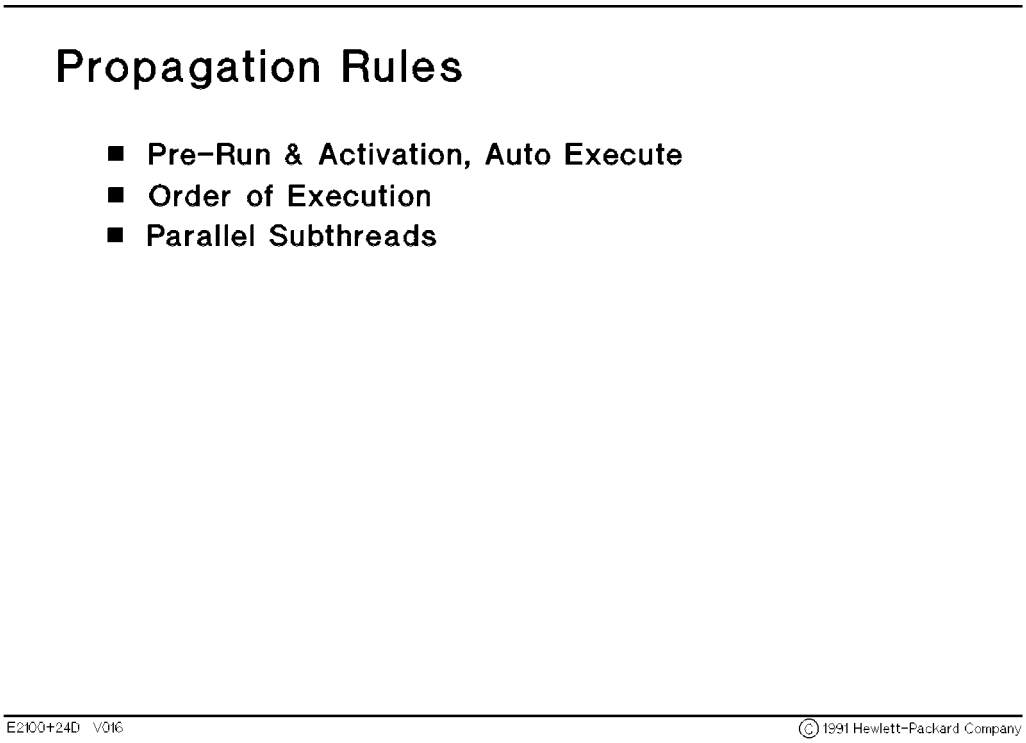
Trapping Errors

- Error output pin can be added
- Allows HP VEE to continue execution after error
  - Error pin activates
    Instead of data output pins
  - Output container holds error number



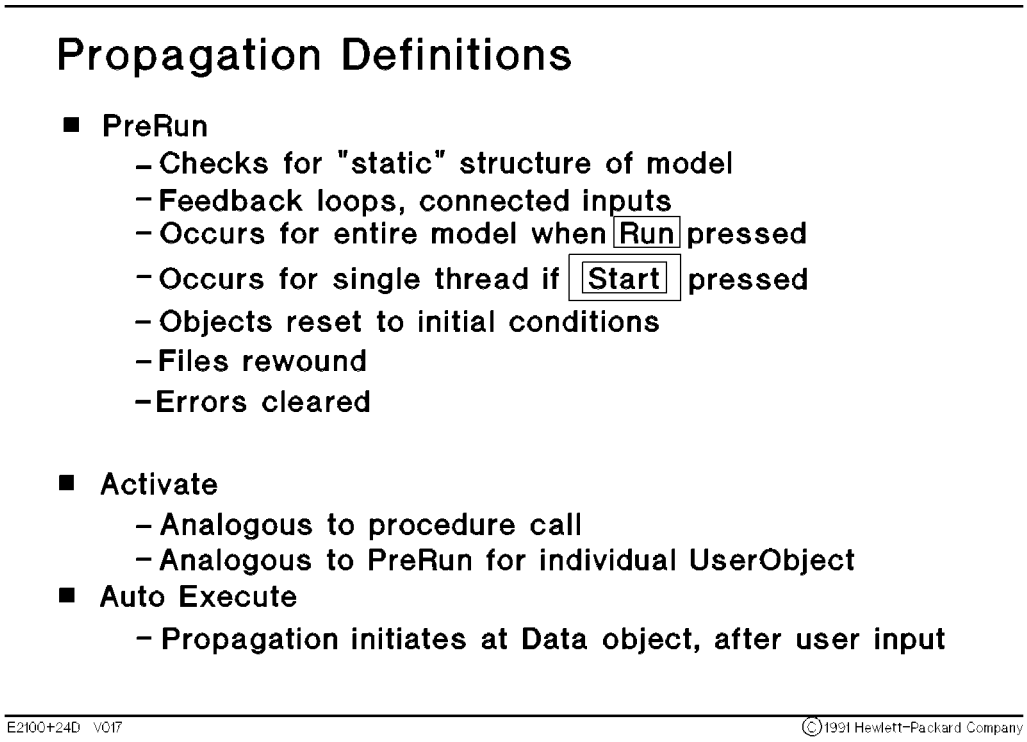
E2100+24D V015

To trap errors, add an error output pin to the object. When an error occurs the error pin activates, sending out an error number. Other objects can then operate on that information, correcting the error if possible. Note that when an error occurs, the data output pins do not send out any data.



Now lets take a look at the Rules of Propogation, or how HP VEE decides which object operates next. We'll also examine:

- PreRun, how HP VEE initializes objects and work areas.
- Activation, what happens when a work area begins operation.
- Auto Execute, where some objects can operate without pressing `Run`.



HP VEE performs a PreRun (the following operations) when you press a `Start` or `Run` button:

- Checks the structure of your model for proper construction and connectivity.
- Checks for Feedback loops, setting data input terminals on feedback loops to `nil`.
- Determines if all data and `XEQ` inputs are connected.
- Resets objects to initial conditions.
- Rewinds data files to their beginning.
- Clears errors.

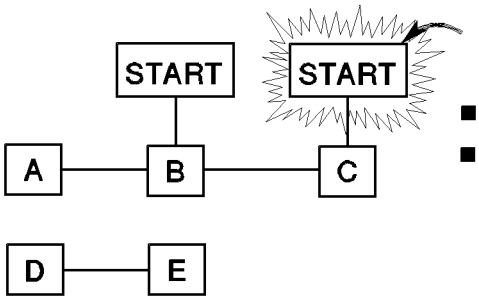
Note that HP VEE PreRuns you entire model when you press `Run`, but only an individual thread when you press `Start` on that thread.

Activation is analagous to a procedure call in a standard programming language. It initializes the state (PreRuns) of an individual `UserObject` each time the `UserObject` operates.

Auto Execute causes an object to propogate data after a user input, if the thread has completed running. It is not necessary to press `Start` or `Run`.

Start Objects

- Allow execution sequence to begin
- Affect only their own thread
- At Run time, all START objects on every thread operate prior to any other objects



- Initiates thread propagation
- D & E unaffected

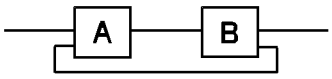
E2100+24D V018

`Start` objects permit exectution of an individual thread within a model. They affect only their thread, not the entire model. When you press `Run`, all `Start` objects operate.

In this example, pressing either `Start` button executes the thread A-B-C. Objects D and E are unaffected. Again, pressing `Run` causes all threads in a model to execute.

Start Object

- Never required EXCEPT to resolve FEEDBACK



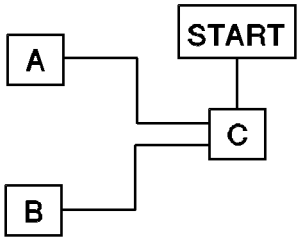
- A cannot operate until BOTH its data pins are pinged
- B cannot operate until A does
- At PreRun, dialog box will advise you to supply a Start

E2100+24D V019

`Start` objects are never required in a model except to resolve where to start a feedback loop. In the example shown here, `A` cannot execute until it receives data on *both* input pins. `B` cannot execute until it receives data on its input pin. By adding a `Start` object, you force HP VEE to begin execution with "old data" on one of the objects. Usually the old data is `nil` and interpreted properly for the object. The PreRun operation will advise you to add a `Start` object if it is necessary.

Propagation

- Unconstrained Objects
  - No input constraints
- Constrained Objects
  - Have either Data Input or Sequence Input pins connected



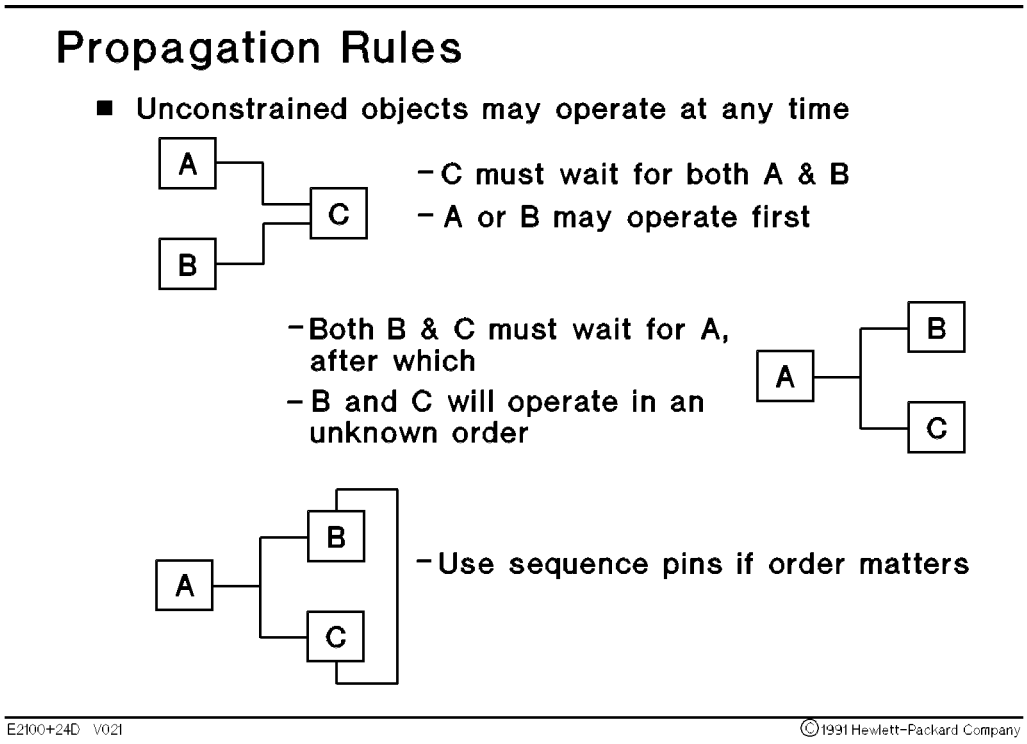
- A or B may operate anytime after Start

E2100+24D V020

HP VEE models execute in the following order:

- `Start` objects operate first.
- Unconstrained objects, objects with no data or sequence inputs, operate next.
- Constrained objects then operate when their input contraints are satisfied.

In this example, A or B may operate anytime after the `Start` object operates.



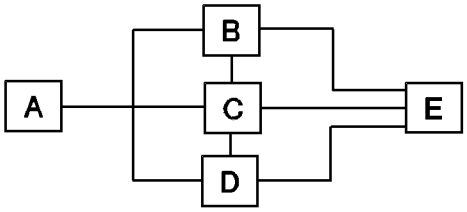
Just to be sure you understand propogation in HP VEE lets look at some more examples.

In the first example, `A` or `B` may execute first, but C must wait until both `A` and `B` have completed their operation.

In the next example, `A` executes first. `B` and `C` must wait until `A` finishes. Then `B` and `C` execute. You cannot determine which one will go first.

If the order in which `B` and `C` operate is important, use a sequence connection as shown in the bottom example. In this case, `B` must wait until `C` finishes.

Propagation Example



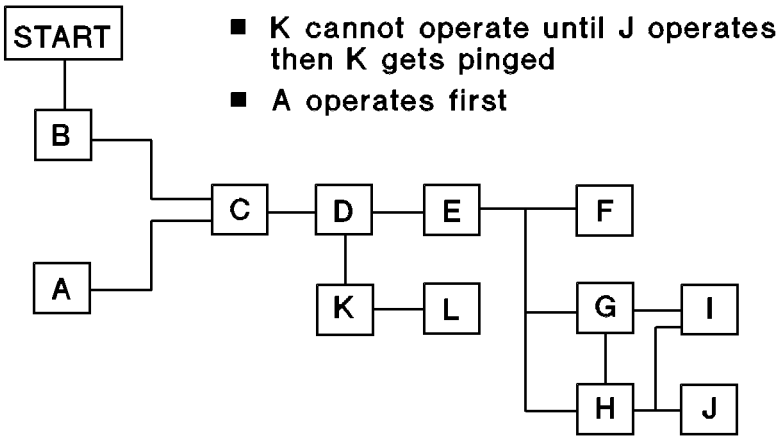
- Notice that the sequence lines order the correct execution of B, C, and D

EXECUTE DONE

E2100+24D V022

See if you can determine the execution order of the objects on this slide.

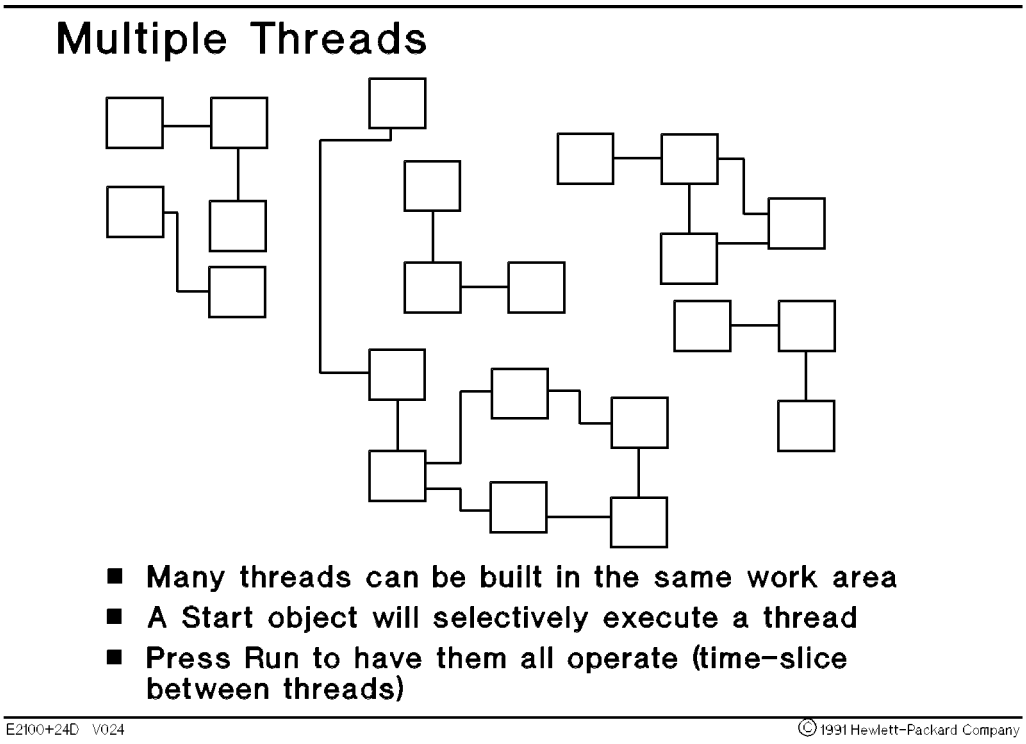
Propagation Example



E2100+24D V023

Here is a more complex example. Note that the objects operate in alphabetical order. Notice that `D` cannot activate its Sequence Out pin until as many objects as possible down-thread from it complete.

In the case of `G`, it activates its Sequence Out pin before `I` completes because `I` cannot complete until `H` operates, providing data for `I`.



You can build many threads in the same work area. A `Start` object will selectively execute an individual thread. Pressing `Run` executes all of the threads, which share the processor.

Multiple Thread Propagation

- Parallel threads are time-sliced by "propagation engine"
- Time slice ≡ 1 primitive object
  - Note:
    - Each object on an iterating subthread of a repeat device (iterator) counts as one timeslice
    - UserObjects are MULTIPLE objects

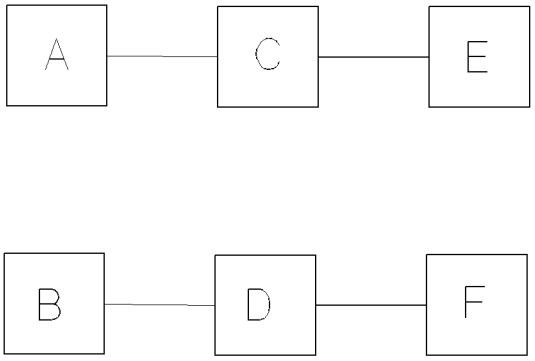
      Each object in a UserObject is a primitive object
- Parallel sub-threads also time-slice

E2100+24D V025 ©1991 Hewlett-Packard Company

When multiple threads exist, HP VEE shares the processor among the various threads. The time slice is the time it takes to execute a single primitive object.

Note that each object on an interating subthread executes for one time slice. Also note that HP VEE treats `UserObjects` as multiple objects. Each object within the `UserObject` operates for one time slice.

Parallel Subthread Example



E2100+24D V026

Here is an example of parallel thread propogation. Note that the objects execute according to the letter in each object. The propogation engine shares time between each thread by alternately executing an object from each.

Debugging

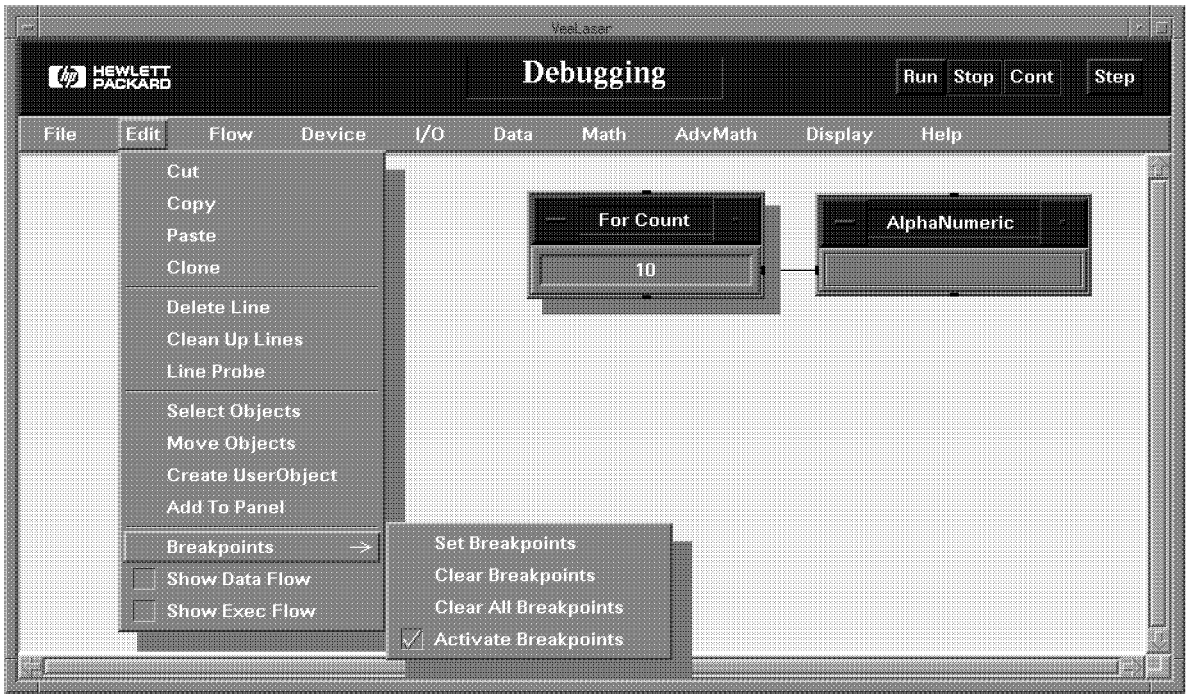
- **Show Execution Flow**
  - **Highlights each object on operation**
- **Show Data Flow**
  - **Shows data container moving along threads**
- **Set Breakpoints**
  - **Stops execution at this point**
- **Line Probe**
  - **Shows data container on thread**

E2100+24D V027

HP VEE provides several tools under the `Edit` menu to help you debug your models.

Show Exec Flow	Highlights the object that is currently executing. A highlighted border appears around the executing object.
Show Data Flow	Shows the route that data takes through the model. A small square marked moves along the lines connecting objects to show the movement of a data container.
Set Breakpoints	Stops execution of the model *before* it exectutes this object. An object with breakpoint set has a black border.
Line Probe	Displays the container information (data) transmitted on a line between two objects.

Screenshot debug



You find the debugging options on the `Edit` menu. `Show Data Flow` and `Show Exec Flow` are check boxes. Click on them to enable this function.

The `Breakpoints` menu cascades to provide additional options. Note that you must select an object before you can use some of these options.

Data Types & Structures

- Int16
- Int32
- Real
- Coord
- Enum
- String
- Complex
- PComplex
- Spectrum
- Waveform

E2100+24D V029

Now lets look at the various data types and structures implemented by HP VEE. You can see that we provide many useful data types. Lets look at each one individually.

INT16

- Signed 16 - Bit Integer
- Integer Numbers from -32768 to +32767
- Used only for I/O to files and instruments

E2100+24D V030

`Int16` is a 16-bit signed integer. It is used only for I/O to files and instruments. Otherwise, HP VEE uses the `Int32` data type for integers.

INT32

- Signed 32 - Bit Integer
- Integer Numbers from -2^{32} to $2^{32}-1$

E2100+24D V031

`Int32` is a 32-bit signed integer. HP VEE uses this data type for nearly every case of integers, except as noted before.

Real

- Real Numbers from -1E308 to +1E308
- IEEE 754 64-bit format
- Includes Date/Time REAL (seconds since midnight, January 1, 1 A.D.)

E2100+24D V032

HP VEE uses the IEEE 754 64-bit format to represent real numbers. It also uses this data type to represent time/date. It measures the number of seconds from 00:00 am (midnight), January 1, 1 A.D.

Coord

- (REAL x1, [REAL x2, ...] REAL y)
- Explicit n-dimensional data points
- One independent variable
- Multiple dependent variables

E2100+24D V033

The `Coord` data type provides a means to represent Coordinates in n-dimensional space. You may have a maximum of 10 dimensions in a `Coord`. Note that is useful to remember that this is one independent variable and multiple dependent variables. HP VEE will operate on the dependent variables with many of the math objects.

`Coord` must be either a scalar or an array of one dimension.

Enum

- {TEXT value, TEXT value, ...}
- Maps onto {0,1,2...}
- Use ordinal(x) to go from TEXT representation to positional representation
- An array of ENUM becomes TEXT

E2100+24D V034

The `Enum` data type gives you the ability to define your own enumerated data type. The values of your data map onto the range 0, 1, 2, ... n. Where n is the number of elements minus one. You can use the object `ordinal(x)` to convert from the `Text` representation to the positional representation. Note that an array of `Enum`s becomes a `Text` array.

Text

- **Arbitrary length of characters**

E2100+24D V035

A `Text` data type is simply an arbitrary number of characters.

Complex/PComplex

- (REAL real-part, REAL imag-part)
- (REAL magnitude, @REAL phase)
  - Phase can be degrees, radians or gradians

E2100+24D V036

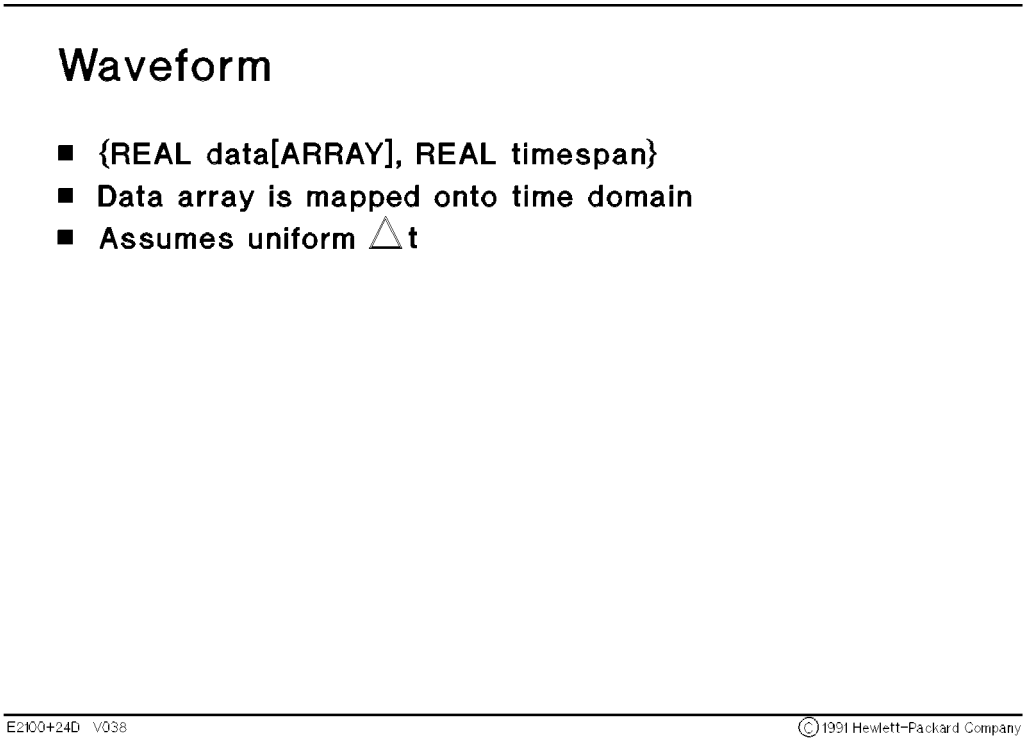
HP VEE uses the data types `Complex` and `PComplex` to represent complex numbers. `Complex` respresents the number using a rectangular coodinate system, while `PComplex` (Polor Complex) represents the number in a polar coodinate system. Note that phase (as well as all angles in HP VEE) can be in degrees, radians or gradians. The `Preferences --> Trig Mode` feature under the `File` menu allows you to select the angle mode.

Spectrum

- {PCOMPLEX data[ARRAY], REAL fstart, REAL fstop, ..}
- {PCOMPLEX data[ARRAY], REAL center, REAL span, ...}
- Data is mapped onto frequency domain
- Assumes uniform frequency (linear or logarithmic)

E2100+24D V037

The `Spectrum` data type represents waveform spectra. Note that the data is an array of `PComplex` data with either start and stop frequencies or center frequency and span. The data is mapped into the frequency domain and assumes a uniform frequency spacing.



The `Waveform` data type represents waveforms. The data is an array of `Real` numbers with a timespan. The data maps onto the time domain and assumes a uniform spacing between data points.

Automatic Data Typing

- Data containers have "data type" tag
- Many objects accept "any" data type
- Objects can generate many different data types
- Type conversion can happen to resolve dissimilar types:
  - Conversion to match input constraints
  - Promotion only so operands match
  - Conversion to match transactions

E2100+24D V039

In most computer programming languages you must convert data types of two operands before they can be operated on together. HP VEE peforms an automatic data type conversion for you, where possible.

Each data container has a data type tag. Most objects accept any data type so it is not necessary to make any conversion. However, occasionally you may want to operate on dissimilar data types. HP VEE will convert the data types to match the input constraints of the operation, or so that the operands match their data types.

Data Promotion & Demotion

- **Principles**
  - **When combining data types, lower is promoted whenever possible (for math operations)**
  - **Must be same shape (on terminals)**
- **Data Loss**
  - **When device expects a fixed data type, "higher" types may lose information to conform**

E2100+24D V040

You can consider data type conversion as simply a data promotion to a “higher” or more complex data type, or a demotion to a “lower” or less complex type.

When converting data types for formula operations, HP VEE promotes whenever possible. Note that both data types must have the same shape, if on a terminal.

When data types are demoted, “higher” data types may lose information. For example, when converting a complex number to a real number, you would lose information.

Data Promotion & Demotion

FROM TYPE \ TO TYPE	Int16	Int32	Real	Complex	P-Complex	Wave-form	Spectrum	Coord	Enum	String
Int16	Y	Y	Y	*	*			*		Y
Int32	*	Y	Y	*	*			*		Y
Real	*	*	Y	*	*			*		Y
Complex				Y	Y					Y
PComplex				Y	Y					Y
Waveform	*	*	*			Y	*	Y		Y
Spectrum				*	*	*	Y			Y
Coord								Y		Y
Enum									Y	Y
String	*	*	*	*	*			*		Y

E2100+24D V041

This table shows which data types can be converted to other data types. Where the box contains a "Y", data can be converted without data loss. Where the box contains a "*", data can be converted with the possibility of data loss. Refer to the *Using HP VEE* manual for more information on data loss. Where the box is empty, no conversion can take place.

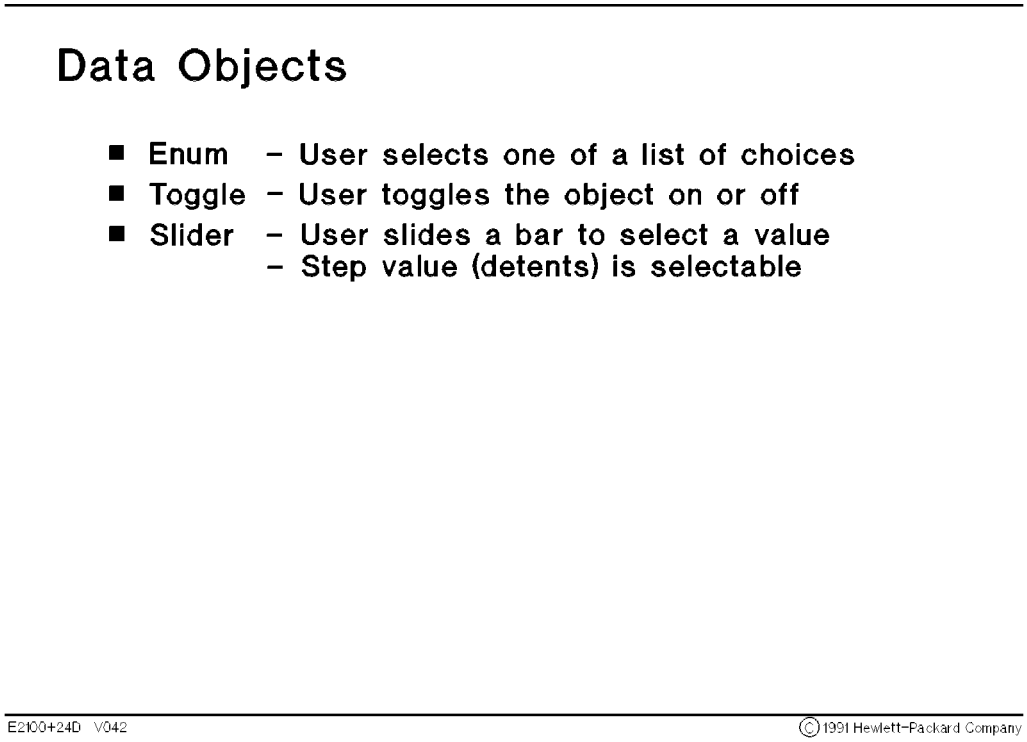
Refer to pages 3-24 and 3-25 in *Using HP VEE* for details on data promotion and demotion.

Note that the `Int16` data type is only used for I/O. You will never use it within your models.

2

Using HP VEE Objects

HP VEE Data Objects



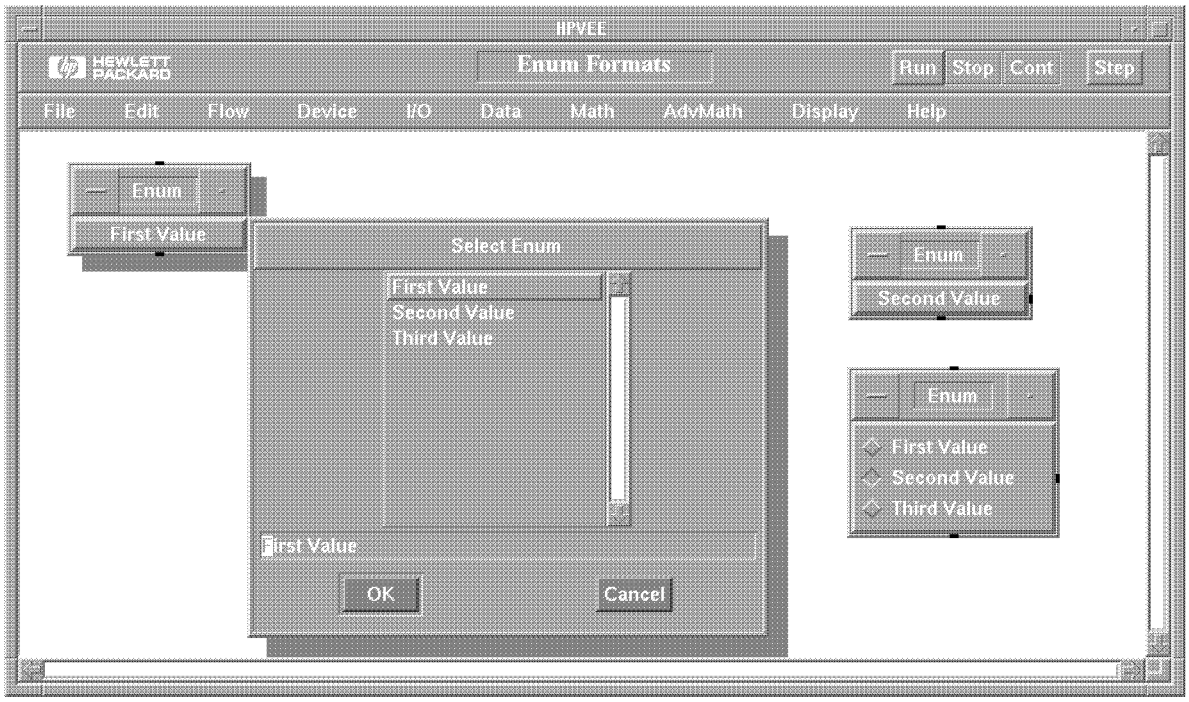
HP VEE provides several data objects to generate data and constants.

The `Enum` object permits you to create an enumerated list of items. The user then chooses one of the items from the list. This can be done by choosing from a list of items, cycling through the items or pushing a radio button.

The `Toggle` object allows the user to toggle the objects output on or off.

The `Slider` objects (Real or Integer) lets the user select a data value by sliding up and down a scale. You can define the step values (detents) between data points on the slider.

Slide—Screen Shot of Various Enum Formats



In this screen shot you can see the three different Enum object formats, list, cyclical and radio button.

Constant Types

Allows the user to type in values to define the fields

- **Text**
- **Integer**
- **Real**
- **Coord**
- **Complex**
- **PComplex**
- **Date/Time**

Evaluates simple calculations (no variables)

E2100+24D V044

Data constant objects permit you to define a constant. Each of these objects outputs a data container as described by its name. You may modify the shape of this data to be arrays.

You can also perform simple calculations within these constants, such as `10*3` or `pi*2`. However, you cannot use variables within these constant calculations.

Data Objects – Unique Capabilities

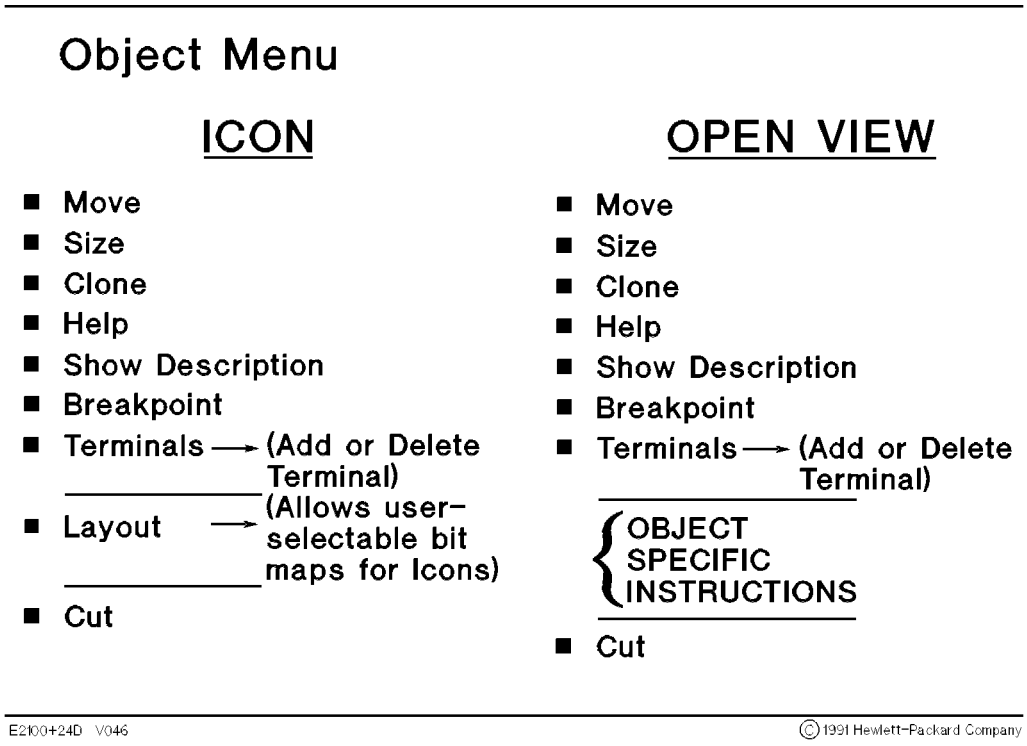
- **Initial Value (Initialize at PreRun, Activate)**
- **Auto Execute**
- **Config (Array Size)**
- **Set Number Formats**
- **Set Object Format (Enum: List, Cyclic, Buttons)**
- **Edit Values**

Many of these capabilities can be added as Control Inputs

E2100+24D V045

Data object have some unique capabilities.

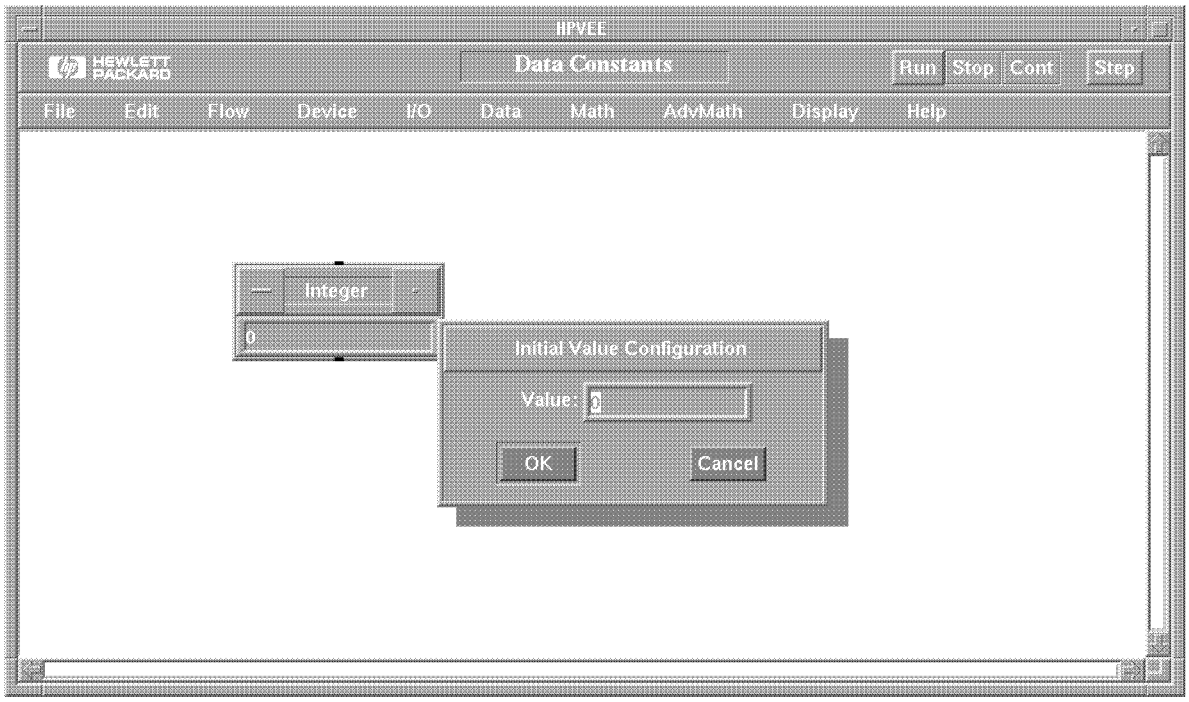
`Initial Value`	A value assigned to the object at PreRun or Activate.
`Auto Execute`	The object sends data when the user interacts with it. Does not require `Start` or `Run`.
`Config`	Define Array size (or Scalar) for a data object.
`Number Formats`	Define number format for this object. (Binary, Octal, Hex, etc.)
`Object Format`	Define format of object. For example, define the format of the `Enum` object, cyclical, list or radio button.
`Edit Values`	Edit the user-defined values of an object.



The object menu allows you to interract with an object. Note that it is slightly different between the icon view and the open view. With this menu you can:

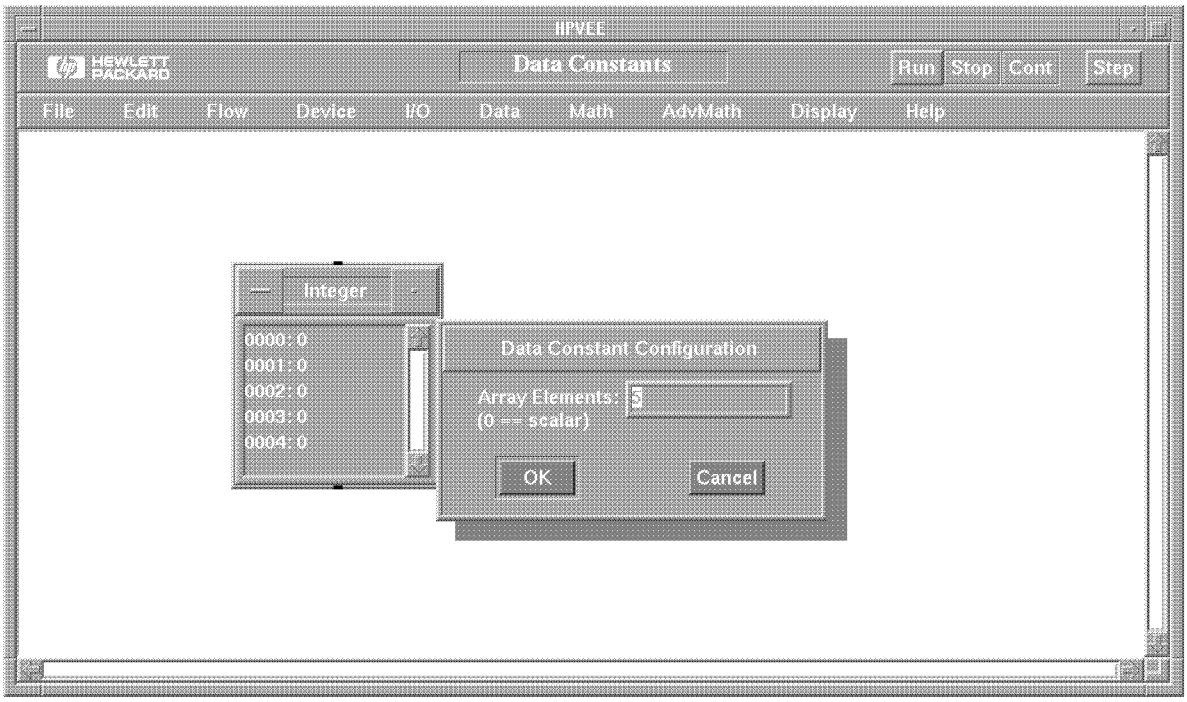
- Change the size of the object
- Move it
- Clone it
- Set and clear breakpoints
- Show the user comments about the object (Show Description)
- Add Pins and Terminals
- Change the object layout (such as adding a user-selectable bitmap)
- Perform other object specific operations
- Delete the object

Slide—Screen Shot of Data Constants



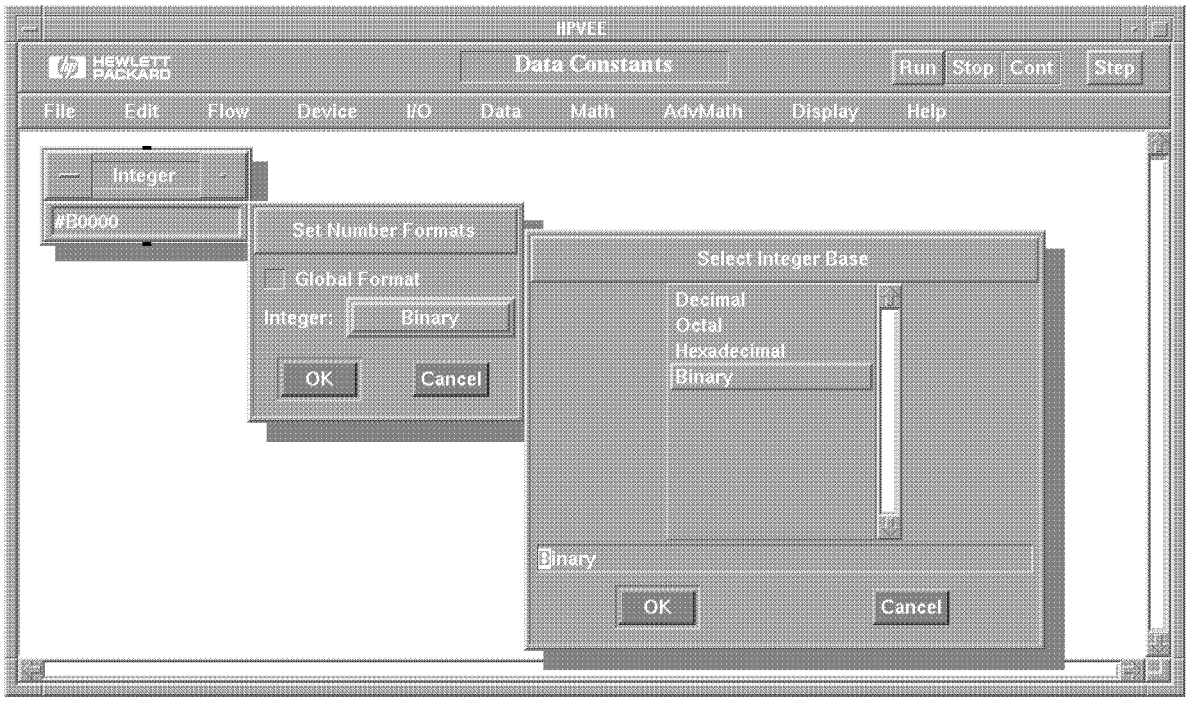
This screen shot illustrates using the Initial Value Configuration feature.

Slide—Screen Shot of Data Constants



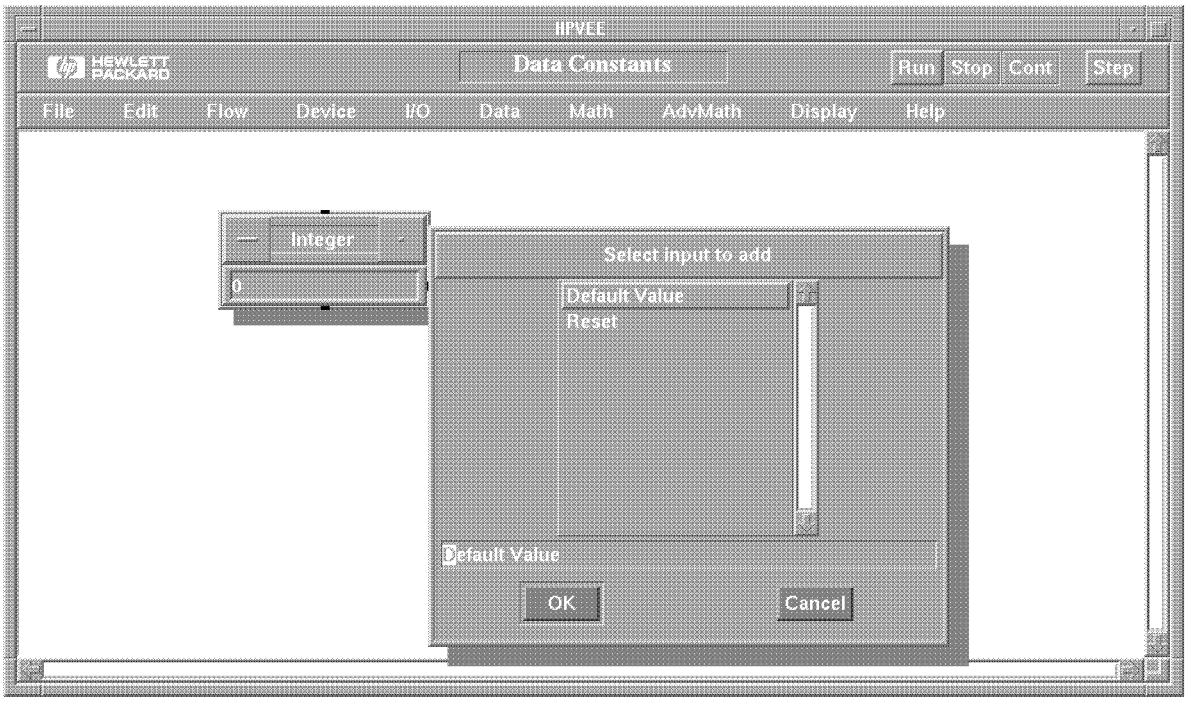
This screen shot illustrates defining an integer array constant.

Slide—Screen Shot of Data Constants



This screen shot illustrates setting number formats for an object.

Slide—Screen Shot of Data Constants



This screen shot illustrates resetting a data constant.

Data Builder/UnBuilder

- Create specific data types
- Retrieve parts of data types
- Requires allocated array:
  - Get/Set values (array), Get/Set mapping (function), build spectrum, build waveform
- Does NOT require allocated array:
  - Build Coord, build PComplex, build Complex, build arb waveform

Other objects build data

See Also

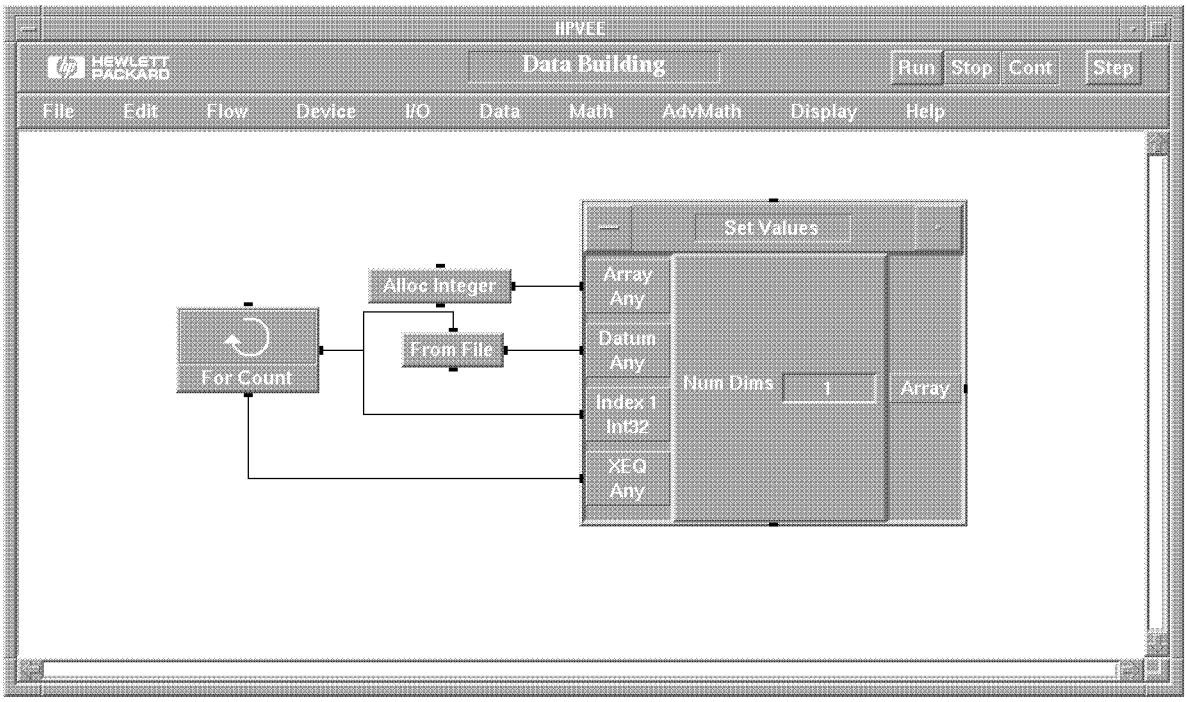
- Collector; virtual arbitrary waveform; generate step, ramp, impulse, etc.

E2100+24D V051

To create data of specific data types, HP VEE provides the `Build` objects. To retrieve data contained within specific data types, HP VEE provides the `UnBuild` objects.

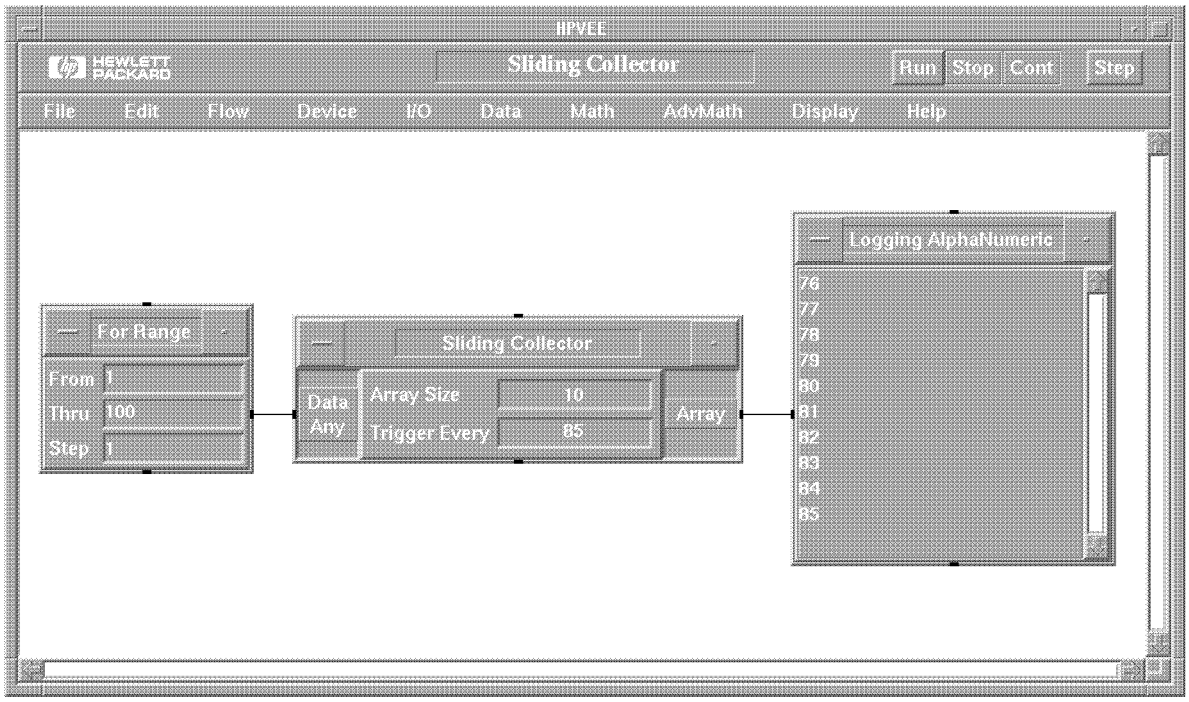
To work with arrays, use the `Get Values` and `Set Values` objects. But first be sure to allocate an array before trying to write data to it.

Slide—Screen Shot of Data Building



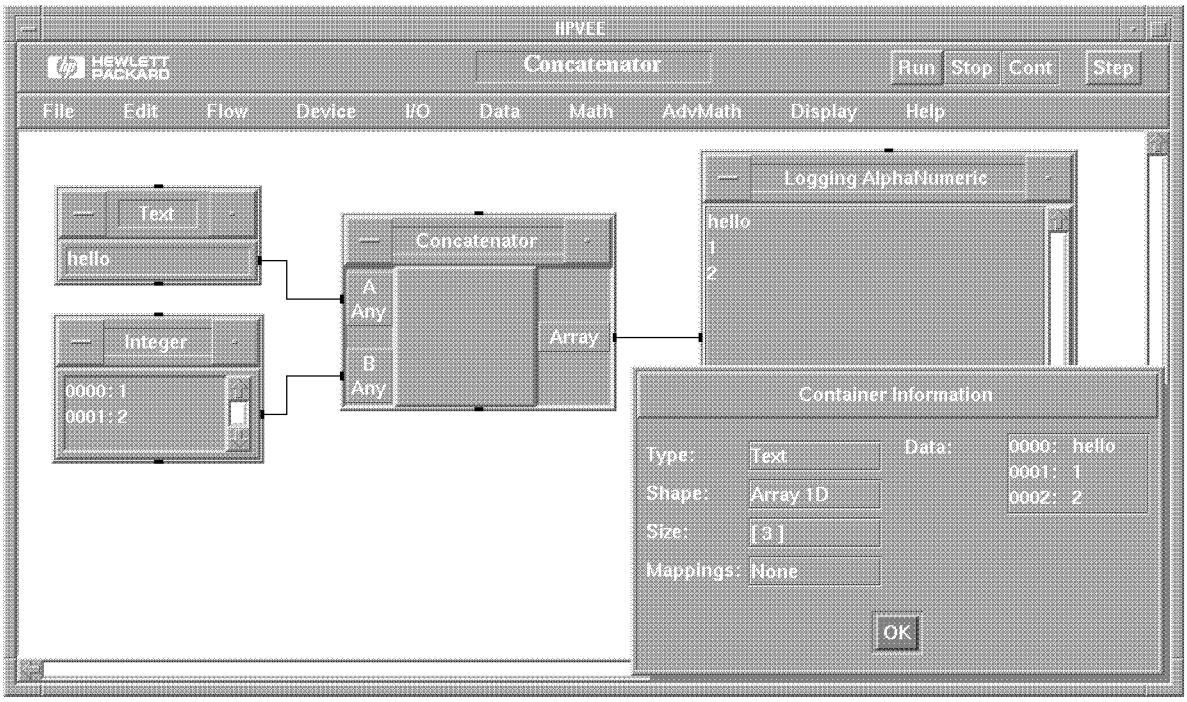
This screen shot illustrates reading individual datum from a file and writing them to an array.

Slide—Screen Shot of Sliding Collector



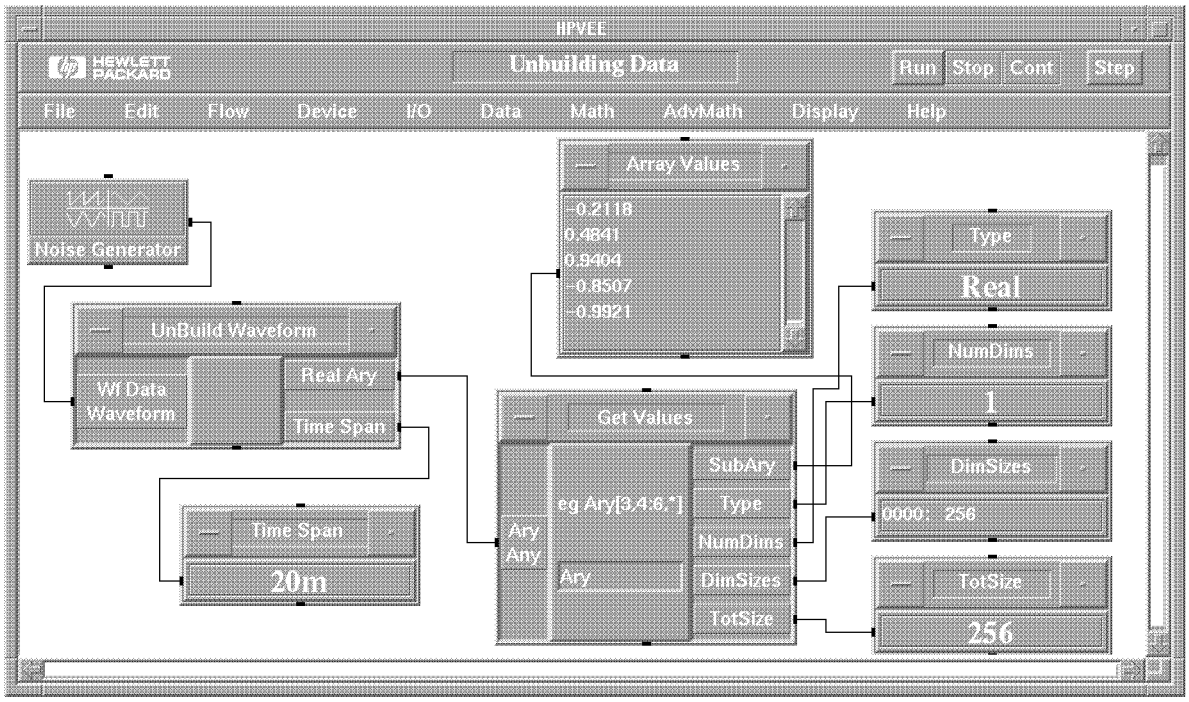
This screen shot shows how the `Sliding Collector` captures a portion of a data stream.

Slide—Screen Shot of Concatenator



This screen shot shows an open data container obtained by using a line probe. Note how HP VEE concatinates text and integer data into a single container. In this case the integers were promoted to be text.

Slide—Screen Shot of UnBuilding Data



This screen shot shows an example of unbuilding a waveform into its component pieces of data.

Sequence Control

- Start
  - Initiates execution of a thread
  - All START boxes are "pressed" (in no particular order) if RUN is pressed
- Confirm (OK)
  - Awaits user confirmation before continuing sequence
- Do
  - Allows developer to specify which subthread fires first
  - Propagates a sub-thread

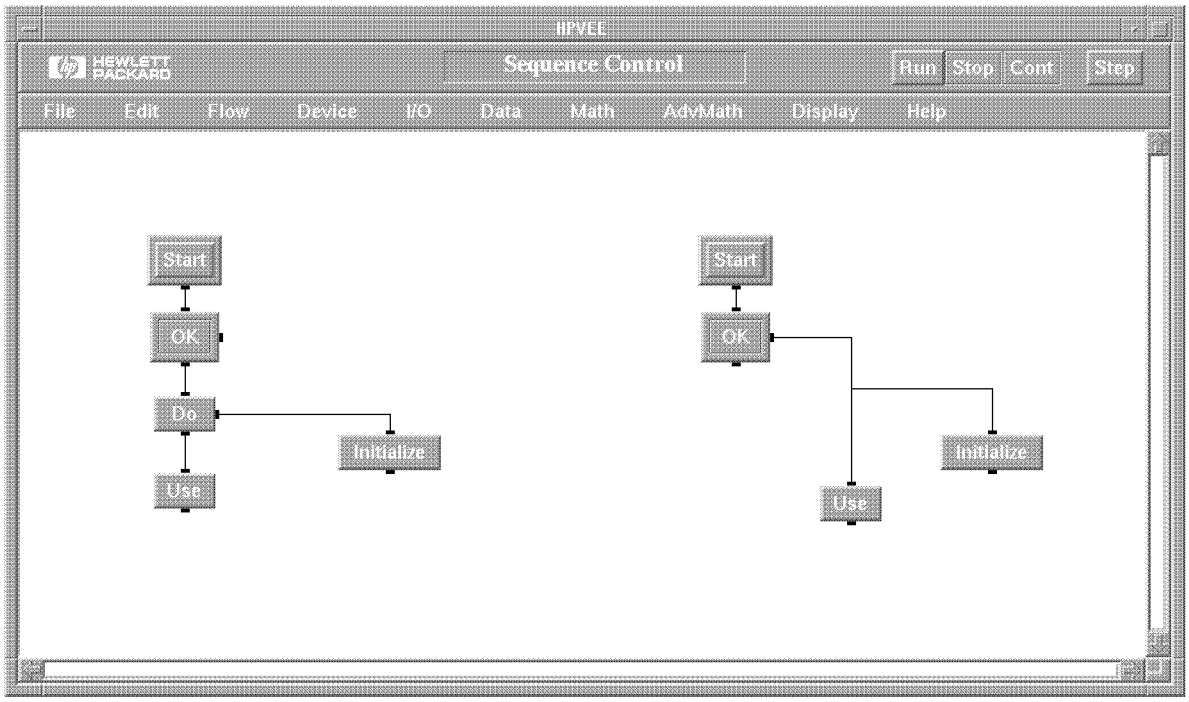
E2100+24D V056

The `Start` object initiate execution of a thread. Remember that all `Start` objects activate (in no particular order) when you press `Run`.

The `Confirm (OK)` object halts execution of the thread until the user presses the `OK` button.

The `Do` object lets you specify which subthread exectutes next. The following screen shot illustrates this point.

Slide—Screen Shot of Sequence Control



This screen shot illustrates the use of the Do object. In the thread on the left, the Initialize object will *always* operate first. In the thread on the right you cannot be sure which will operate first.

Repeat (Iterators)

- Repeatedly propagate data onto a subthread
- Bounded Loop
  - For Count
  - For Range
  - For Log Range
- Endless Loop
  - Until Break
  - On Cycle

E2100+24D V058

The `Repeat` objects let you repeatedly execute a subthread.

The first set of objects repeat a bounded loop, executing a set number of times.

`For Count` This object executes this subthread the number of times defined by the count. Note that if you use the output of the `Count` object that it counts from zero (0) *not* one.

`For Range` This object executes a subthread a number of times specified by a beginning value, and ending value and an increment.

`For Log Range` This object executes a subthread a number of times specified by a beginning value, and ending value and an increment. The output values are evenly distributed along the log 10 scale.

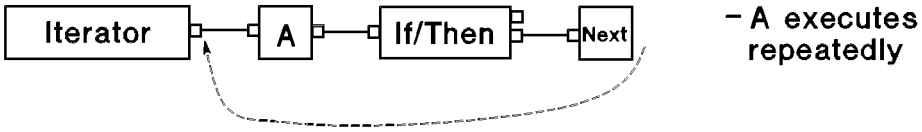
The second set of objects loop endlessly.

`Until Break` This object repeatedly executes a subthread until it encounters a `Break` object.

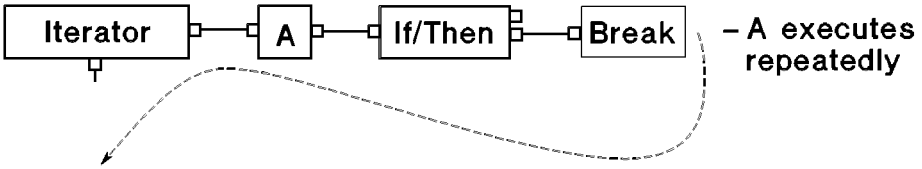
`On Cycle` This object repeatedly executes a subthread at a regulary timed interval.

Early Loop Termination

- Next – terminates propagation of current iteration



- Break – terminates current and future iterations



E2100+24D V059

HP VEE provides two ways to terminate a loop early.

The `Next` object ends the current iteration and allows the iterator to go on to the next iteration.

The `Break` object ends the current and all future iterations. The iterator activates its Sequence Out pin and stops operating.

Flow (Data)

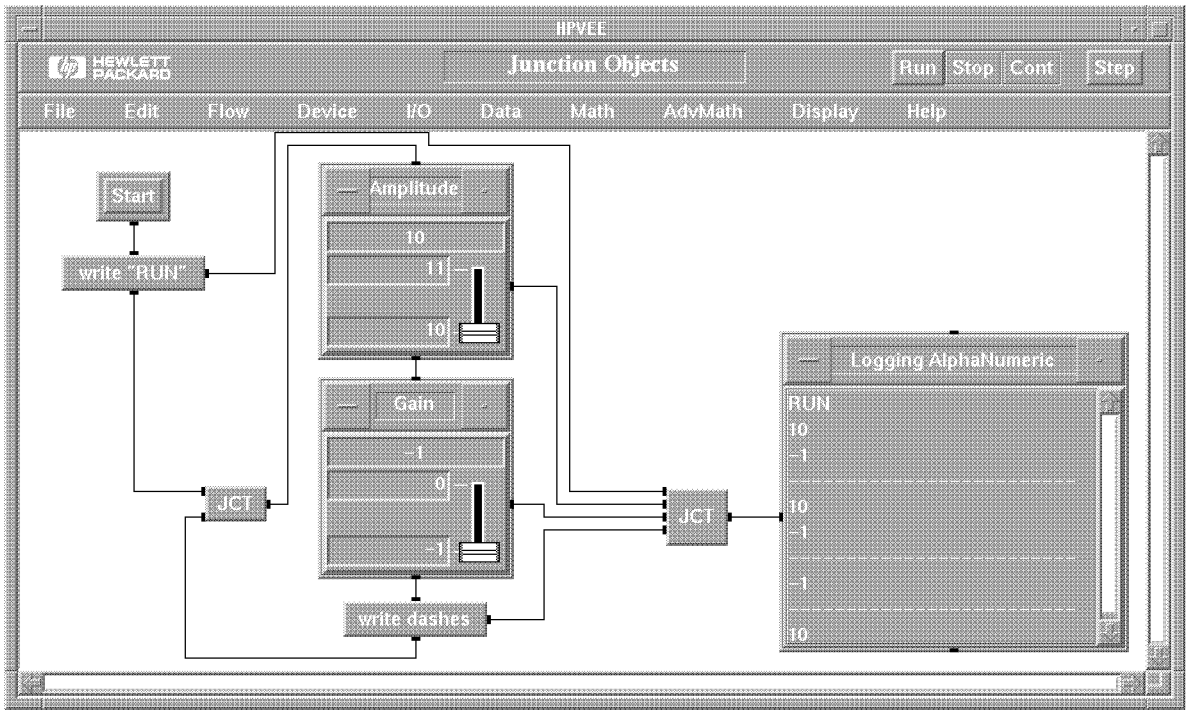
- Junction
  - Wired-OR which sends its most recent input data
  - Often used to send 2 or more data lines to the same input pin
  - Extra inputs are added as Data inputs
  - Only object with asynchronous inputs
- Gate
  - Similar to a "latch"
  - Holds input data until Sequence In is pinged (No sequence in connection on Gate ⟶ data passes through)

E2100+24D V060

As you may recall, an input pin may only have one connection. However, you may occasionally wnat to connect two outputs to one input. The `Junction` object does this for you. It acts like a "Wired-OR", sending out the most recently received input data container. If you need to use more than two inputs, simply add more data inputs to the `Junction`.

The `Gate` object is very similar to a latch. It holds its last input until its sequence in pin is activated. If the sequence in pin is not connected, it simply sends the data out immmediately.

Slide—Screen Shot of Junction Objects



This screen shot shows how a JCT object combines the data stream from several objects.

Conditional Branching

- If/Then
  - Allows testing according to user formula
  - Allows many inputs
  - Allows Else/If and Else outputs to give the capability for multi-conditionals (case arguments)
- Conditionals
  - Pre-formulated two-way comparisons

==	Equal
!=	Not equal
>,<	Greater, less than
>=,<=	Greater or equal, less or equal

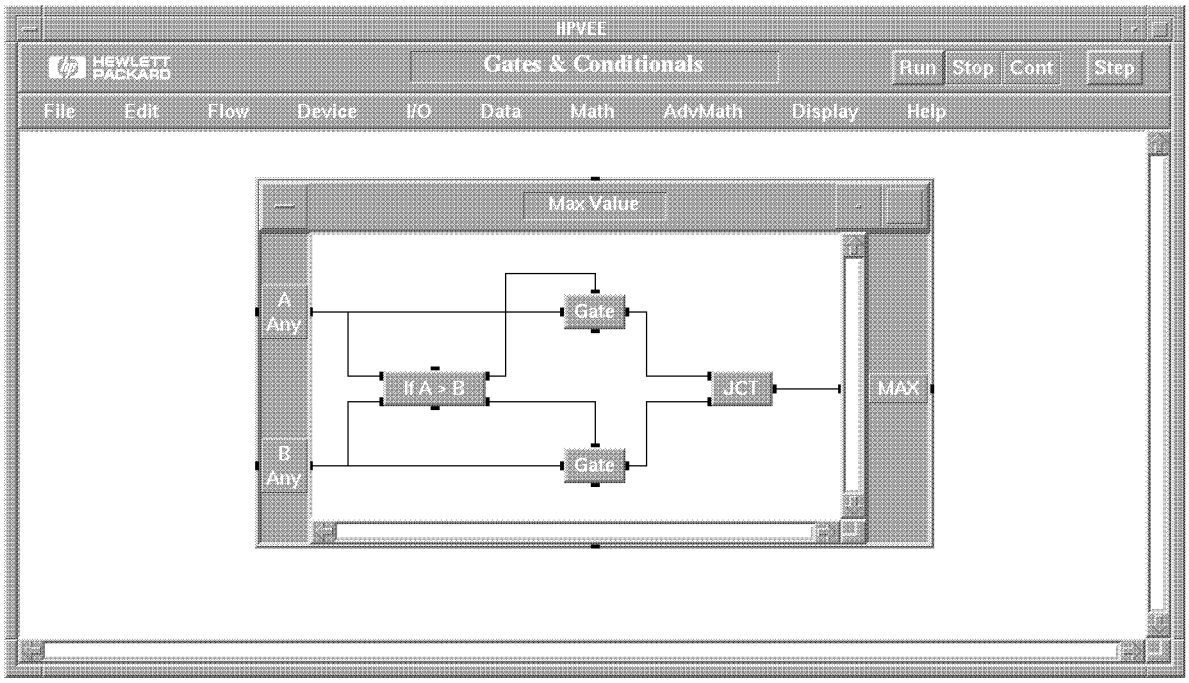
E2100+24D V062

HP VEE provides the `If/Then` and `Conditionals` objects so that you can test data and branch accordingly.

The `If/Then` object tests according to a user-provided formula. It also allows multiple inputs and the `Else/If` construct to provide multi-conditional tests.

The `Conditionals` are simply pre-formulated `If/Then` objects.

Slide—Screen Shot of Gates & Conditionals



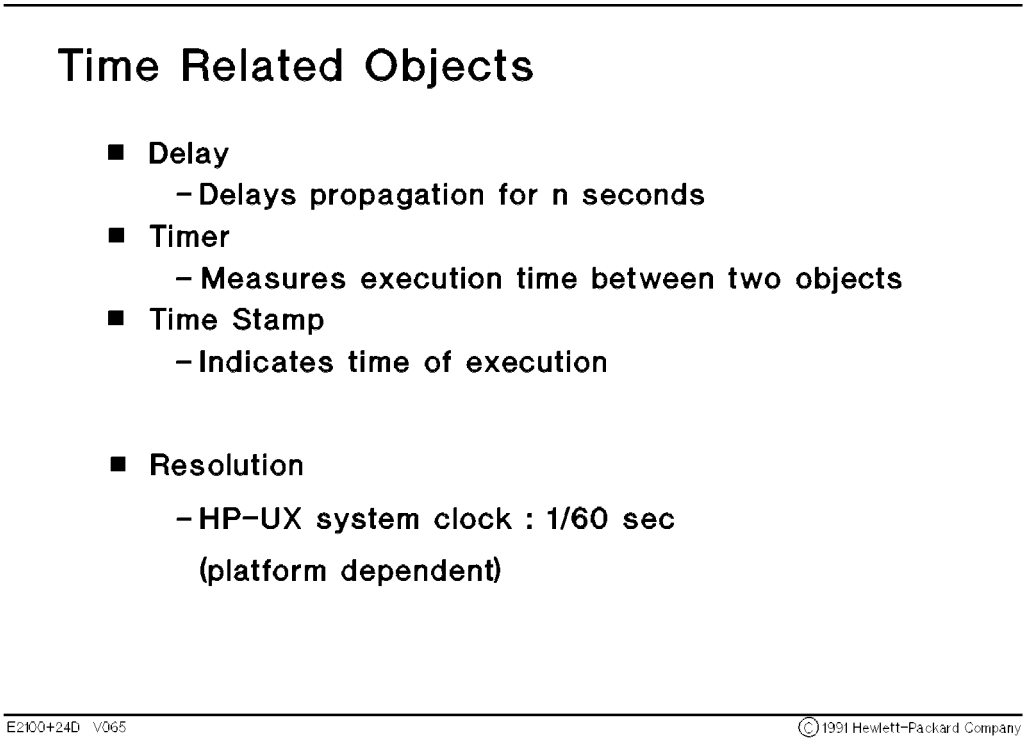
This screen shot shows how a `Conditional` object and `Gate` objects select one of two inputs depending on the input values.

Termination (Exits)

- Exit Thread
  - Terminates propagation of an individual thread
- Stop
  - Terminates model propagation
  - Equivalent to pressing stop button

E2100+24D V064

These two objects provide exits. `Exit Thread` terminates an individual thread. `Stop` terminates the execution of a model. Note that it stops the model immediately. No other objects operate. It is equivalent to pressing the `Stop` button.



The `Delay` objects puts the thread to sleep for n seconds, as defined by the user. After that time the thread continues.

The `Timer` object measures the time between receiving two data containers. You can use this to measure the execution time between two objects.

The `Time Stamp` provides the real number corresponding to the system real-time clock.

Note that the clock and timer resolution is system dependent, but generally is accurate to 1/60th of a second.

Misc. Objects

- Counter
  - Counts activations
- Accumulator
  - Running total
- Shift Register
  - Holds previous values
- DeMultiplexer
  - Redirect data or flow to 1 of n outputs
- Comparator
  - Compares data
  - Counts failures
  - Collects failures

E2100+24D V066

The `Counter` counts the number of times that it receives a data container.

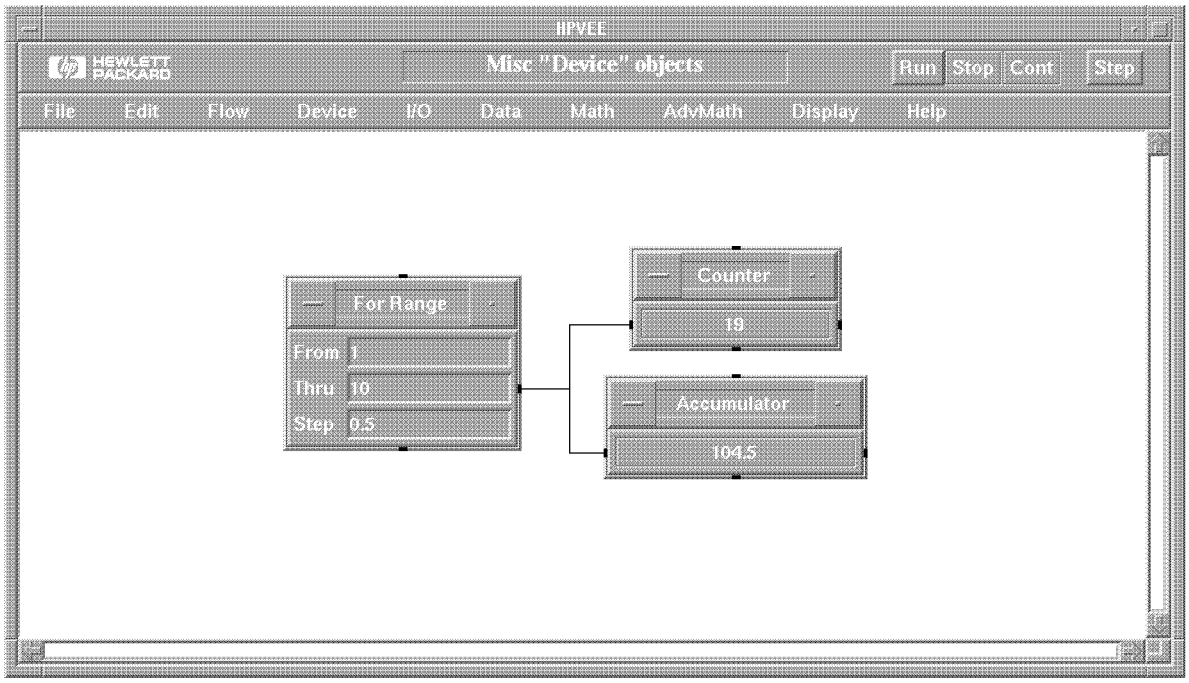
The `Accumulator` keeps a running total of the data it receives.

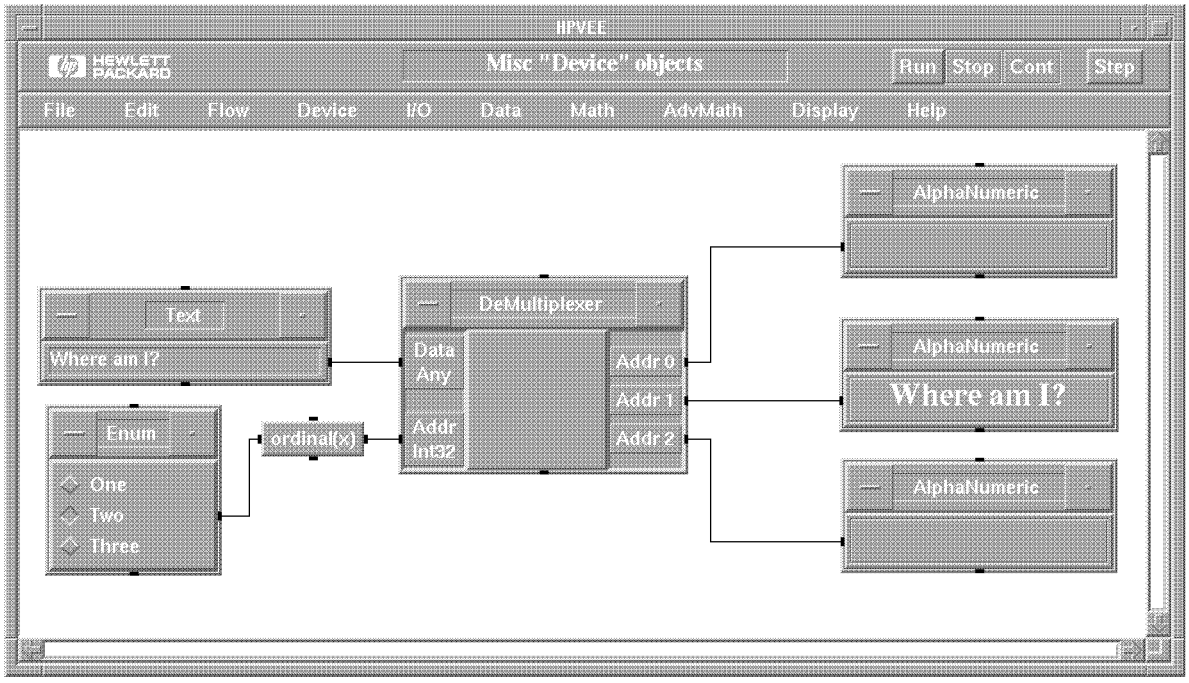
The `Shift Register` holds the previous `n` values. You can define `n` by the number of output pins on this object.

The `DeMultiplexer` redirects data containers to 1 of `n` threads. You can define `n` by the number of ouput pins on this object.

The `Comparator` compares two values, counts the number of failures and collects the coordinates of the failures.

Slide—Screen Shot of Misc. Objects





Textual Displays

- Alphanumeric
  - Displays a single value
- Logging Alphanumeric
  - Scrolling text display
  - Configure buffer size
- VU Meter
  - Analog meter
  - Red, yellow, green limits available

All textual displays allow:

- Clear at PreRun
- Clear at Activate
- Number Formats

E2100+24D V069

The `Alphanumeric` object displays a single text value.

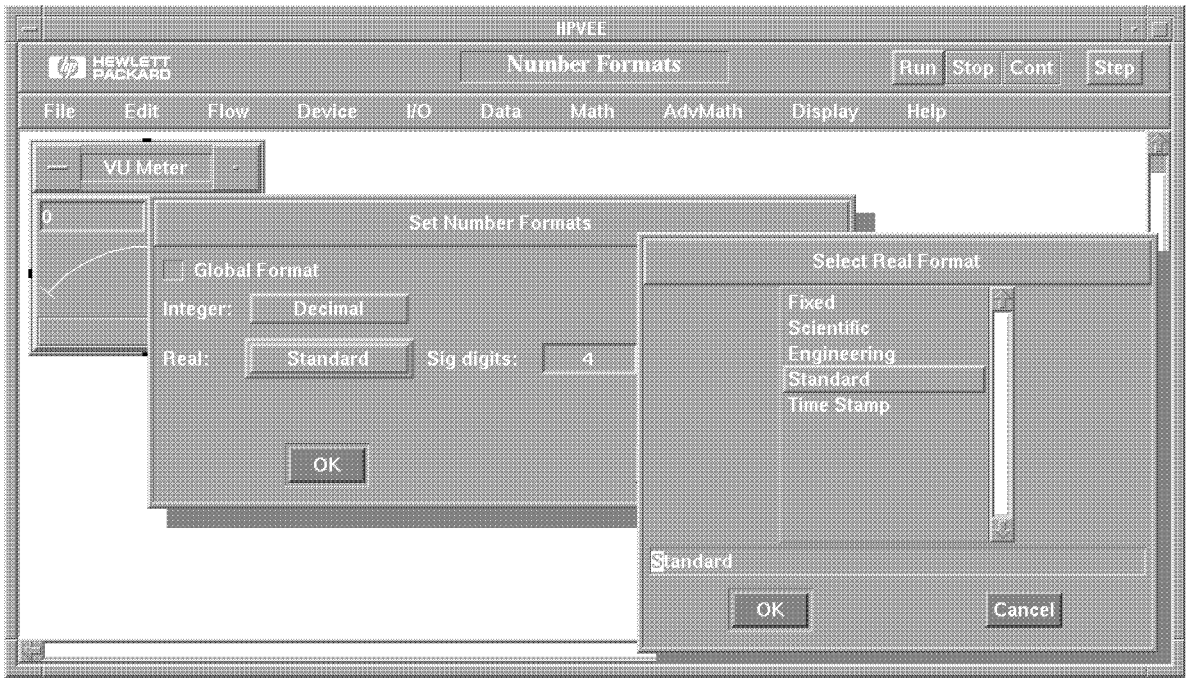
The `Logging Alphanumeric` provides a scrolling text display. The user can configure the buffer size for the data.

The `VU Meter` is an analog meter display. You can add red, yellow, and green limits on the scale.

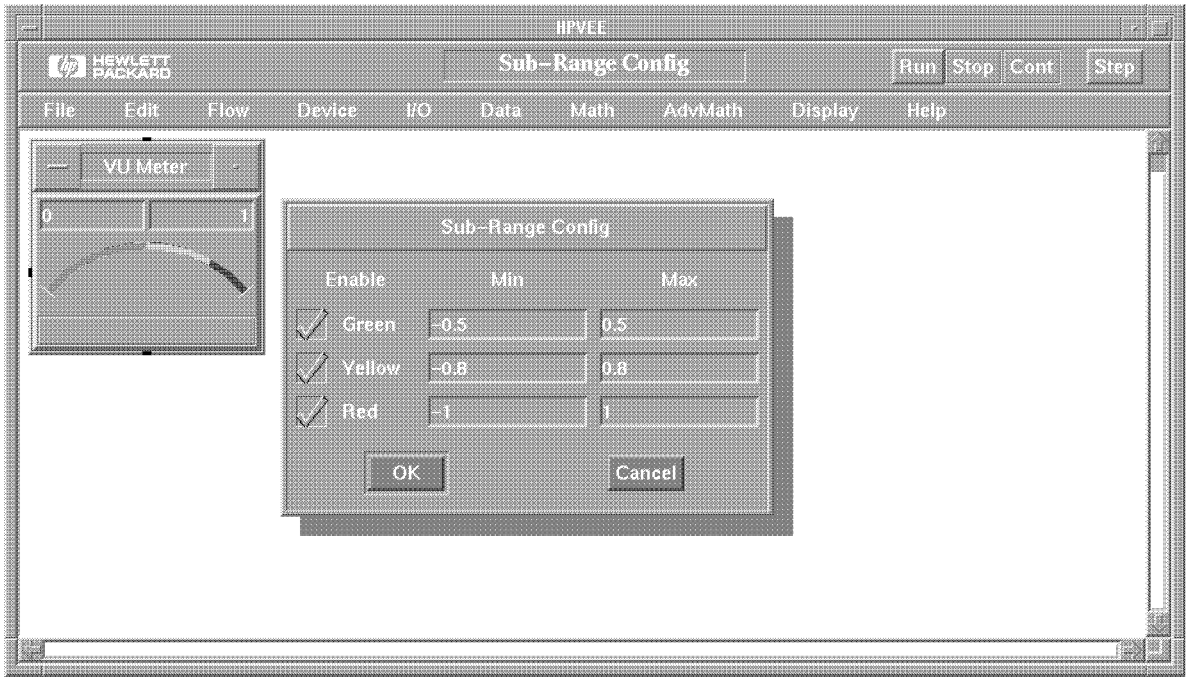
These displays permit:

- Clear at PreRun
- Clear at Activate
- Set Number Formats
- Set Buffer Size

Slide—Screen Shot of VU Meter



Slide—Screen Shot of VU Meter



Graphical Displays

- XY Trace
  - Y data points plotted on an arbitrary X axis
- Strip Chart
  - Y data points plotted on an arbitrary axis
  - X and Y axis both scroll as data is received
- Complex Plane
  - Plots Real vs. Imaginary
- X vs Y Plot
  - Plots pairs of X,Y data points
- Polar Plot
  - Plots radius vs. angle
  - Includes options for Smith, Inverted Smith
- Waveform (Time)
  - Plots waveform data type data
- Spectrum

E2100+24D V072

There are six different graphical displays.

The `XY Trace` plots a two-dimensional cartesian coordinate plot.

The `Strip Chart` displays XY data while continuously scrolling the X and Y axis.

The `Complex Plane` plots a two-dimensional cartesian coordinate plot of complex data (Real vs Imaginary).

The `X vs Y Plot` plots pairs of (x,y) data points.

The `Polar Plot` plots a two-dimensional polar plot. It includes options to plot Smith charts and Inverted Smith charts.

The `Waveform (Time)` plots a two-dimensional waveform plot against time.

Spectrum Displays

- Magnitude
  - Plots magnitude vs frequency
- Phase
  - Plots phase vs. frequency
- Magnitude vs Phase

  Polar
  - Data plotted on a polar plane

  Smith
  - Data plotted on a Smith chart

E2100+24D V073

These objects display Spectrum plots. Note that you can plot magnitude vs phase or phase vs frequency. You can also plot magnitude vs phase on a polar plane or a Smith chart.

Display Customization

- **Multiple Data, Control Inputs**
- **Autoscale (X only, Y only, Both)**
- **Clear control (at Activate, at PreRun, Next Curve)**
- **Zoom (In, Out, Etc.)**

E2100+24D V074

You can customize the display objects in many ways. For multiple graphs, add more data inputs. `Clear` and `Autoscale` can be control inputs.

Plot multiple data instances by using the `Next Curve` control.

You can magnify the view of the data by using the `Zoom` command.

Display Customization (cont.)

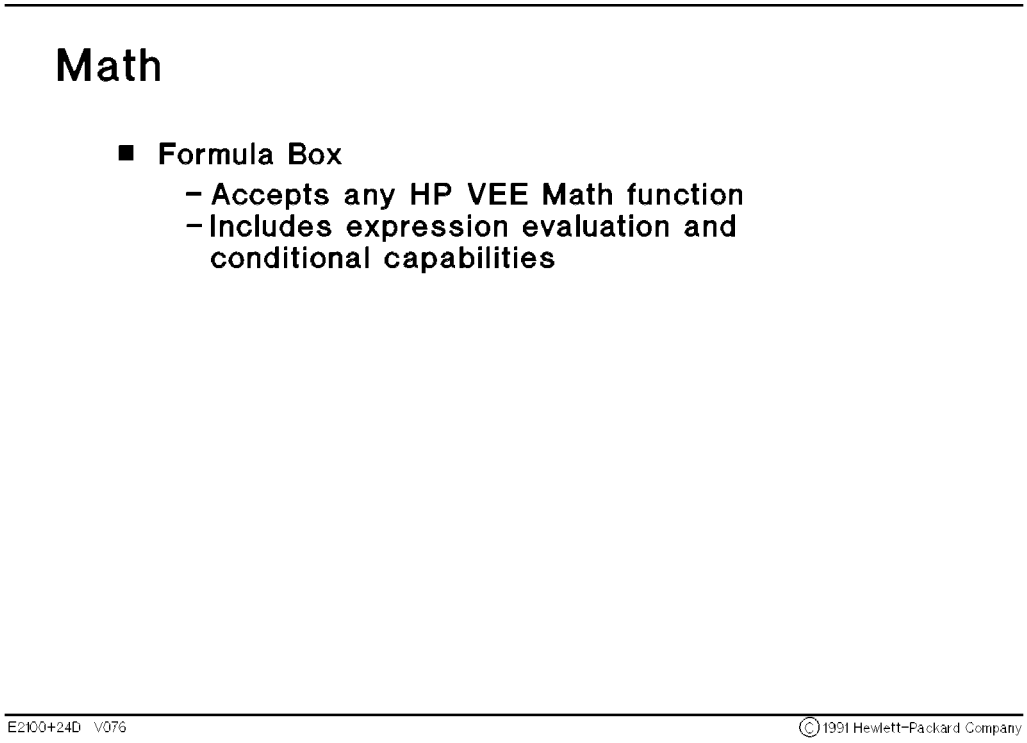
- Markers (One, Two, Delta,Interpolate, Etc.)
- Grid Type (Tic Marks, Lines, Etc.)
- Panel Layout (Graph Only, Show Scales, Etc.)
- Set Trace (Scaling, Color, Line Type, Etc.)
- Add Additional Scales on the Right

Note that many functions can be added as control pins

E2100+24D V075

You can also modify the panel layout, grid type, trace color and trace texture used on the displays.

Note that many of these functions may be added as control pins to the object.



The `Formula` object accepts any HP VEE Math function including expression evaluation and conditional tests. With this object you can define any math function.

Use the `Formula` objects to increase the efficiency of your model. `Formula` objects operate most effeciently since it will only execute one object instead of the many objects that would otherwise be needed. For example, instead of using several objects to create the math operation `sin(A+(B/C))`, simply type the formula into a `Formula` object.

Math

- **Generate (Functions)**
  - Ramp
  - Log Ramp

E2100+24D V077

In addition to the `Formula` object, HP VEE provides some predefined `Formula` objects with prewritten math operations. These are listed on the following slides. If you can't find your favorite math function, you can write it using the `Formula` object.

The following slides list all of these math functions.

Math

- General
  - +
  - –
  - *
  - /
  - ^
  - Mod
  - Div
- Relational
  - ==
  - !=
  - <
  - >
  - <=
  - >=
- Bitwise
  - Bit
  - Bits
  - Set bit
  - Clear bit
  - Bit and
  - Bit or
  - Bit xor
  - Bit compl
  - Bit shift
- Logical
  - And
  - Or
  - Xor
  - Not

E2100+24D V078

Math

- Real Parts
  - Abs
  - Sign of
  - Ordinal
  - Round
  - Floor
  - Ceil
  - Int part
  - Frac part
- Complex Parts
  - J
  - Re
  - Im
  - Mag
  - Phase
  - Conj

E2100+24D V079

Math

- Power
  - Sq
  - Sqrt
  - Cubert (cube root)
  - Recip
  - Log
  - Log 10
  - Exp
  - Exp 10
- Polynomial
  - 1: Poly (x,[a0 a1])
  - 2: Poly (x,[a0 a1 a2])
  - 3: Poly (x,[a0 a1 a2 a3])
  - N: Poly (x,[a0 a1 ...aN])

E2100+24D V080

Math

- Trig
  - Sin
  - Cos
  - Tan
  - Cot
  - Asin
  - Acos
  - Atan
  - Acot
  - Atan2
- Hyperbolic Trig
  - Sinh
  - Cosh
  - Tanh
  - Coth
  - Asinh
  - Acosh
  - Atanh
  - Acoth

E2100+24D V081

Math

- Time & Date
  - Now
  - Wday
  - Mday
  - Month
  - Year
  - Dmytodate
  - Hmstosec
  - Hmstohour

Advanced Math

- Array
  - Init
  - Rotate
  - Concat
  - Sum
  - Prod
- Matrix
  - Det
  - Inverse
  - Transpose
  - Identity
  - Minor
  - Cofactor
  - Matmultiply
  - Matdivide
- Calculus
  - Integral
  - Deriv(x,1)
  - Deriv(x,2)
  - Deriv(x,order)
  - Defintegral
  - Derivat(x,1,pt)
  - Derivat(x,2,pt)
  - Derivat(x, order, pt)

Advanced Math

- Regression
  - Linear
  - Logarithmic
  - Exponential
  - Power curve
  - Polynomial
- Data Filtering
  - Polysmooth
  - Meansmooth
  - Movingavg
  - Clipupper
  - Cliplower
  - Minindex
  - Maxindex
  - Minx
  - Maxx

Advanced Math

- Probability
  - Random (low, high)
  - Randomize
  - Random seed
  - Perm
  - Comb
  - Gamma
  - Beta
  - Factorial
  - Binomial
  - Erfc
  - Erf
- Statistics
  - Min
  - Max
  - Median
  - Mode
  - Mean
  - Sdev
  - Vari
  - Rms
- Freq. Distribution
  - Lin mag dist
  - Log mag dist

Advanced Math

- Bessel
  - J0
  - J1
  - Jn
  - Y0
  - Y1
  - Yn
  - Ai
  - Bi
- Hyper Bessel
  - I0
  - I1
  - K0
  - K1
- Signal Processing
  - Fft
  - Ifft
  - Convolve
  - Xcorrelate
  - Bartlett
  - Hamming
  - Hanning
  - Blackman
  - Rect

Lab 2a—Apple Bagger

Background

Manufacturers fill food packages using a netweight method. In other words, they fill containers by weight not by volume. But just for fun, a new manager wants to know how many apples it takes to fill a ten pound basket. Your job is to weigh and count the apples that fill the basket.

Task

Create a model that counts how many apples it takes to fill a ten pound basket of apples. Each apple weighs between 0 and 1 pound.

Suggested Objects

This model can be created with eight or fewer objects. Choose from the following objects:

```
Start
Until Break
Random Number
Accumulator
Real
Conditional (A>=B)
Stop
Counter
If/Then
Break
```

Lab 2b—Testing Numbers

Task 1

Create a model that allows a user to enter a number between 0 and 100. If the number is greater than or equal to 50, display the number. If the number is less than 50, display the message, "Sorry!"

Suggested Objects

This model can be created with seven or fewer objects. Choose from the following objects:

`Start`

`Integer Slider`

`Real`

`If/Then`

`Formula`

`Gate`

`Text`

`Junction`

`Alphanumeric`

Task 2

After the model is working with seven objects, try the following:

1. Delete the `Start` object, leaving the user input as the first object in the thread.
2. Create this model without using a `Gate` object.

Lab 2c—Collect Random Numbers

Task

Create a model that generates 100 random numbers and displays them. Record the total time required to generate and display the values.

Suggested Objects

This model can be created with six or fewer objects. Choose from the following objects:

- `Start`
- `For Range`
- `Until Break`
- `Random Seed`
- `Random Number`
- `Collector`
- `Set Values`
- `Allocate Array`
- `Logging Alphanumeric`
- `Strip Chart`
- `VU Meter`
- `Date/Time`
- `Timer`
- `Time Stamp`
- `Break`

Hint

To improve performance, send the data to the display only once (one container) by first collecting it in a `Collector` object.

Lab 2d—Random Number Generator

Task

Create a random number generator. Display the random numbers. Test for randomness by displaying the numbers graphically. Provide control for the following parameters:

- Maximum random number
- Minumum random number
- number of random numbers generated

3

Virtual Sources

Virtual Source Objects

Virtual Source Objects

- Simulated function, pulse, noise, arb waveform generators
- Generate dynamic data models
  - Useful for prototyping
- Full control of model
  - Phase, amplitude, sample points, etc.
- Waveform data type output

E2100+24D V087

The `Virtual Source` objects, found in the `Device` menu, provide simulated function, pulse and noise generators. The dynamic data models they generate are useful for prototyping complex models. You have full control over the device parameters

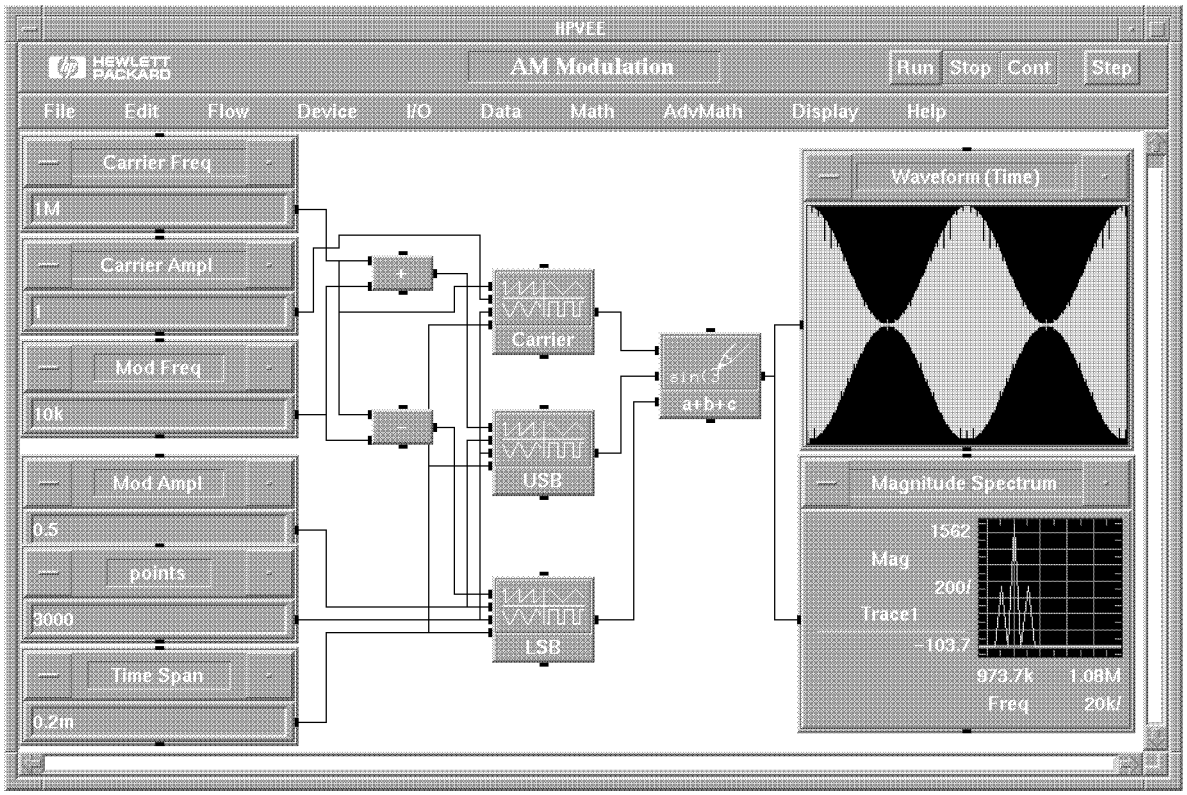
Function Generator

- Functions
  - Sine, Cosine, Square, Triangle, +Ramp, −Ramp, DcOnly
- Frequency
- Amplitude
- DC Offset
- Phase
  - Deg, Rad, Grad
- Time Span
  - Time interval for waveform sample
- Num Points
  - Determines sampling detail
  - Too few points can cause aliasing

E2100+24D V088

The `Function Generator` provides the waveform types listed in the slide above. Note that you can control frquency, amplitude, offset and phase. In addition you can select the time interval and the number of sampling points for the waveform. Remember that the number of sampling points determines the detail of the waveform and the processing time. Also remember that too few points can cause aliasing and give erroneous information. HP VEE warns you if you use too few points unless you disable the `Error on Aliasing` feature.

Screenshot - AM Modulation



Here is an example of one use for the `Function Generator`. Note that the `Formula` object can add waveforms.

Pulse Generator

- Rep Rate
  - Repetitions per second
- Pulse Width
- Pulse Delay
  - From 0 seconds
- Thresholds
  - 0%–100%, 10%–90%, 20%–80% (rise/fall time calculation)
- High – Maximum Value
- Low – Minimum Value
- Burst Mode
  - On/Off
  - Burst Count
  - Burst Rep Rate (per second)

E2100+24D V090

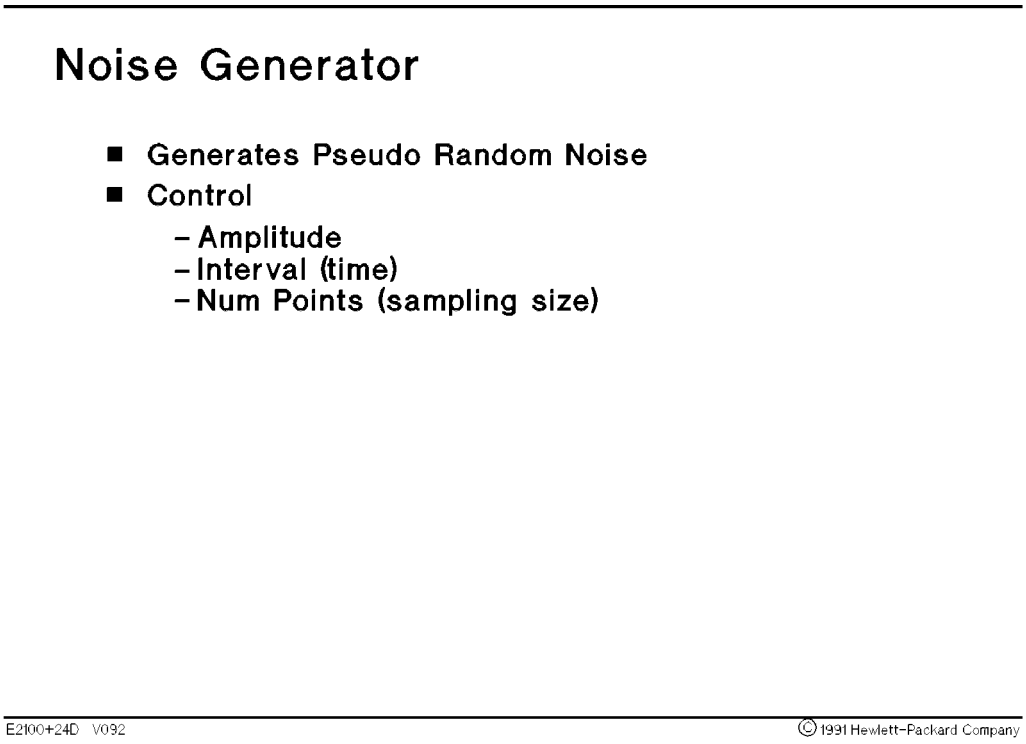
The `Pulse Generator` provides the pulse funciton. Note that you have full control of the generator as with the function generator.

Pulse Generator

- Interval
  - Waveform time interval
- Num Points
  - Sampling size

E2100+24D V091

Again, you can control the time interval of the waveforem and the sampling size.



The `Noise Generator` provide pseudo random noise. Again HP VEE permits you to control all the parameters.

Other Waveform Sources

- **Build Waveform**
- **Build Arb Waveform**

E2100+24D V093

You can also create waveform data using the `Build Waveform` and `Build Arb Waveform` objects. You can also build waveforms by feeding the `Ramp` and `Log Ramp` math functions into the `Build Waveform` object.

Lab 3a—Two Mask Test

Task 1

1. Create a 50Hz sine wave with a user-controlable amount of noise.
2. Test the noisy sine wave to be certain that it stays below the following limits:

Upper Test Mask

Time	Maximum Value
0	0.5
2.2m	1.2
7.2m	1.2
10.2m	0.5
20m	0.5

3. If the waveform exceeds these limits, mark the failing points with a red diamond.

Hint

You can change the format of the Graphical Displays from lines to dots and *other* graphical markers. Look under the `Traces and Scales ...` menu operation on the display object's menu.

Notice that the `Function Generator` default funtion is `Cosine`. Click on the raised box labeled `Cosine` and select the `Sine` function in the `Select Function` list box. A sine wave works best with the mask coordinates provided in this lab.

Task 2 (Optional)

1. Create a lower set of limits.
2. Make the test repeatable and show the percent of failures.

Lab 3b—Lissajous Figures

Background

In "the good old days," back before scopes *needed* a trigger, engineers determined their unknown frequencies by comparing them to a known frequency. The frequency standard drove the horizontal sweep, while the unknown frequency drove the vertical sweep. By consulting a table of Lissajous figures, the unknown frequency and phase would be determined.

Task

1. Create a model that generates and displays Lissajous figures.
2. Sweep the frequency ratio from 1x to 5x.
3. For each frequency ratio, vary the phase from 0 to 180 degrees.

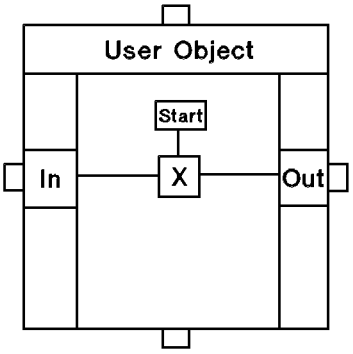
4

UserObjects

UserObjects

UserObjects

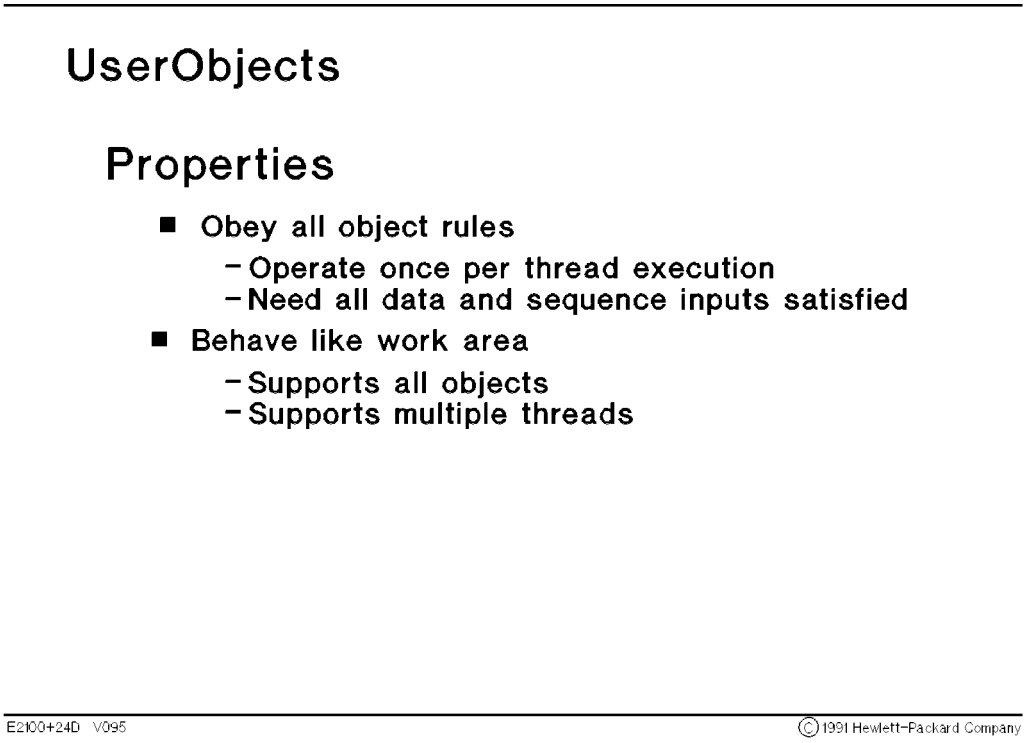
- A work area within an object



- A Context

E2100+24D V094

`UserObjects` provide you with a work area within a workarea. You build models inside it that are completely independent of the work area outside of it. In fact you can run a model inside the `UserObject` without affecting anything else in your work area. They create a new context, or working environment.



`UserObjects` obey all of the same rules as other objects, namely

- They operate only once per thread execution (including sending out data).
- They must have all data and sequence inputs satisfied before operating.

They behave exactly like the general work area, supporting all HP VEE objects and multiple threads.

Purpose

- **Encapsulate groups of objects that provide a function into single object**
  - **–Unclutters work area**
  - **– Facilitates easy understanding of model behavior**
- **Allows modular ("top down") design**
  - **– Unlimited nesting**
- **Can be stored in central object directory**
  - **– Easy sharing and re–use**

E2100+24D V096

`UserObjects` are quite helpful in the development of HP VEE models. They permit you to encapsulate groups of objects that provide a function into a single object, thus uncluttering the work area. This also helps to document the model and guide understanding the model behavior.

This also facilitates "top-down" design. You can nest an unlimited number of `UserObjects`. So create your model as a block diagram. Then simply fill in each `UserObject` with the necessary functions.

You can also create a library of shared `UserObjects`.

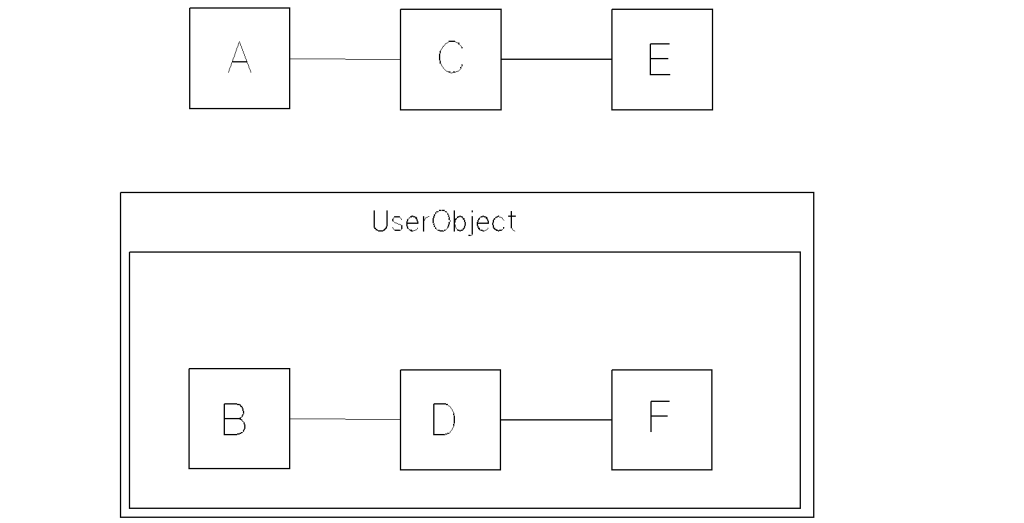
Initiation and Execution of UserObjects

- Activated when data and sequence inputs are satisfied
- Data inputs act as Start objects
- Each object operates in a time-sliced fashion
- Internal vs. external activation
  - Internal activation (Start); data does not activate data out pins

The `UserObject` operates in the following fashion:

- It activates only when all data and sequence inputs are satisfied, even if one of its multiple internal threads only requires one of the data inputs. All internal threads activate at once.
- The data inputs act as `Start` objects for each thread. It then follows all of the standard propogation rules.
- Each object executes in the time-sliced manner describe previously for multiple threads.
- When a thread is activated internally by a `Start` object within the the `UserObject` the data remains inside the `UserObject` and does not propagate outside. Data will only propogate out of a `UserObject` when it is activated by its surrounding context.

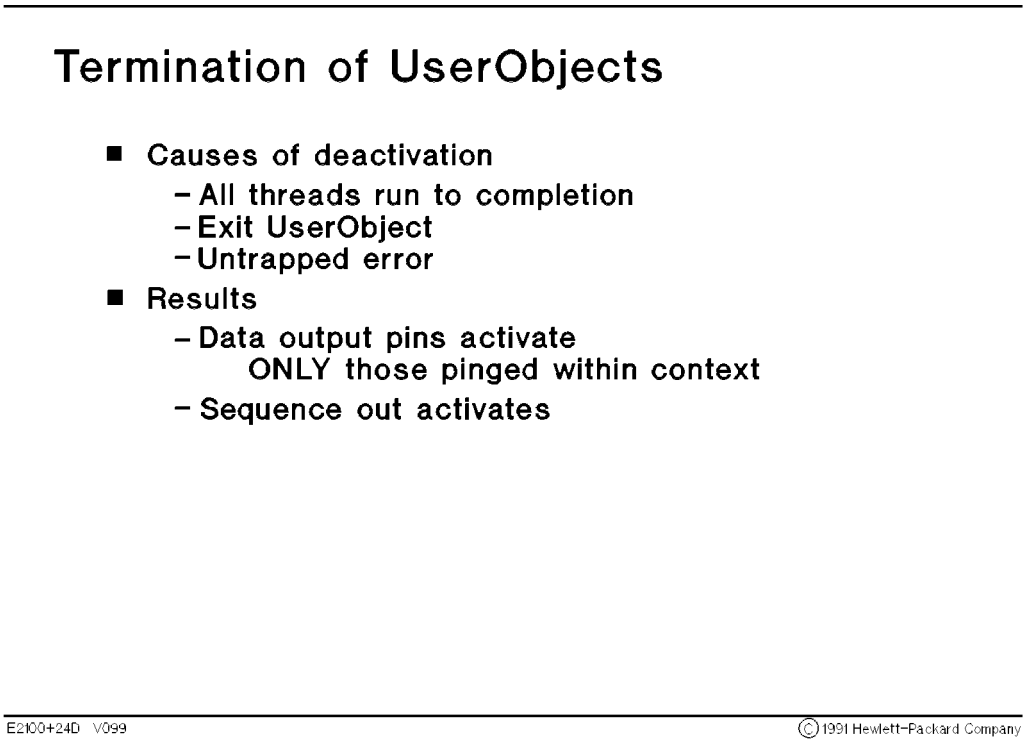
Parallel Subthread Example



E2100+24D V098

Here is an example of how the propogation engine shares time between parallel threads, where a `UserObject` contains one of the threads.

Note that the propogation engine treats each object inside the `UserObject` as a primitive object and executes them one at a time. In this example, the objects execute in alphabetical order of the labels .



A `UserObject` terminates operation for the following reasons:

- *All* threads completed execution.
- Thread execution encounters an `Exit UserObject` device.
- A thread causes an untrapped error.

When the `UserObject` terminates operation it activates only its data output pins that received data from within the `UserObject`. It then activates its sequence out pin.

Early Termination

- **Exit UserObject**
  - All threads in context halt
  - Outputs which received data activate
  - Sequence out activates
- **Escape**
  - User-generated error
  - All threads in context halt
  - NO data pins activate
  - Error pin generates escape code
    Else "error" dialog
- **Errors can "bubble up" through nested UserObjects**

E2100+24D V100

As mentioned before, `UserObjects` terminate operation when it encounters an `Exit UserObject`. At that point

- All threads in the `UserObject` terminate operation.
- Output pins that received data, activate.
- Sequence out activates.

`UserObjects` will also stop operation when they ecounter an `Escape` object. This provides a user-generated error condition. At that time

- All threads in the `UserObject` terminate operation.
- *No* output pins activate.
- The `Error` pin (if one exists on this `UserObject`) generates the escape code from the `Escape` object. If there is no `Error` pin, an error dialog box appears.

Note that you can allow error messages to propagate up through nested `UserObjects`.

Building a UserObject – Encapsulate Existing Objects (Method 1)

- Select desired object(s)
- Create UserObject

- **Advantages**
  - All connections become data pins
  - Allows prototyping in top-level workarea
- **Disadvantages**
  - Redundant connections must be edited
  - Ill-conceived object selection yields nonfunctional UserObject

E2100+24D V101

There are two methods of creating `UserObjects`. In the first method, you select your desired objects, create a working function and then encapsulate it in a `UserObject`.

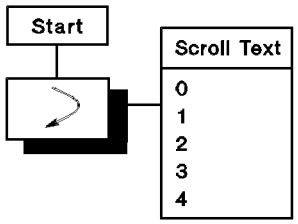
The advantage to this method is that all connections become data pins automatically. It also allows you to prototype in the main work area.

This method also presents some disadvantages. Encapsulating a function often creates redundant data pin connections which must be deleted. Also, ill-conceived object selection and the misunderstanding of how an object operates often yeilds a non-funcioning `UserObject`. "It just doesn't work like it did before."

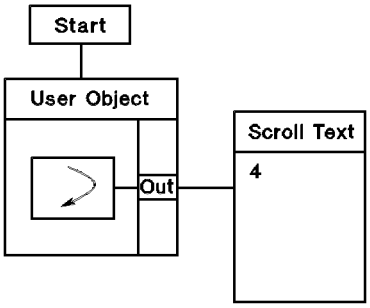
We'll illustrate this with the following slide.

Common Problem in Create User Object

- The working model...



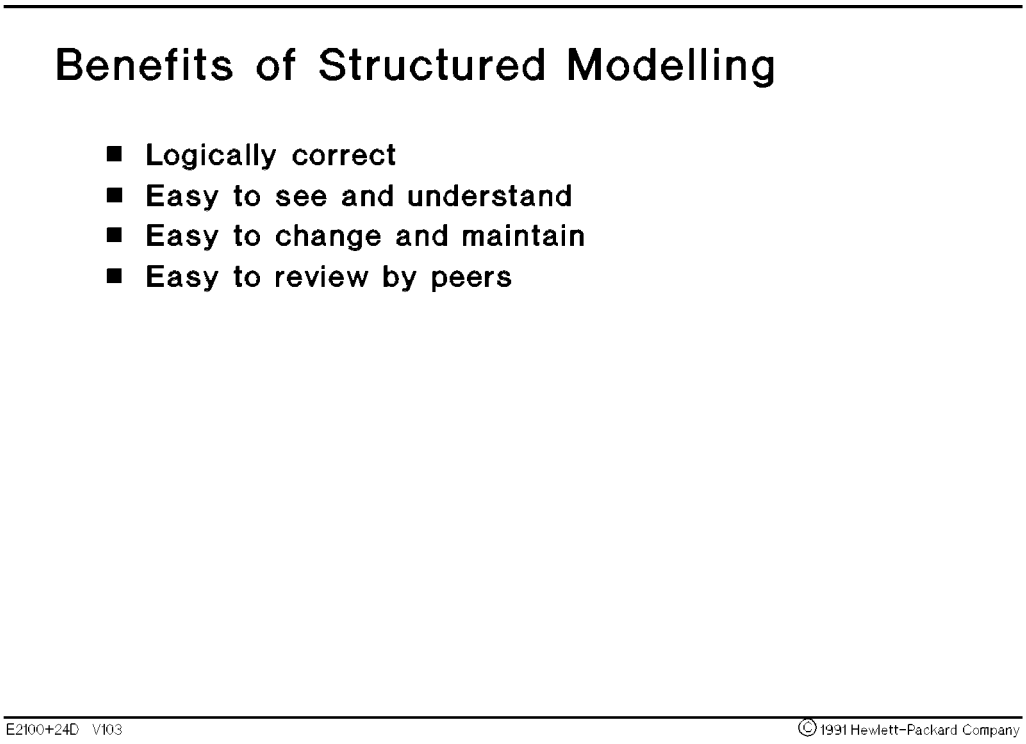
- ... partially encapsulated



- GIVES DIFFERENT RESULTS!

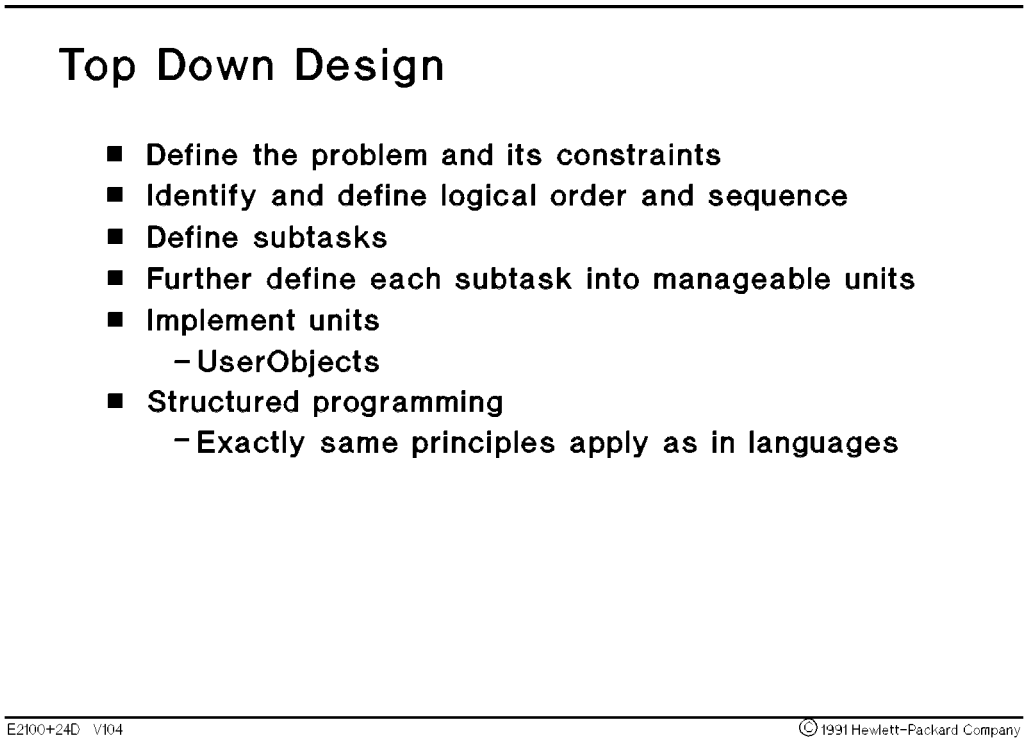
E2100+24D V102

In the model at the top, the iterator generates five pieces of data. But encapsulating the iterator in a `UserObject` produces only one datum. Remember that objects only operate once and only provide one data container.



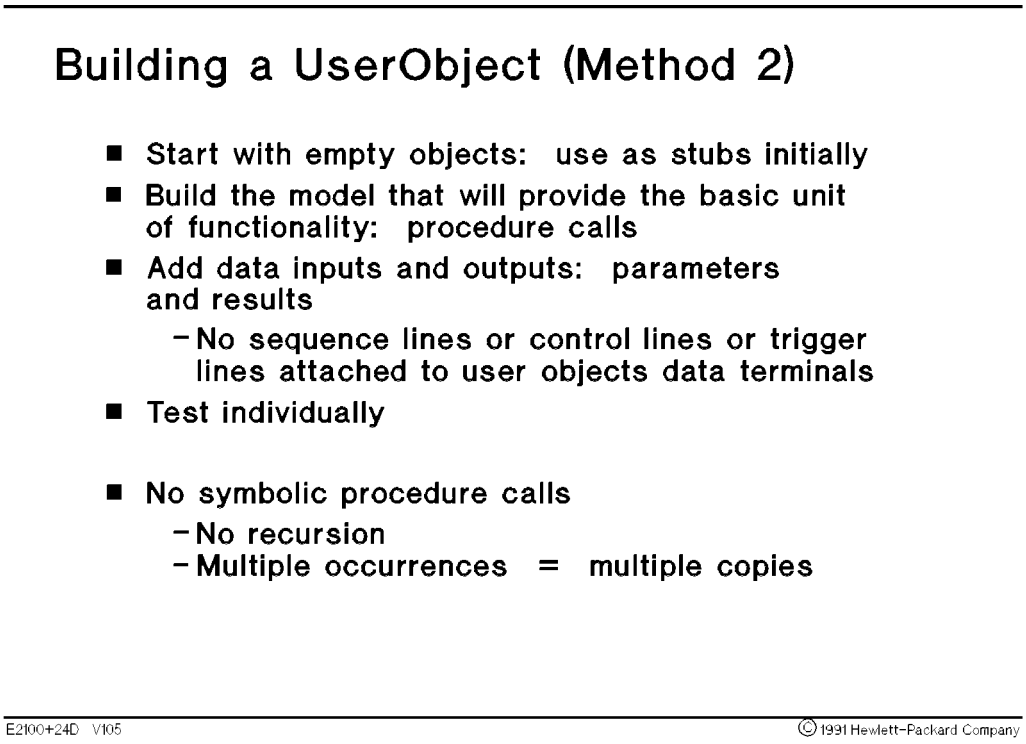
Here you can see the benefits of Structured Modelling.

- Its easier to verify that the model is logically correct.
- Its easier to see and understand the model and its structures.
- Its much easier to change and maintain the model.
- Its easier for peers to reveiw the model.



Lets reveiw the "Top-Down" design methodology by refering to the above slide.

- Define the problem, including all of its contraints.
- Identify a logical order and sequence for each of the tasks within the problem.
- Define each subtask within each task. Note how well this fits into HP VEE's paradigm of nested `UserObjects`.
- Continue defining each subtask into manageble units.
- Now implement each of these tasks as a `UserObject` within HP VEE.

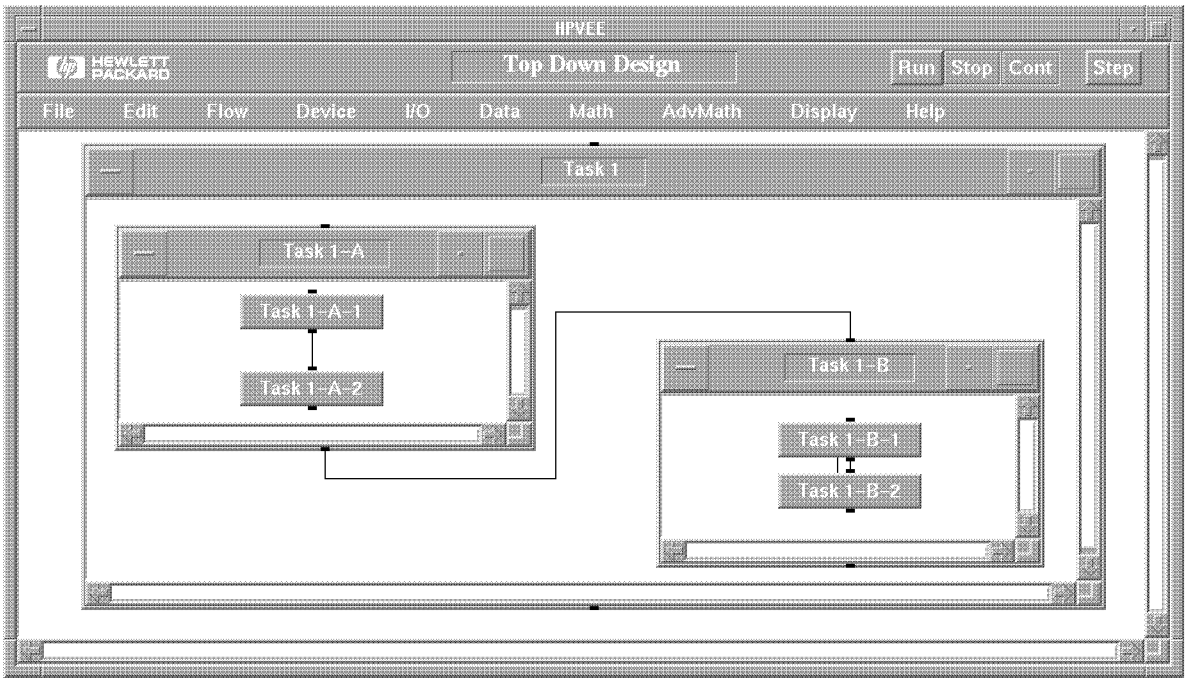


Here's how you do top-down design with `UserObjects`.

- Start with empty `UserObjects`. Use them as stubs or empty boxes.
- Build the model that provides basic functionality.
- Add data inputs and outputs
- Test each `UserObject` individually.

Note that HP VEE does not support recursion. When you need multiple occurances of an object you must make multiple copies.

Screenshot - Top-down Design



Here is an example of a top-down design model.

Lab 4a—Damped Sinewave Generator

Task 1

Create a `UserObject` that generates a damped sine wave. The object's inputs are:

- Initial Amplitude
- Damping Factor
- Frequency
- Timespan
- Number of Points

Its output is a waveform.

Hint

Remember that the formula for a exponentially decreasing term involves the form e^{-kt}. Also, HP VEE provides the `exp(x)` object under the `Math` menu to perform the exponentiation function.

Note that HP VEE has an object that generates a log waveform for you.

(`Math --> Generate --> logRamp`)

Lab 4b—Random Noise Generator

Task

1. Create a `UserObject` that generates a random noise waveform.
2. Display the noise waveform and the noise spectrum.
3. Provide control for the following parameters:
   a. Amplitude
   b. number of points
   c. Interval (timespan)
   d. DC offset
4. Compare its performance to the built-in Noise Generator.

5

User Interaction

User Interaction

User Interaction

Definition – a user is someone who runs a model developed by someone else

- User Inputs
- Customization
- Panel Views
- Secure Models
- Combining Panels and UserObjects

E2100+24D V107

Lets look now at some features that HP VEE provides that help you interract with people using your model.

We'll discuss:

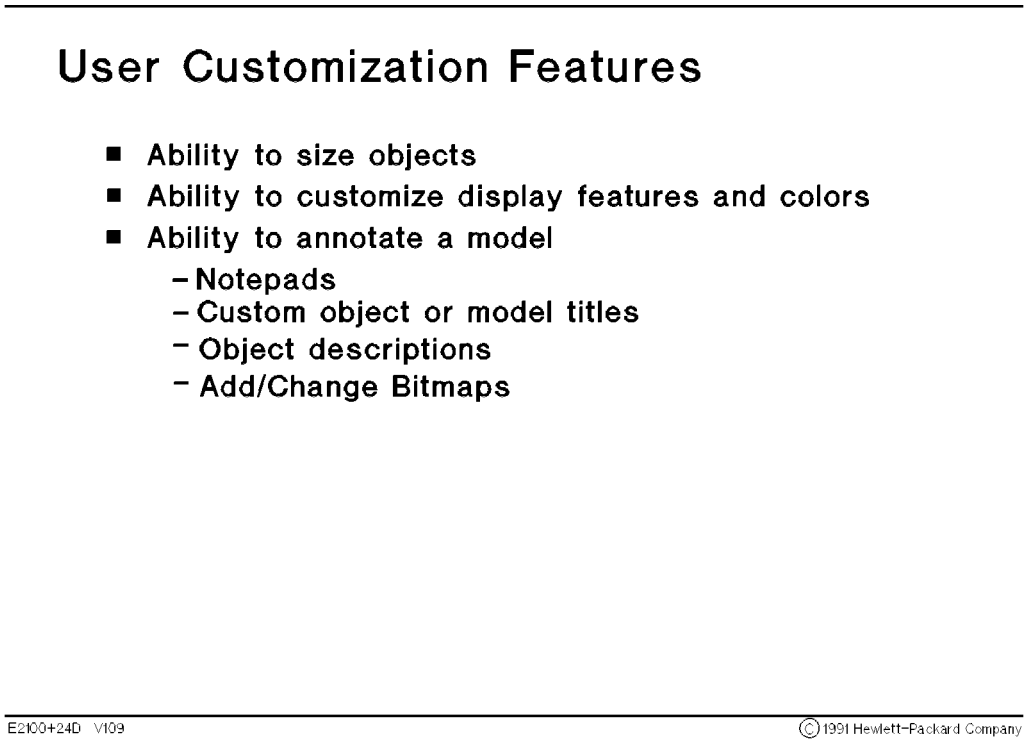
User Inputs	How the user enters data.
Customization	How you can customize the manner in which users interact with your model.
Panel Views	A custom interface for your users.
Secure Models	You can secure your model so that others cannot alter it.
Combine Panels and UserObjects	You can attach custom panel views to specific UserObjects.

User Inputs

- Enum, Toggle, Sliders, Constants, Dialog Box
  - Allow developer to prompt user for a variety of inputs
  - Each input object allows "AUTO EXECUTE"
  - Users input values without having to RE-START the model

E2100+24D V108

The user enters data into a model by using the `Enum`, `Toggle`, `Slider`, and `Constants` objects. These objects permit you to prompt the user for inputs. These objects also allow `Auto Execute`, which automatically propogates the data to other objects. Note that you don't need to stop and restart the model to enter data. It can be entered while the model is running.

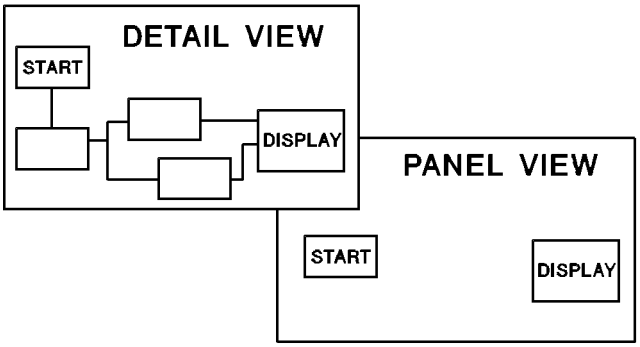


You can customize your model in many ways such as:

- Changing the size of objects to denote importance or to make them easier to use.
- Changing display features and colors.
- Anotating your model with user instructions or documentation for maintenance purposes. You can make annotations by:
  - Using `Notepad` objects write notes to yourself in the work area.
  - Editing object titles for further clarity of function or purpose.
  - Writing in the `Show Description` area of each object.
  - Changing or adding Bitmaps to the icon view of the object.

Panel View

- An alternate view of the model
- Developer chooses objects from the Detail View
- Data and sequence lines are not shown on the Panel View



E2100+24D V110

The panel view of a model provides another means of customizing your model. You choose which objects appear on the panel. The panel view does not show the interconnecting lines between objects. By using a panel view, you simplify and clarify the model seen by the user.

Panel Views

- Show only the objects necessary for operation
- Secure the model from user intervention
- Provide an easy to read interface to a complex model
- Improve performance by decreasing screen interaction

E2100+24D V111

With Panel Views you display only the objects necessary to operate your model. You can also secure your model behind the panel, thus preventing the user from altering the model. Again, the Panel View provides an easy to read interface for complex models. Panel Views also enhance performance by decreasing screen interaction.

Creating a Panel View

1. **Build the model and verify that it runs properly**
2. **Select the one or more objects you want to show on the panel**
3. **Select Edit – Add to Panel**
4. **Move and size objects on the panel to maximize its effectiveness**
5. **Press Panel and Detail to move between views**

E2100+24D V112

So how do you create a Panel View?

1. Create the model and verify that it works as desired.
2. Select the objects that you want to appear on the Panel View. These should be objects that the user interacts with or that display information for the user.
3. Select `Add to Panel` from the `Edit` menu.
4. Move and size the objects on the panel View to maximize its effectiveness for the user.
5. Use `Panel` and `Detail` buttons on the title bar to move between views.

Remember that changes to the Detail View *do not* affect the Panel View.

Panel View Characteristics

- **Fewer choices appear on the main menu in panel view**
- **If you cut an object on the detail view, its corresponding object on the panel is gone**
- **The appearance (size, location, etc.) is not shared between views**
- **Shared values include:**
  - **Initialize Values**
  - **Clear Values**
  - **Number Formats**
  - **Scaling**
  - **Etc.**

E2100+24D V113

Note that fewer choices appear on the main menu bar in the panel view. This prohibits the user from altering the operation of the model.

When you delete an object from the detail view, its corresponding object on the panel view is also deleted.

HP VEE does not share appearance characteristics between the panel and detail views. You can change the size or location in one without affecting the other. However, it does share the characteristics:

- Initial Values
- Clear Values
- Number Formats
- Scaling
- etc.

Securing a Panel View

- Creates a panel that does not allow a user to access the detail view
- 3 Step Process:
  1. Create the model and the panel view;
  2. Select Secure, and save the source file
     Both detail and panel views are available, yet the panel view can no longer be edited
  3. To remove access to the detail view, go to the panel view and save the model
     Be certain to select a unique name so that you don't overwrite the source file

E2100+24D V114

Securing a panel prevents the user from seeing the detail view. To secure a panel perform the following steps:

1. Create the model and the panel.
2. Select the `Secure` function from the `File` menu.
3. Save the source file.
4. Save the secured file using a *different* file name.

UserObjects With Panel Views

- A UserObject is an independent work area within an object
- The UserObject allows developers to create a panel view within the object
- Select objects within the UserObject and use the object menu – Edit to add them to a panel

E2100+24D V11.5

Remember that a `UserObject` is an independent work area within the main work area or another object. Each `UserObject` allows you to create a panel view associated with that individual object. As before, select the objects that you want to appear on the panel view, then select `Add to Panel` from the `UserObject`'s object menu or the pop-up `Edit` menu inside the `UserObject` work area. `Add to Panel` is context sensitive.

Using "Show Panel on Exec"

- Create a UserObject with a panel view
- Select the object menu, and select Show Panel on Exec.
- When the UserObject operates, the panel "Pops Up" on the work area

E2100+24D V116

The `UserObjects` also implement a unique feature with respect to the panel view. You can display the panel *only* while the `UserObject` is operating. To do this, select `Show Panel on Exec` from the `UserObject`'s object menu. Now the panel view associated with that `UserObject` appears in the work area when the `UserObject` is operating.

Show Panel on Exec

- When the UserObject operates, the panel opens up in the center of the work area
- The view closes when the UserObject finishes - so -
- To use this feature effectively - developers should use the Confirm (OK) object to pause execution until the user responds

When the `UserObject` operates, the panel view appears in the work area. It disappears when the `UserObject` finishes. Therefore, to use this feature effectively you should add a `Confirm (OK)` object to the panel view to pause execution until the user responds.

Lab 5—Create a Custom Dialog Box

Task

Create a `UserObject` which interacts with the operator. Use two inputs, A and B. If A and B are equal, send A to the output. If A and B are unequal, prompt the operator to select either A or B as the information displayed. If the operator does not choose within 10 seconds, generate an error.

Hint:

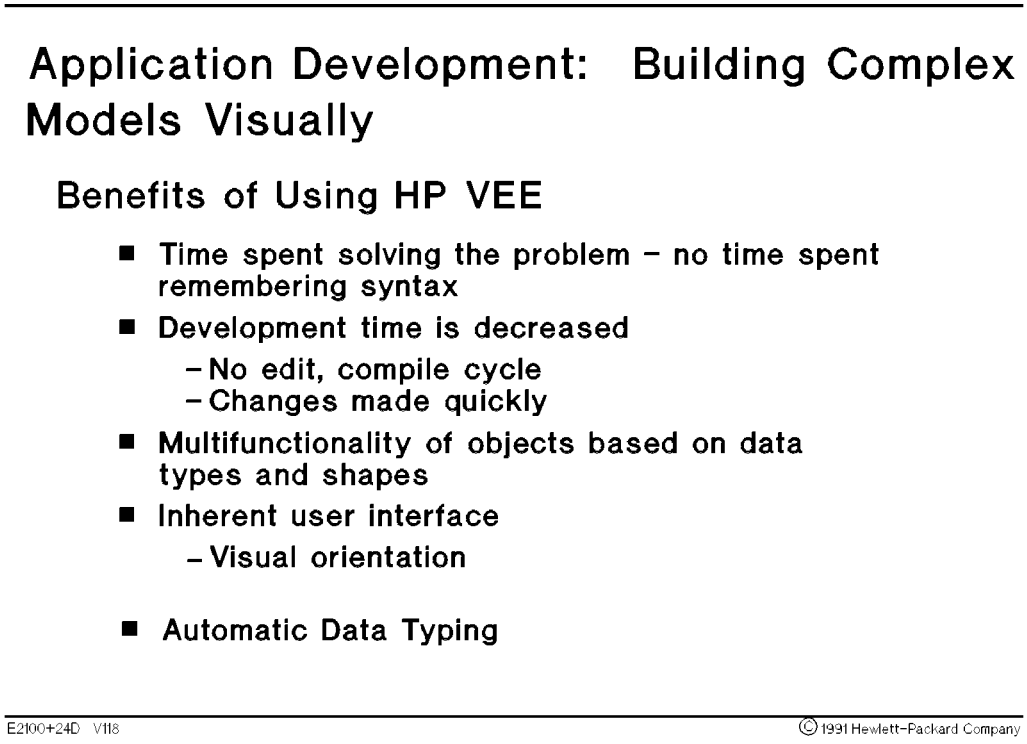
Each panel that "pops up" needs to be a separate `UserObject`. `UserObjects` may be nested!

Also, remember to enable, `Display on Exec` on each `UserObject` where you want the Panel View to pop into the workarea.

6

Application Development Techniques

Application Development Techniques



HP VEE helps you create solutions to your programming problems quickly and easily. You spend more of your time actually defining and describing your problem, rather than trying to find syntax errors or missing semicolons. You will also find that your development time is much shorter since there is no edit-compile cycle. You simply build your model on the screen and then execute it. The HP VEE paradigm also provides an inherent user interface.

The Visual Engineering Environment: Paradigm Shift

- The same, only different: programming languages
  - Highly cognitive, visually based, less abstract, more conceptual
  - One picture is worth a thousand lines (of code)
- Data flow easy here, control flow takes some thinking
- Remember block diagrams, sketches, flow charts
- In the end, hands-on experience facilitates the paradigm shift

E2100+24D V119

Developing models in HP VEE requires a paradigm shift on your part. You need to set aside for the moment all of the traditional programming that you have done in the past.

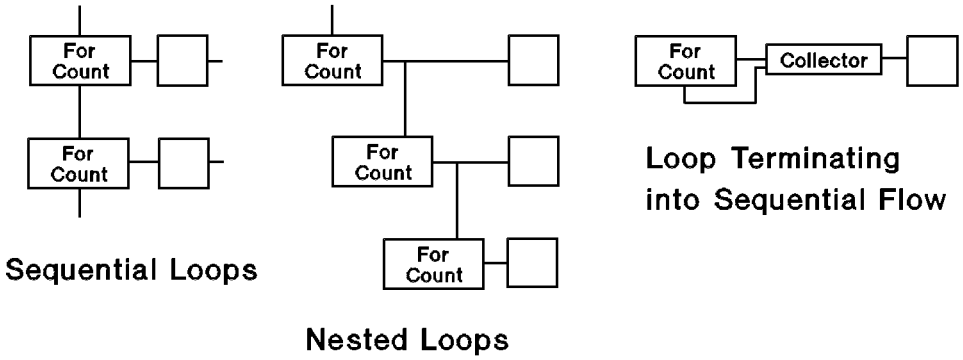
Change the way you think about the problem from "what happens next" to "what do I need to do to the data". Think in terms of Data flow instead of Control flow. Operate on blocks of data instead of individual datum.

Remember the block diagrams, sketches and flow charts that you used to originate ideas and designs. Use these same ideas to generate your HP VEE model.

One picture is worth a thousand lines of code.

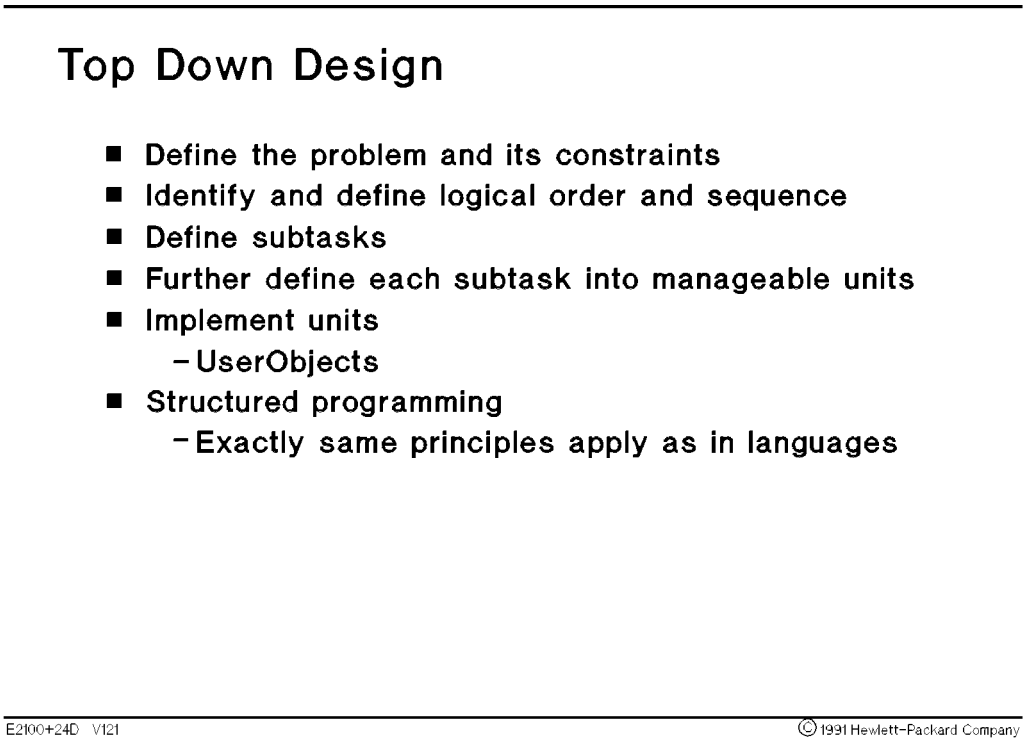
Think Visually, Spatially: Structures

- Subthread basic unit of work
  - Applicable to functionality of UserObject
- Structures of subthreads



E2100+24D V120

Remember to think spatially. An HP VEE subthread is the basic unit of your model. Each block in your block diagram becomes a `UserObject` in the HP VEE model. Here we illustrate how you structure subthreads for sequential and nested loops.



At this point, remember the points of top-down design which we discussed previously.

- Define the problem, including all of its contraints.
- Identify a logical order and sequence for each of the tasks within the problem.
- Define each subtask within each task. Note how well this fits into HP VEE's paradigm of nested `UserObjects`.
- Continue defining each subtask into manageble units.
- Now implement each of these tasks as a `UserObject` within HP VEE.

Complexities

- User interfaces
  - Like conventional programming, can be complex
  - Use of UserObjects, panel views, Show on Execute
  - Make the distinction between execution of user input and execution of the algorithms
- Optimization
  - Features of data, displays, number crunching
  - HP-UX escape, named pipes

E2100+24D V122

Another consideration for the model developer is the user interface. As with standard programming, you can create complex and useless interfaces with HP VEE. However, if remembering little tricks like, `Show on Execute` with Panel Views and careful, logical and orderly spatial layout of your model greatly enhance the usability and quality of your model.

To optimize performance remember:

- Iconify as many objects as possible. Displays on the objects require more processing time to update.
- Combine as many calculations as possible into a single `Formula` object. This reduces the actual calculation time.
- `Use HP-UX Escape`s to enhance performance with compiled library programs.

Levels of Complexity

- Visual Calculator
  - Simple, straight forward
  - More data flow, less control
  - No programmatic change
- Applications
  - Complexity equal to that of 1,000 lines or more programs
  - Lots of control flow: conditionals, programmatic change
  - Robustness: dealing with a user
  - Error handling
  - Different from test & measurement: Basic
  - Lots of subtasks

E2100+24D V123

As you look at your models, you will find that some models are simply "Visual Calculations", simple straight forward models.

However, larger models and applications require more complex models. By applying the principles discussed earlier, with careful thought with respect to robustness, the user interface and error handling you will create useful models quickly and with ease.

Beginnings and Endings

- Run vs. Start vs. Auto Execute
  - All not necessarily the same for given model
- User interaction
  - Fill–in, then execute
  - Execution and data input
  - The toggle object
- Control of execution
  - The OK object
- When is the model terminated
  - Implicit vs. explicit

E2100+24D V124

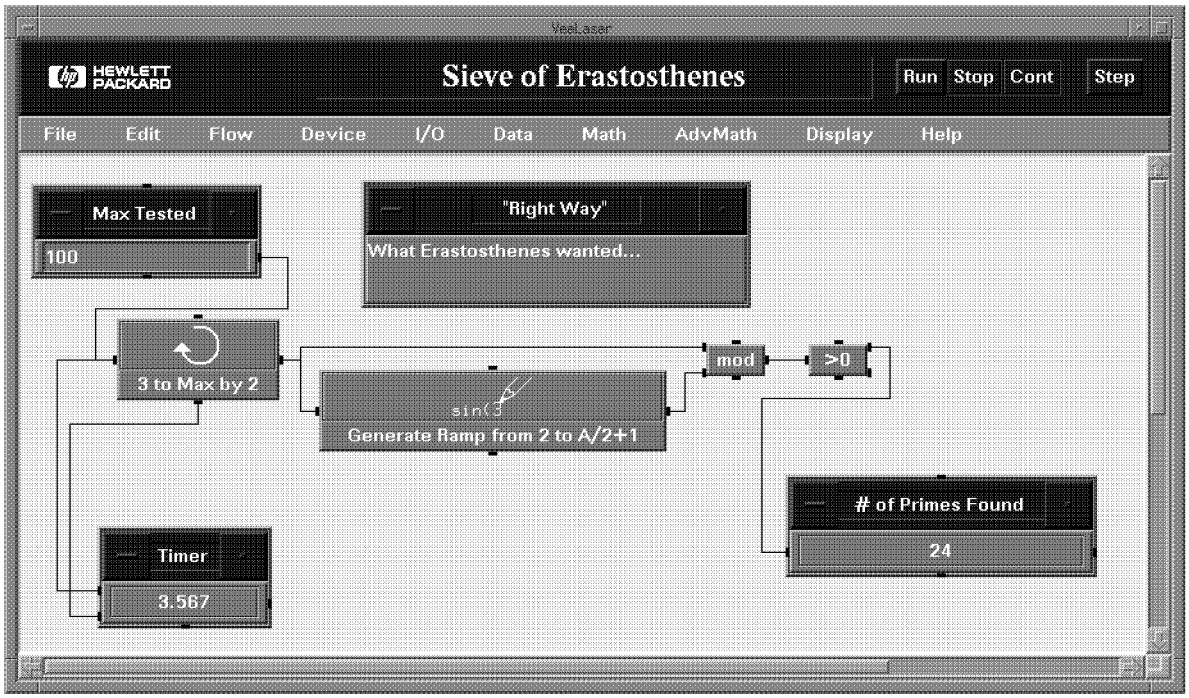
You also need to consider the beginnings and endings of models.

Remember that `Run`, `Start` and `Auto Execute` affect models differently.

How will you handle user interaction? Entry boxes, `Toggle` objects, `Confirm (Ok)` objects.

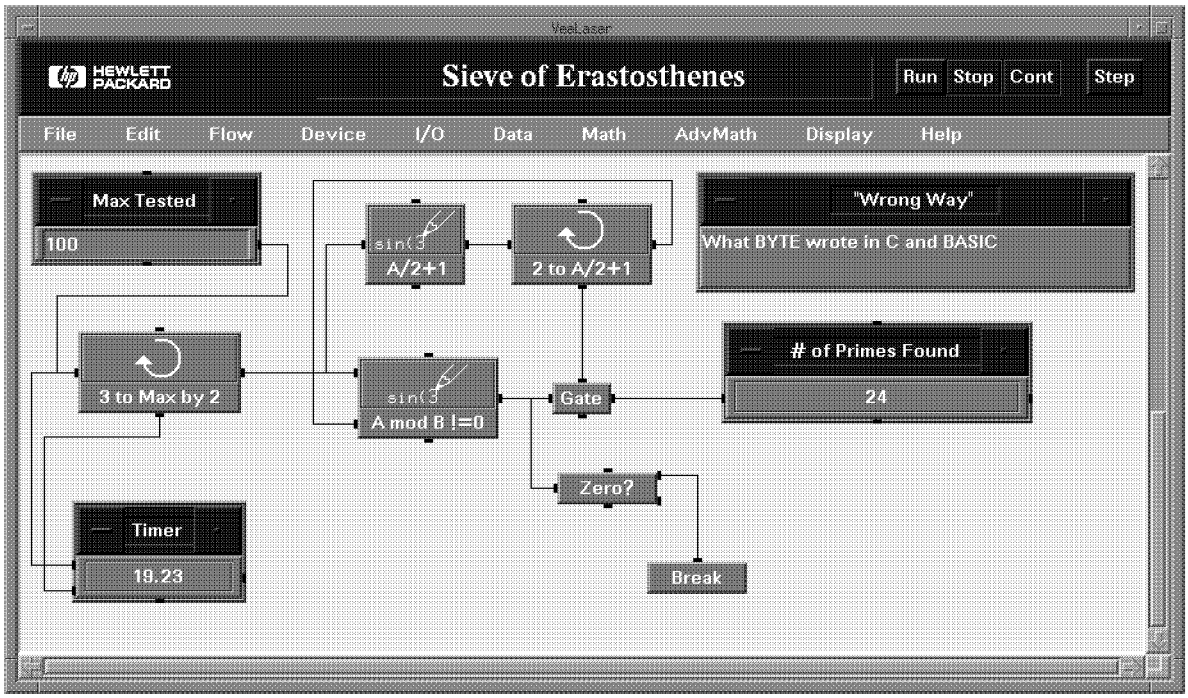
How do you terminate the model? Will it exit cleanly or leave the user wondering what's happening?

Screenshot sieve

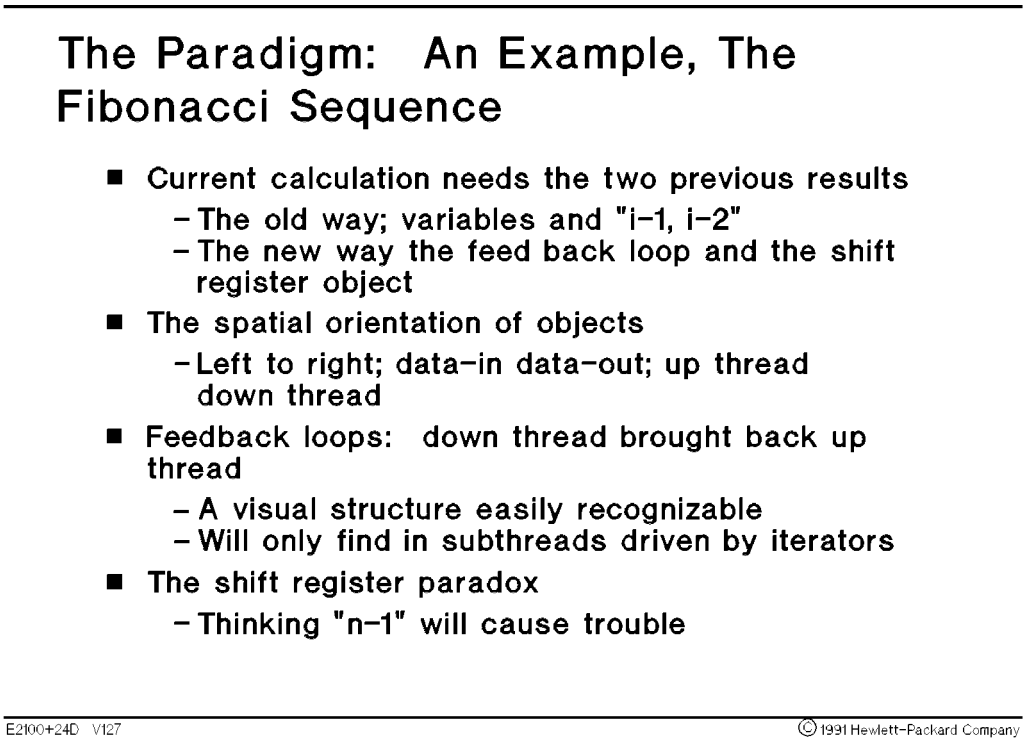


Here's an example of a well designed HP VEE model versus a model designed using an old programming paradigm. Notice that in this model large amounts of data flow between each object in a single container. The model on the next page shows a model using the old programming paradigms.

Screenshot sieve2



This model operates on each individual datum. Note the difference in execution speed as well as the additional complexity of the model. The model on the previous page uses one `Formula` object to generate a container of data for comparison. This model generates each datum, one at a time.



To help you see the differences and practice developing an application, lets look at the Fibonacci Sequence model, a classical programming problem.

The Fibonacci sequence is: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

This is simply the sequence that occurs by adding the previous two numbers together to get the next number in the sequence. With "old-style" programming, we use variables and subscripts `i-1` and `i-2`. But with HP VEE you simply use a feedback loop that keeps track of the previous values.

Just some further notes. You will only find feedback in subthreads driven by iterating objects (`Repeat`). And don't try to think of this in the old `n-1` paradigm. Its a new way of thinking for programs. It harks back to the old days of engineering.

Lab 6a—Model Building Techniques

Background

The Fibonacci Number sequence is a series of numbers such that any given number in the sequence is the sum of the last two numbers. Hence, the first 10 Fibonacci numbers are:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55

Task

Create a model that computes fibonacci numbers. The model requires just one input parameter—how many numbers in the sequence to compute. The output is in the form of a one-dimensional array.

Suggested Objects

Important objects to the solution are:

- `+` object
- `Shift Register`
- `Collector`
- `Junction`

Hint:

Use data feedback loops.

Lab 6b—Model Building Techniques

Task

Create a model that generates an array of random numbers, each of which is between 1 and 10. The sum of all the numbers in the array should be less than or equal to 100. This problem illustrates the use of the `Until Break` iterative loop.

7

I/O Transactions and Data Formatting

I/O Transactions and Data Formatting

I/O Transactions and Data Formatting

Purpose – To provide communication paths to:

- **"Standard I/O" facilities**
- **The file system**
- **Line printer spooler**
- **Strings**

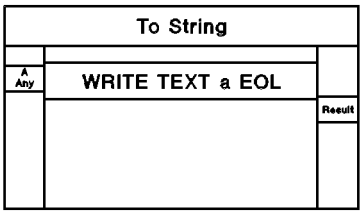
E2100+24D V128

HP VEE uses transactions and data formatting to provide:

- "Standard I/O" facilities or communication with unix `stdin` and `stdout`.
- Ability to read and write files.
- Access to the printer spooler.
- Ability to format strings.

I/O Transactions

- **All communication paths are implemented as Transaction Objects**
- **Individual transactions handle multiple data items**



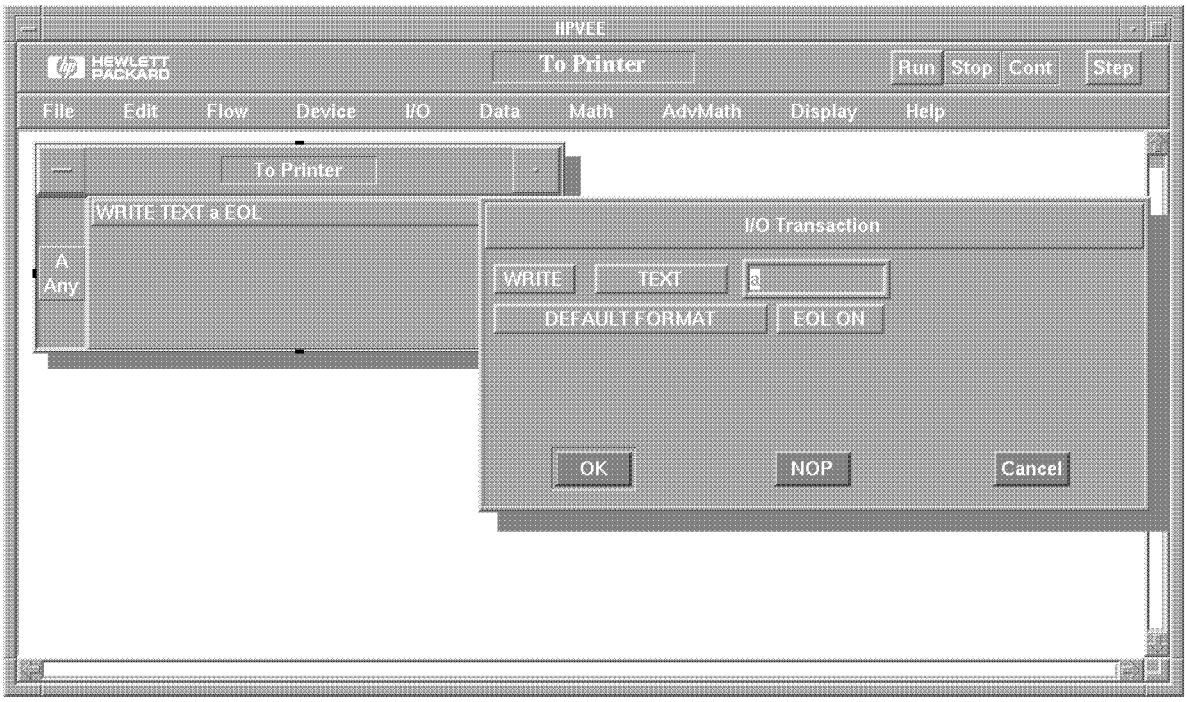
E2100+24D V129

All of the communications paths between HP VEE and other HP-UX resources go through transaction objects. For example, this slide shows a `To String` transaction object.

Note that each object may have multiple transactions contained within it, ie., multiple reads and writes.

Also note that each transaction may handle multiple data items. For example you can read in a single coordinate pair, or an array of coordinate pairs as a single transaction.

Screenshot - To Printer



This screen shot illustrates the `To Printer` object. You can open each transaction to modify it for your particular operation.

Transactions

- Specify action
  - READ, WRITE, EXECUTE, WAIT
- Specify encoding (interpretation) of data
  - TEXT, BYTE, CASE for data being written
  - TEXT, BINARY, BINBLOCK, CONTAINER for data being read
- Specify formatting of data
  - Numerics represented as REAL, INTEGER HEX, OCTAL
  - Full control of field width, justification

E2100+24D V131

You can specify the following actions in a transaction box:

- `READ`—Read data into HP VEE from another resource
- `WRITE`—Write data out of HP VEE to another resource
- `EXECUTE`—Performs operations on the resource
- `WAIT`—Waits the specified number of seconds before performing the next transaction

You can specify the following data encodings in a transaction box:

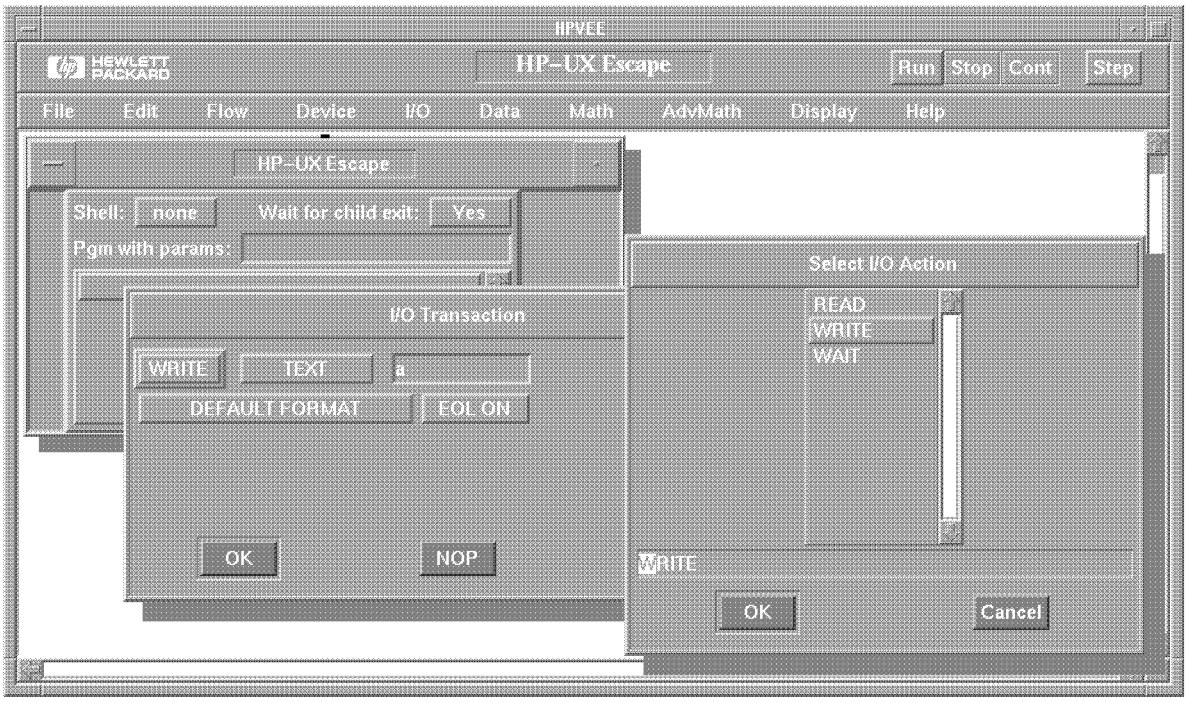
- `TEXT`—(Read or Write)
- `BYTE`—(Wrie Only)
- `CASE`—(Write Only)
- `BINARY`—(Read Only)
- `BINBLOCK`—(Read Only)
- `CONTAINER`—(Read Only)

We'll explain these in detail in just a moment.

You can specify the following data formats in a transaction box (with full control of field width and justification):

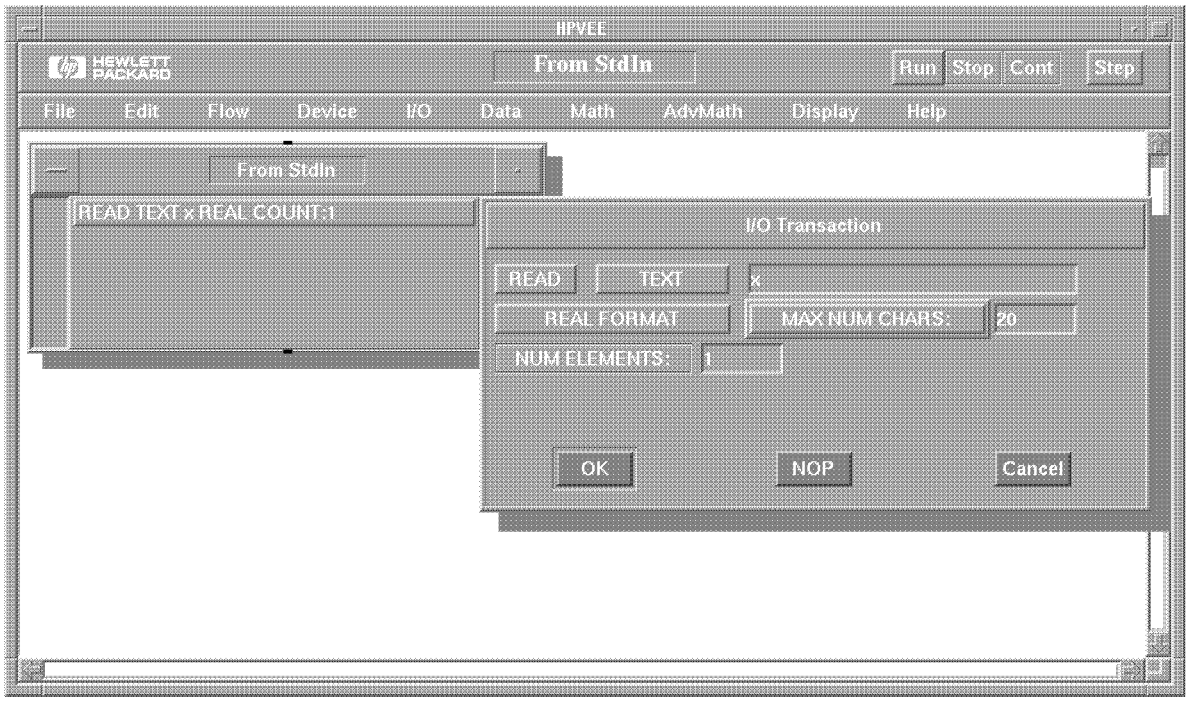
- `REAL`
- `INTEGER`
- `HEX`
- `OCTAL`

Screenshot - HP-UX Escape

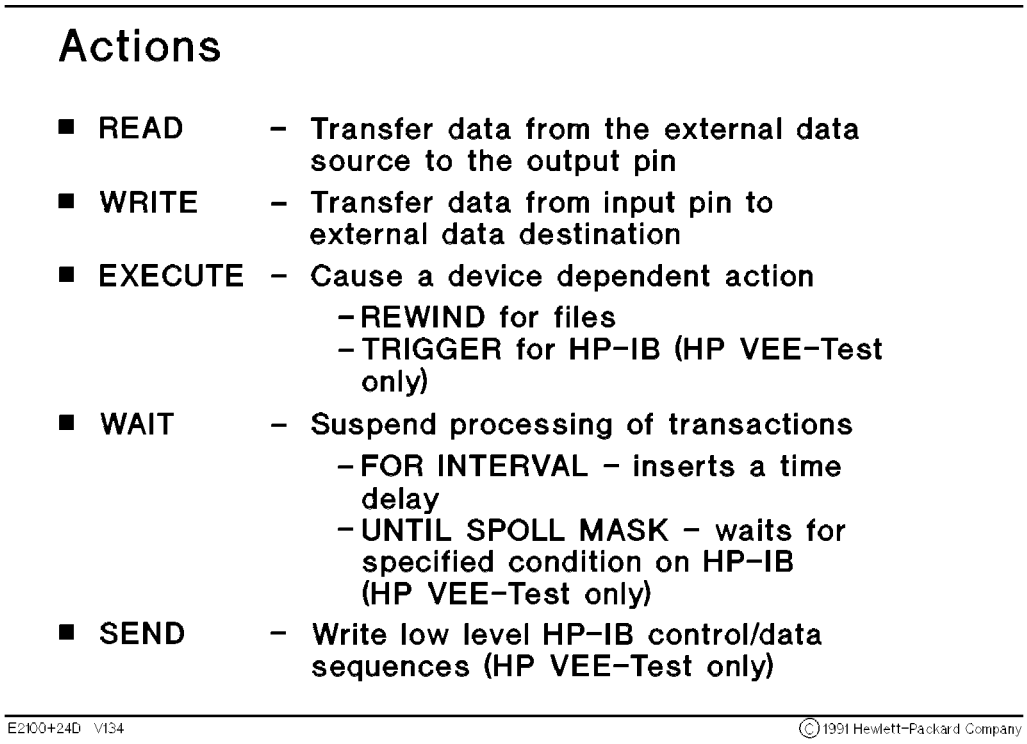


This screen shot illustrates selecting an action for an `HP-UX Escape` object.

Screenshot - From StdIn

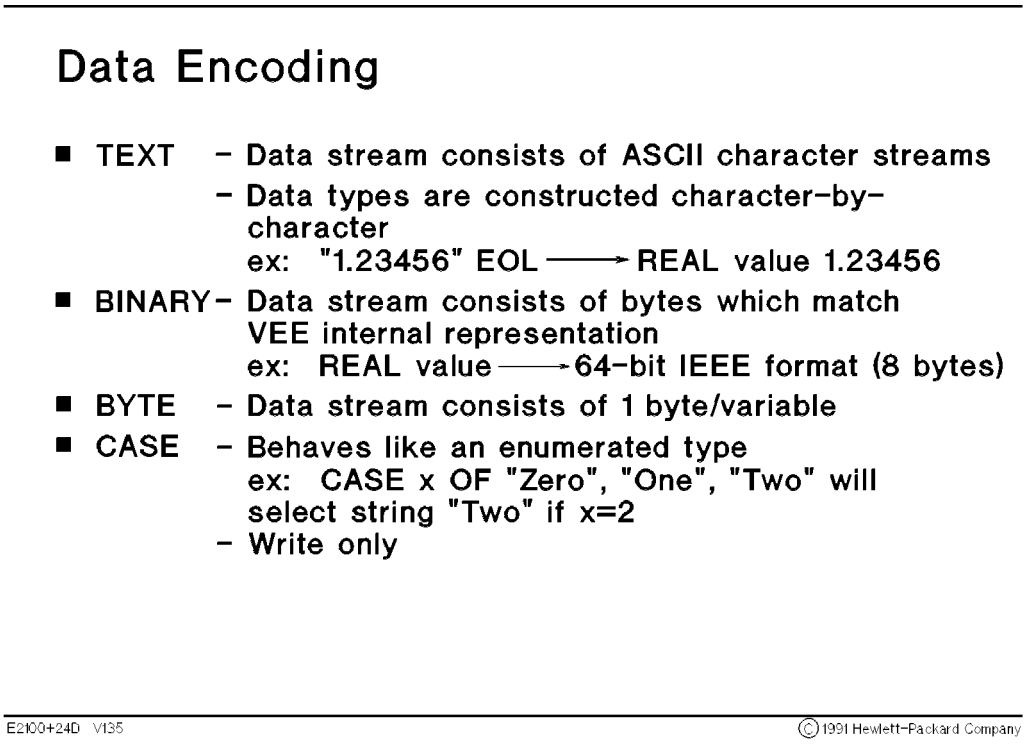


This screen shot illustrates reading text from a `From StdIn` object.



You can specify the following actions in a transaction box:

- READ—Read data into HP VEE from another resource
- WRITE—Write data out of HP VEE to another resource
- EXECUTE—Performs a device-dependent operation on the resource
  - REWIND files to the beginning
  - TRIGGER HP-IB command (HP VEE-Test only)
- WAIT—Waits the specified number of seconds before performing the next transaction
  - FOR INTERVAL inserts a time delay
  - UNTIL SPOLL MASK waits for the specified serial poll condition on HP-IB (HP VEE-Test only)
- SEND—Write a low level HP-IB control or data sequence (HP VEE-Test only)



You can specify the following data encodings in a transaction box:

- `TEXT`—A data stream of ASCII characters. Data types are constructed character by character. For example, the data stream `"1.23456" EOL` becomes the REAL value 1.23456.
- `BINARY`—A data stream of bytes which match the HP VEE internal representation. For example, a REAL value will have the IEEE 754 64-bit data format, which uses 8 bytes.
- `BYTE`—A data stream consisting of one byte per variable.
- `CASE`—A data stream that behaves like an enumerated type. For example, the statement `CASE x OF "Zero", "One", "Two"` will select the string `Two` if `x` equals 2.

  Note that this works only for `WRITE` transactions.

Data Encoding

- **BINBLOCK** – Data stream is sent as IEEE – 488.2 indefinite length block
  - A "#" character
  - A digit specifying the length of the length field
  - The length field specifying the number of bytes to follow

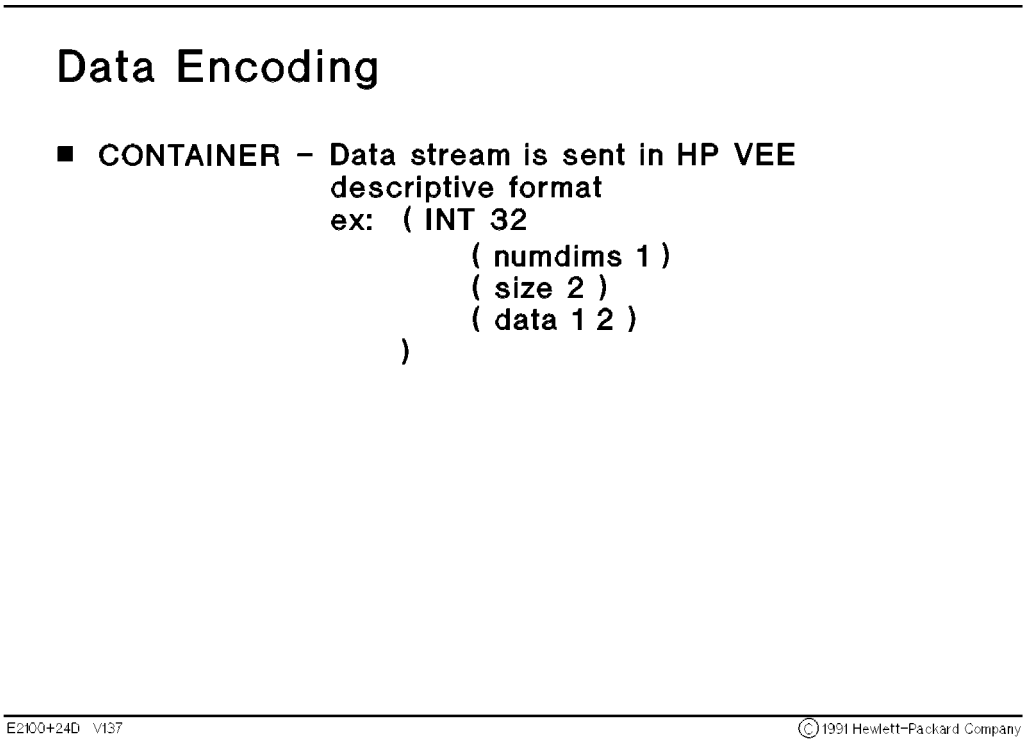
  ex: #12AB = a 1 digit length digit
  length = 2
  data = AB
  #2101234567890 = a 2 digit length
  length = 10
  data = 1234567890

E2100+24D V136

- `BINBLOCK`—A data stream sent as an IEEE-488.2 indefinite length block. This data stream consists of the following elements:
  - `#` character
  - `digit` specifying the length of the length field
  - `length field` specifying the number of bytes that will follow
  - `data`

  For example, `#12AB` defines, one length digit, length of two, and data equal to `AB`.

  Another example, `#2101234567890` defines, two length digits, length of ten, and data equal to `1234567890`.

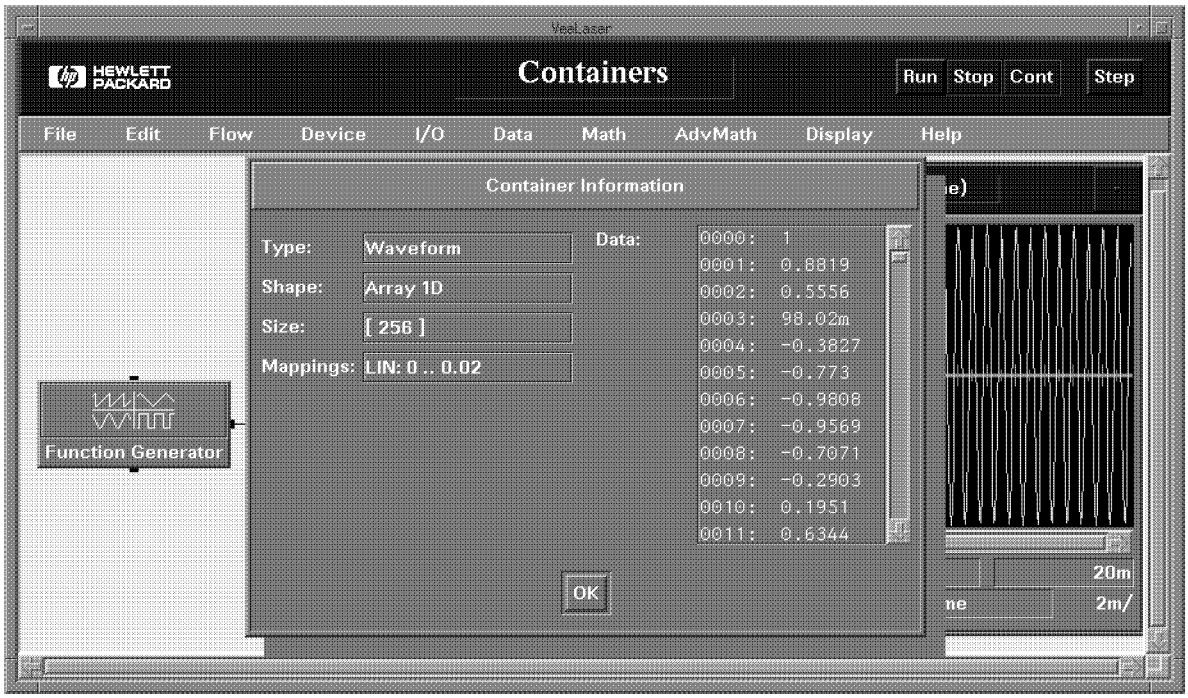


`CONTAINER`—A data stream in HP VEE descriptive format.

For example:

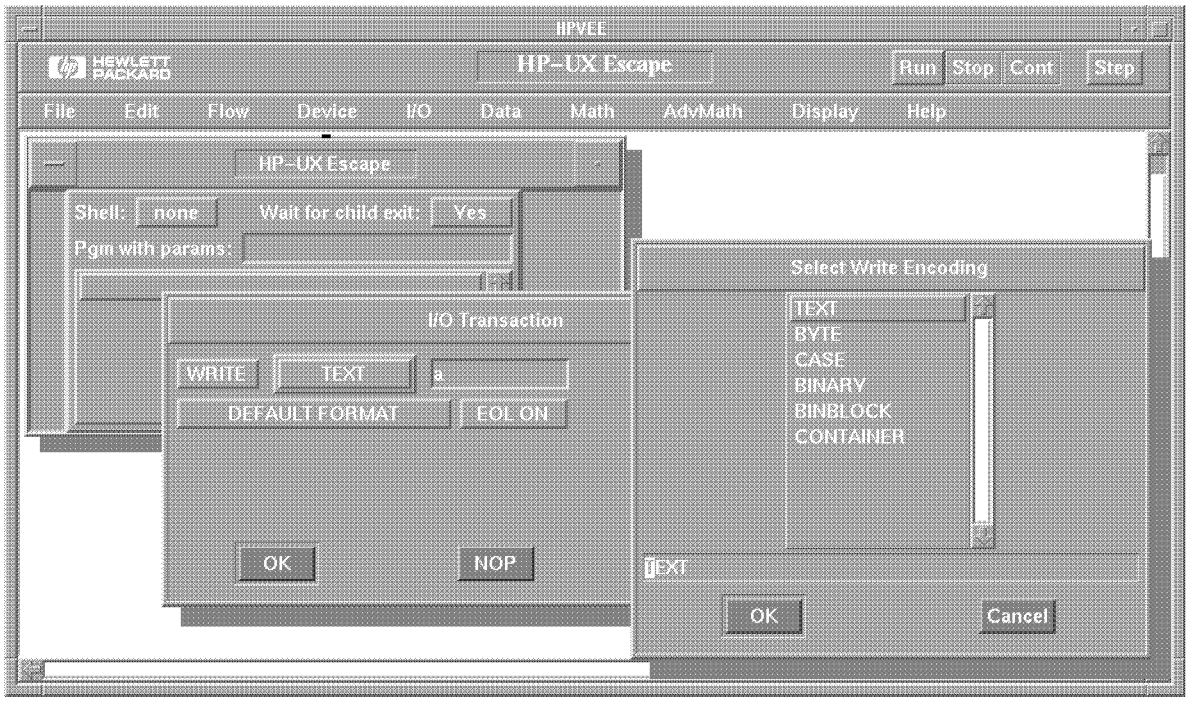
```
(INT 32
    ( numdims 1 )
    ( size 2 )
    ( data 1 2 )
)
```

Screenshot - Containers

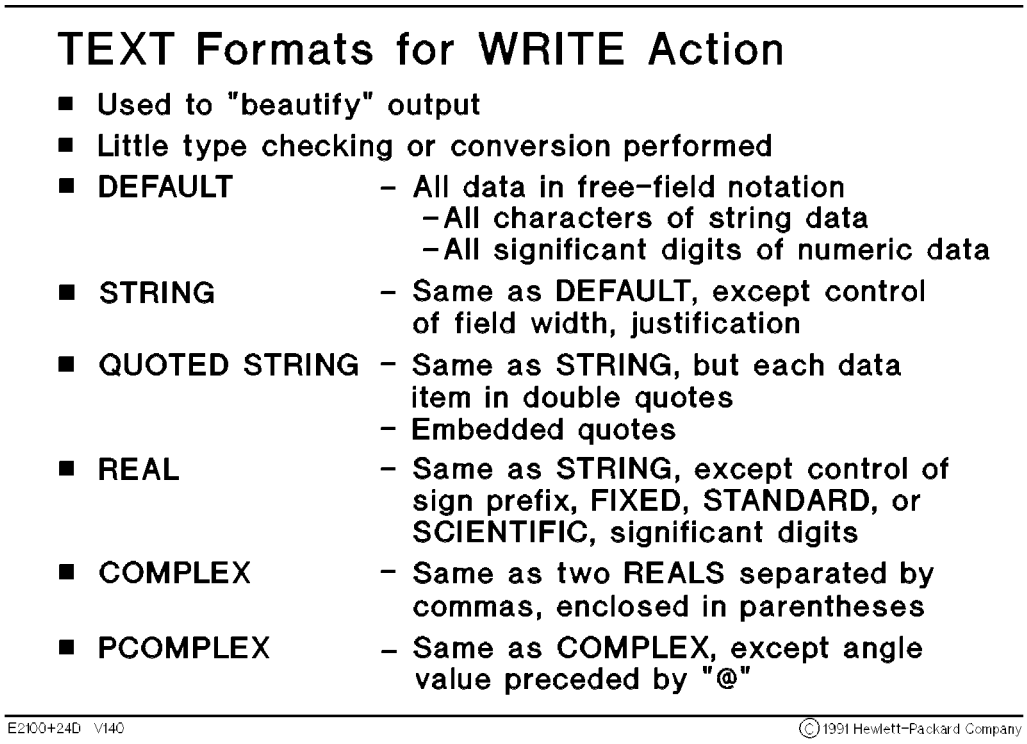


Here's what a container looks like, using `Line Probe`. Notice that it lists the data type, shape and mappings as well as listing the actual data.

Screenshot - HP-UX Escape TEXT



This screen shot illustrates selecting a write encoding data format.



These are the `TEXT` formats available for a `WRITE` action in a transaction box. You use them to "beautify" your output data. Note that very little type checking or conversion takes place. It simply formats your data.

`DEFAULT` All data is in a free-field notation. It includes all characters of string data or all significant digits of numeric data.

`STRING` All data is in a free-field notation, as in the DEFAULT format. However, with this format you can control the field width and justification (left or right).

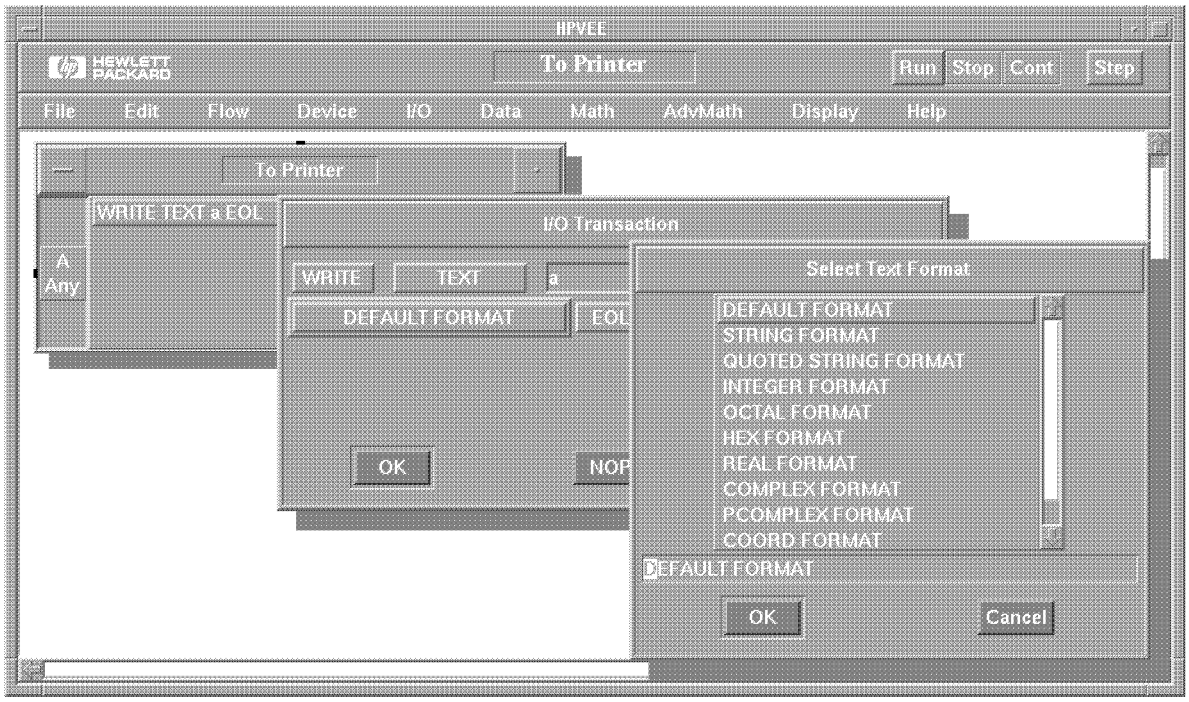
`QUOTED STRING` This format is the same as STRING format, except that all data is enclosed in double quotes. It also handles data with embedded quotes.

`REAL` All data is in a free-field notation, as the above formats. You can control the sign prefix and number of significant digits. You can also designate `FIXED`, `STANDARD`, or `SCIENTIFIC` notation formats for the data.

`COMPLEX` This format appears the same as two `REAL`s separated by a comma.

`PCOMPLEX` This format appears the same as `COMPLEX`, except that the angle value is preceded by a `@`.

Screenshot - To Printer



This screen shot illustrates selecting a `TEXT` encoding data format.

TEXT Formats for READ Action

- Match input data stream to required values and types
- Data conversion enforced
- Output pins take on type and shape required
- CHAR – Reads specified number of characters
  – Stored in string
- TOKEN – Allows multiple strings to be entered from data stream
  - SPACE DELIM – strings are separated by spaces
  - INCLUDE CHARACTERS – strings delimited by any non-member of set
  - EXCLUDE CHARACTERS – strings delimited by any member of set
- STRING – Reads all characters up to a specified limit

E2100+24D V142

You also specify `TEXT` formats for `READ` actions. HP VEE then matches the input data stream to the required values and types. It enforces the data conversion specified. The object's output pin then takes on the type and shape of the data.

`CHAR` This format reads the specified number of characters and stores it as a string.

`TOKEN` This format allows you to read several strings from a data stream.

- `SPACE DELIM` specifies that a space separates each string.
- `INCLUDE CHARACTERS` specifies that any character not specified in the set separates each string.
- `EXCLUDE CHARACTERS` specifies that any character in the specified set separates each string.

`STRING` This format reads all characters up to the number specified.

The Number Builder

- When numeric format is imposed on TEXT data stream, "number builder" attempts to extract numeric value from data
- Data is skipped while looking for numeric character
- Data is used by builder until EOL or non-numeric encountered
- Number is built
- Numeric means
  - 0-7 for OCTAL
  - 0-9, a-f, A-F for HEX
  - 0-9 for INTEGER
  - +, -, 0-9, e, E, decimal point for REAL

E2100+24D V143

HP VEE uses a Number Builder to extract a numeric value from textual data when you specify a numeric format for `READ` action data. The number builder works in the following manner:

1. Skip data until it encounters a numeric character.
2. Build value from data until it encounters an `EOL` or non-numeric character.
3. Build the final number.

Valid numeric characters are:

- `OCTAL`: 0-7
- `HEX`: 0-9, a-f, A-F
- `INTEGER`: 0-9
- `REAL`: `+`, `-`, 0-9, `e`, `E`, . (decimal point)

Text Formats for READ - Numeric

- OCTAL, HEXADECIMAL, INTEGER – Attempt to build INT32 value from numeric data received
  - OCTAL accepts 0..7
  - HEX accepts 0..9, a–f, A–F
  - INTEGER accepts 0..9
- REAL – Builds REAL64 value
  - Accepts 0..9, +, –, e, E, . (decimal point)
- COMPLEX – Expects two REAL values
- PCOMPLEX – As COMPLEX, except must specify RAD, DEG, GRAD to interpret angle
- COORD – Expects specified number of REAL values

E2100+24D V144

The text formats for numeric `READ` actions are:

`OCTAL`	build an `Int32` from received numeric data.
`HEXADECIMAL`	build an `Int32` from received numeric data.
`INTEGER`	build an `Int32` from received numeric data.
`REAL`	build a `Real` (64-bit) from received numeric data.
`COMPLEX`	build two `Reals` (64-bit) from received numeric data.
`PCOMPLEX`	build two `Reals` (64-bit) from received numeric data. Remember to specify `RAD`, `DEG`, or `GRAD` to interpret the angle correctly.
`COORD`	build the specified number of `Reals` (64-bit) from received numeric data.

8

Files and Standard I/O

Files and Standard I/O

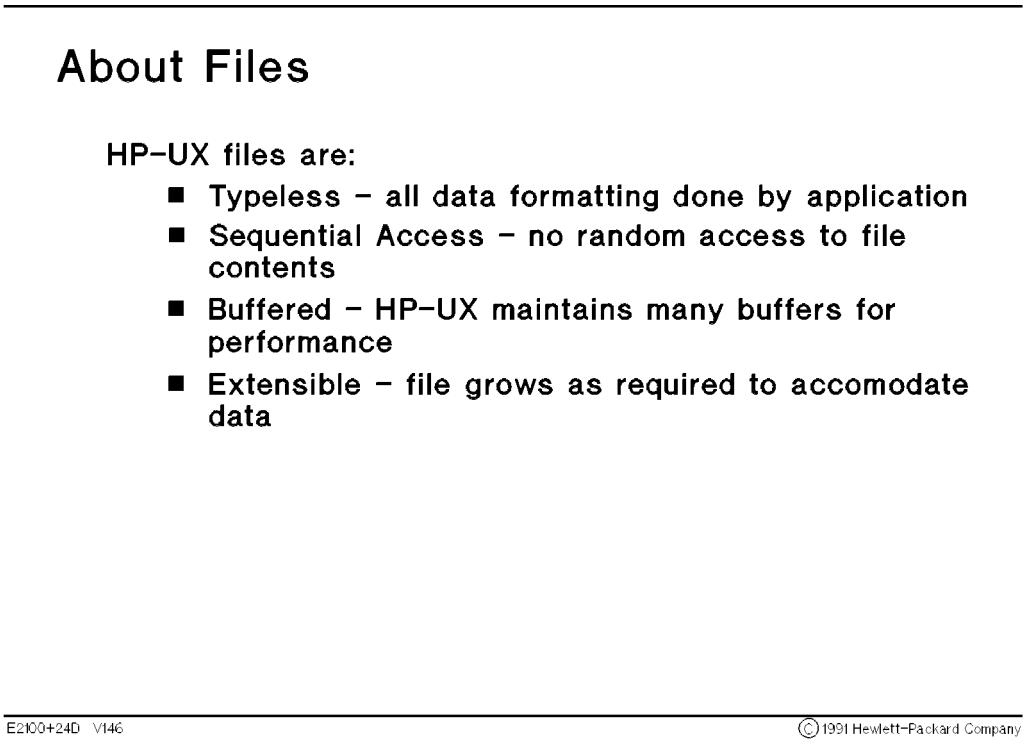
Communication with File System

Purpose:

- **Storing and retrieving data from other programs**
- **"Permanent" data archival**
- **Simple communication with other processes**

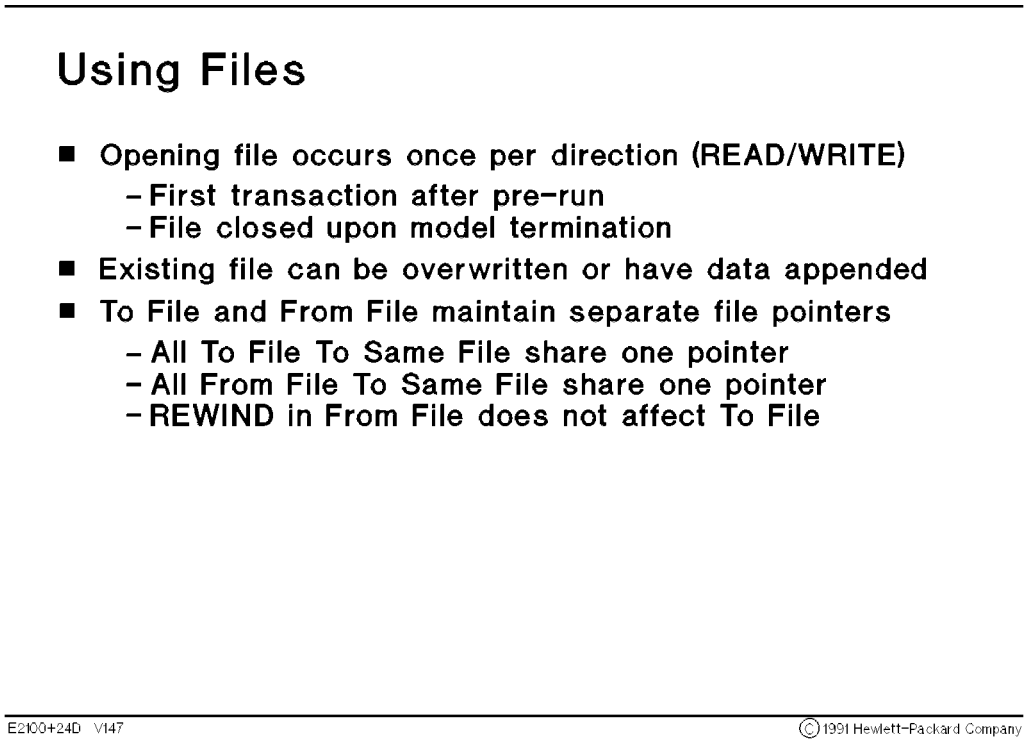
E2100+24D V145

HP VEE provides objects for you to communicate data with the “outside world”. That way you can share information with other programs, or simply archive data generated by your model.



Here are some thing to remember about HP-UX files:

- They are typeless. The program reading or writing the data performs all data formatting.
- They provide ONLY sequential access, not random access to their contents.
- The operating system buffers all file transfers to improve performance.
- Files are extensible, meaning that they grow in size as require to hold.



When you Use Files remember:

- HP VEE opens each file once per direction (`READ` or `WRITE`).
  - The first file object that operates after PreRun opens the file.
  - HP VEE closes each file when the model finishes.
- File objects either overwrite or append data to existing files.
- `To File` and `From File` objects maintain separate file pointers. However, you should note that:
  - All `To File` objects that use the same file share one file pointer.
  - All `From File` objects that use the same file share one file pointer.
  - A `REWIND` command from a `From File` object does not affect the file pointer of a `To File` object using the same file.

File I/O Transactions

- To File and From File support two EXECUTE commands
  - REWIND – All further READ or WRITE operations start at beginning of file
    – Cannot use in To File open in APPEND mode
  - CLEAR – Useful only in To File in OVERWRITE mode
    – Resets file to zero length (erases old data)

E2100+24D V148

`To File` and `From File` support two `EXECUTE` commands to help you work with files:

`REWIND`	Move file pointer to beginning of file. All further `READ` or `WRITE` operations will be from that point. Note that you cannot use `REWIND` on a file opened in `APPEND` mode.
`CLEAR`	Resets file to zero length, erasing all old data. Note that you cannot use this command with a file opened in `OVERWRITE` mode.

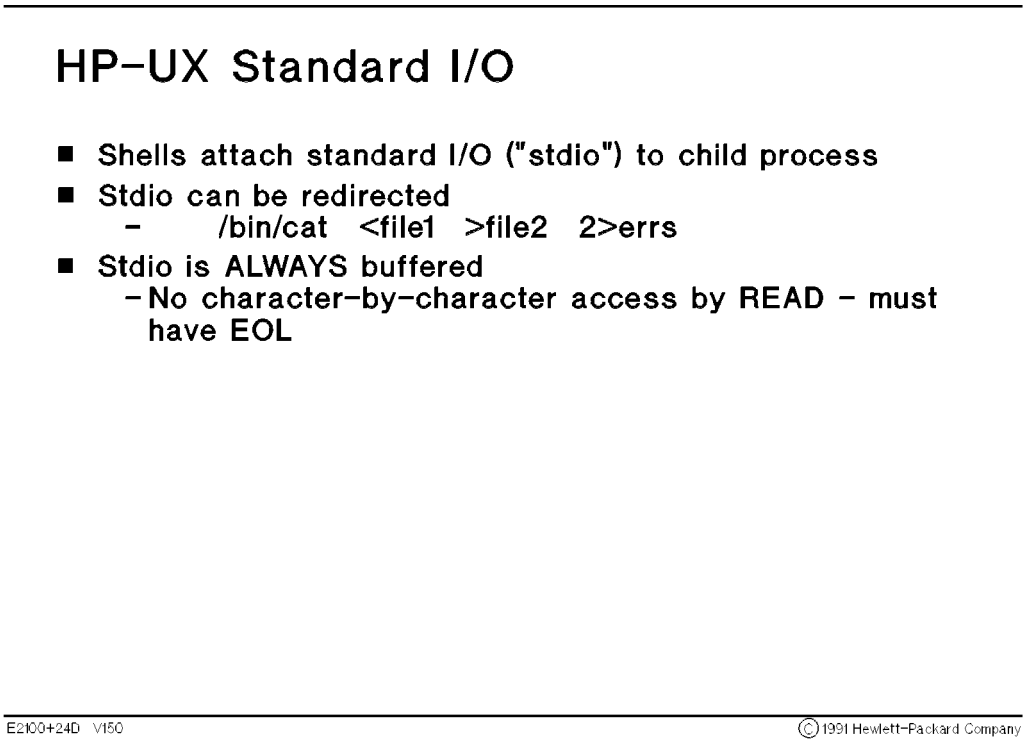
HP-UX Standard I/O

- HP-UX associates 3 communication paths per process
  - Standard Input ("stdin")
    - Normally the keyboard
  - Standard Output ("stdout")
    - Normally the display
  - Standard Error ("stderr")
    - Normally the display

E2100+24D V149

HP-UX associates three standard communications paths with each process that it runs. These paths are:

Standard Input (`stdin`)	Provides input data to the process. Normally associated with the keyboard.
Standard Output (`stdout`)	Receives output data from the process. Normally associated with the display.
Standard Error (`stderr`)	Receives error information from the process. Normally associated with the display.



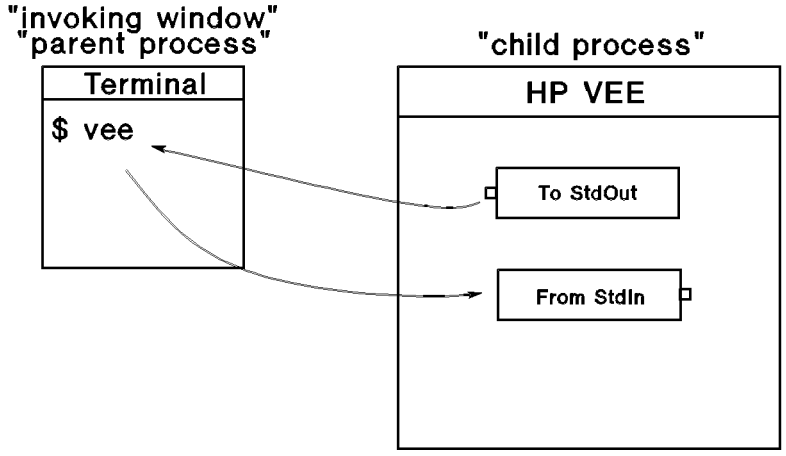
HP-UX shells attach these standard I/O ("stdio") paths to a child process. You can redirect data to the child process from files or other sources through these paths.

The command `/bin/cat <file1 >file2 2>errs` sends data to the program `/bin/cat` from file `file1`, receives data from `cat` and writes it into `file2` and writes any errors in the file `errs`.

Remember that `stdio` is always buffered. You cannot write one character at a time. You must send an entire line of data terminated with an `EOL` character.

Communication via Standard I/O

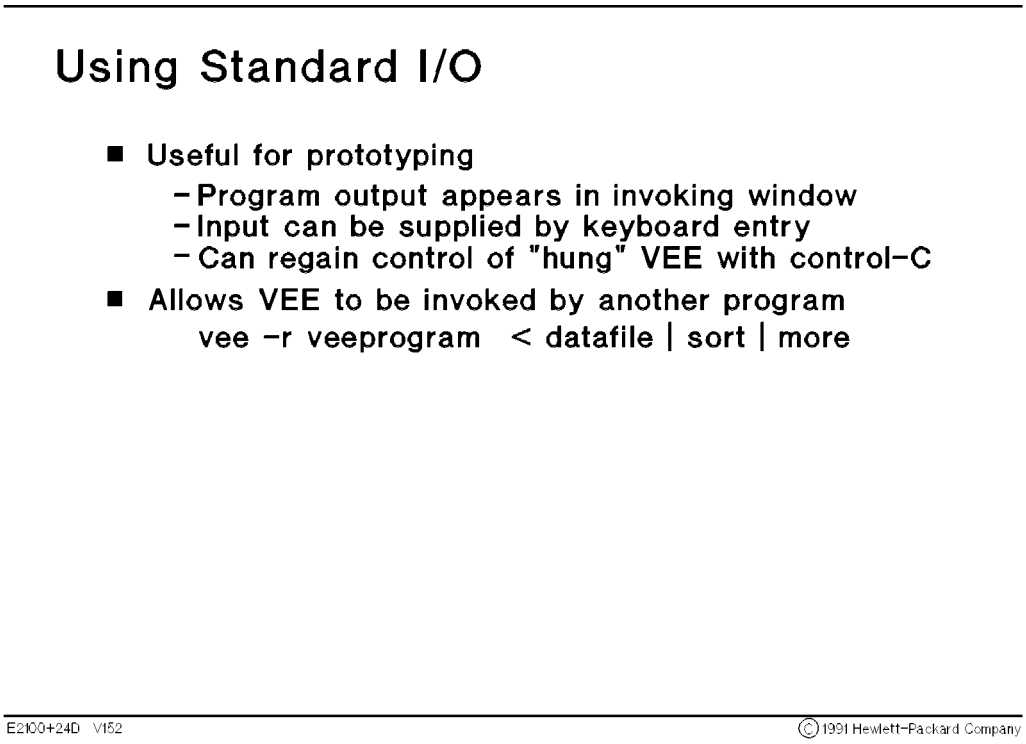
- **When VEE is invoked from terminal window, stdio is passed to VEE**



E2100+24D V151

When you run HP VEE from a window, HP-UX attaches `stdio` to that window. Therefore, the `To StdOut` object writes data back to the window, and the `From StdIn` object reads data from that window.

Note When you run HP VEE in background mode in HP-UX, there is *no* connection to standard I/O. In other words, the `To StdOut` and `From StdIn` objects will *not* operate.



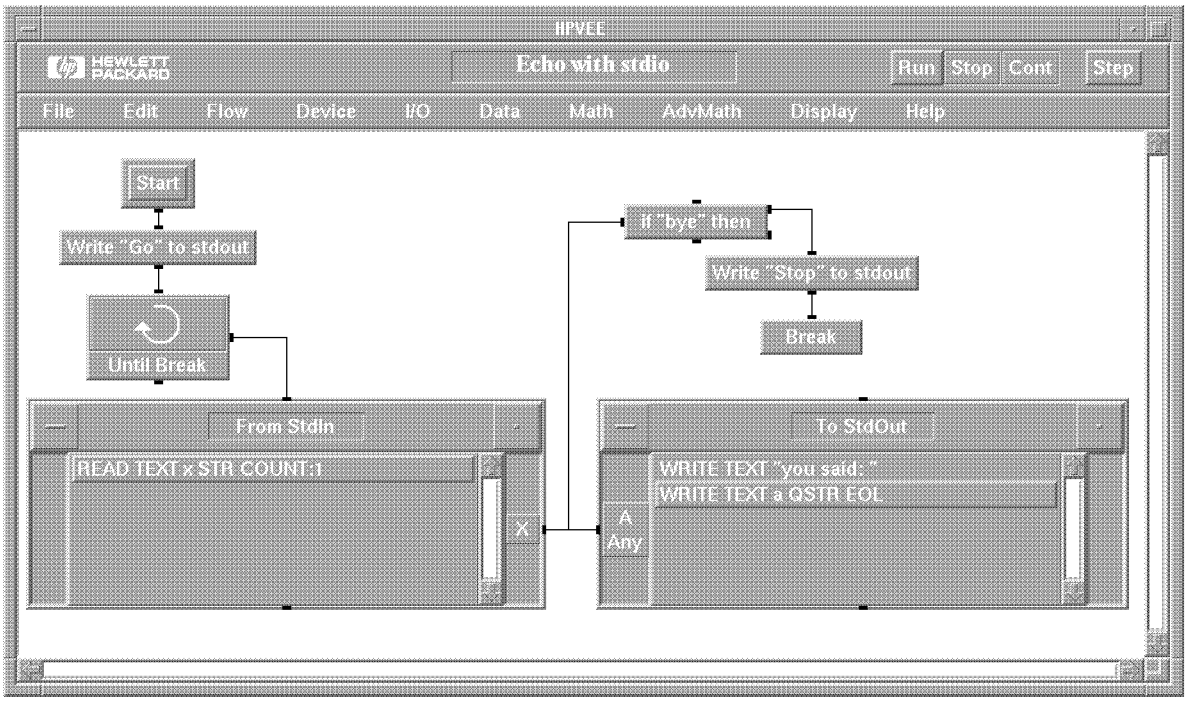
Standard I/O proves to be very useful when prototyping HP VEE models. The model can write data to the invoking window. You can type data on the keyboard for entry into HP VEE. You can also regain control of HP VEE with Ctrl-C when it hangs.

You could also call HP VEE from another program. For example,

```
veeengine -r veeprogram < datafile | sort | more
```

calls HP VEE, starting the model `veeprogram`. It then sends in data from `datafile`.

Screenshot - Echo with stdio



Here is an example of the use of standard I/O.

Note `From StdOut` doesn't work if you run HP VEE in background mode on HP-UX.

Lab 7a—Communicating with HP VEE

Task

Build a model that sends an initial prompt to the invoking terminal, then echoes back each response in quotes. Have this model read and echo continuously until the word `bye` is entered.

Hint:

Use `veetest -r [LABNAME] -iconic`

Lab 7b—Using Files for Fun & Profit

Background

Knowing that you are all highly motivated, patriotic, taxpaying, loyal employees, we thought that you'd like to plot your profits for the last 30 years. We've provided a set of hypothetical profit percentages in a file called `/labs/profit.dat`. Read the data from the file and plot it on your display. Since you'd also like to know where to invest your money, you also want to calculate a five year moving average of this data. Remember, data in the file is individual percentages for each year. The data has the format of YEAR PERCENTAGE, for example the 1990 data would be `90 4.33`.

Tasks

Build an HP VEE model that:

1. opens the file `profit.dat`
2. reads all of the data from the file
3. calculates a five-year moving average of the data
4. transforms the data into a "graphable form"
5. plots the data and moving average on a display.

Hints

Read the data from the file as a `COORD` data type.

Use the `movingAvg` object (found under the `Data Filtering` menu in the `AdvMath` menu.)

Lab 7c—To File

Task

Create an HP VEE model to:

1. Generate 100 random numbers along with their index as follows:

```
1:  xxx
2:  yyy
3:  zzz
.. etc. ..
```

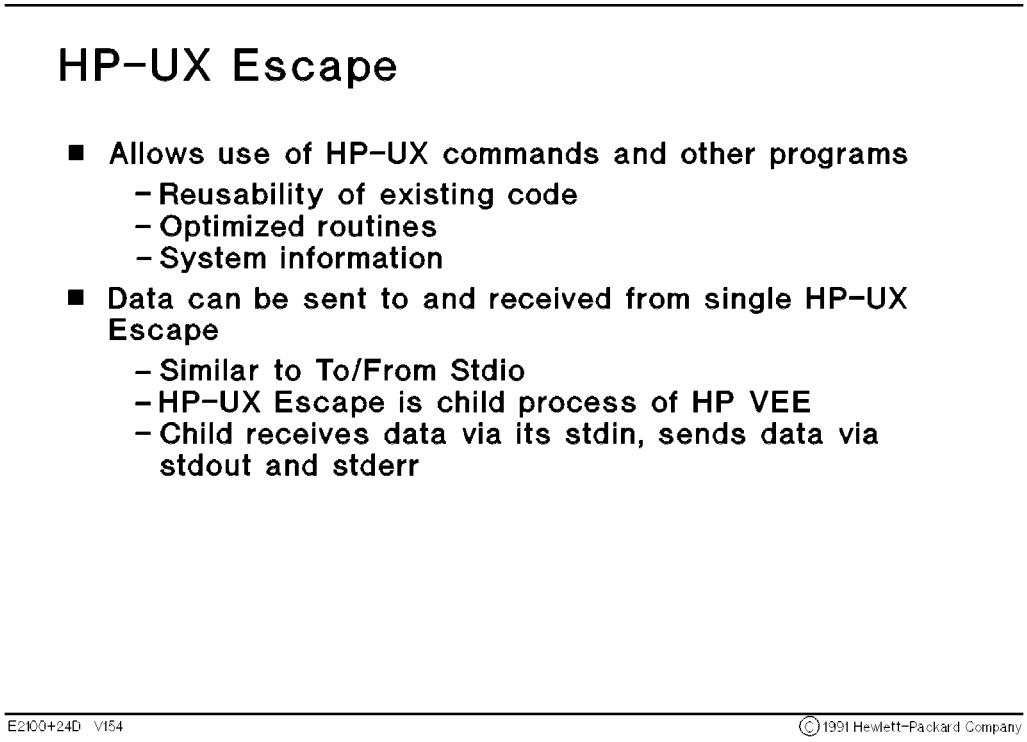
2. Write the time of day at the beginning of a data file.
3. Write these random numbers to the file.
4. Calculate the mean and standard deviation of the random numbers.
5. Append this data to the end of the data file as:

```
Mean:  mmm
Std. Dev:  sss
```

9

HP-UX Escapes

HP-UX Escapes



`HP-UX Escapes` give you the means to use HP-UX commands, shells and other programs with HP VEE. With this object you can:

- Reuse existing code. Call programs written in `C`, Pascal or other programming languages.
- Call libraries of optimized programs, and routines
- Obtain system information from HP-UX.

You can send and receive data from a single `HP-UX Escape` object. This object uses the transaction box syntax that we saw in other other I/O objects previously. `HP-UX Escape` is similar to the `To/From Stdio` objects. `HP-UX Escape` calls the program as a child process of HP VEE. The program receives data and sends data through its standard I/O channels, `stdin`, `stdout` and `stderr`.

Using Shell

Process can be invoked directly, or a shell can act as intermediary

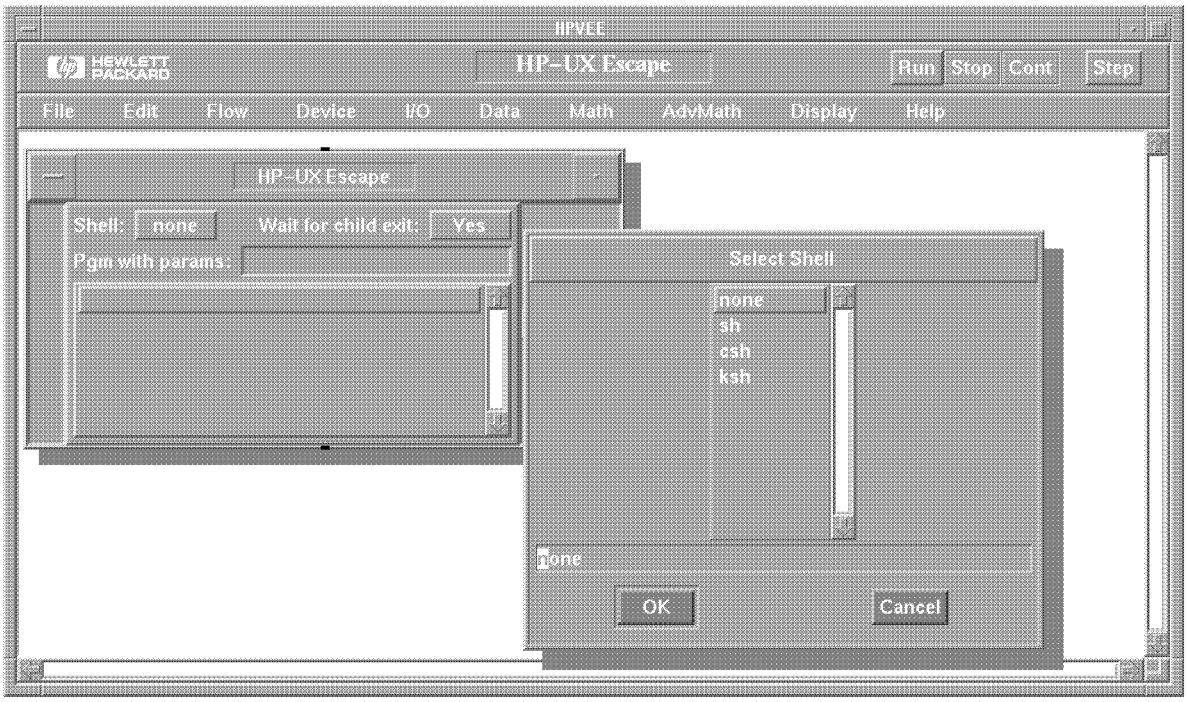
- **Advantages:** Interpret metacharacters ("*", etc.)
  Set up pipes, stdio redirection
- **Disadvantages:** Increased overhead (number of processes)
  Increased startup time (due to reading .kshrc, .profile, .cshrc)

E2100+24D V155

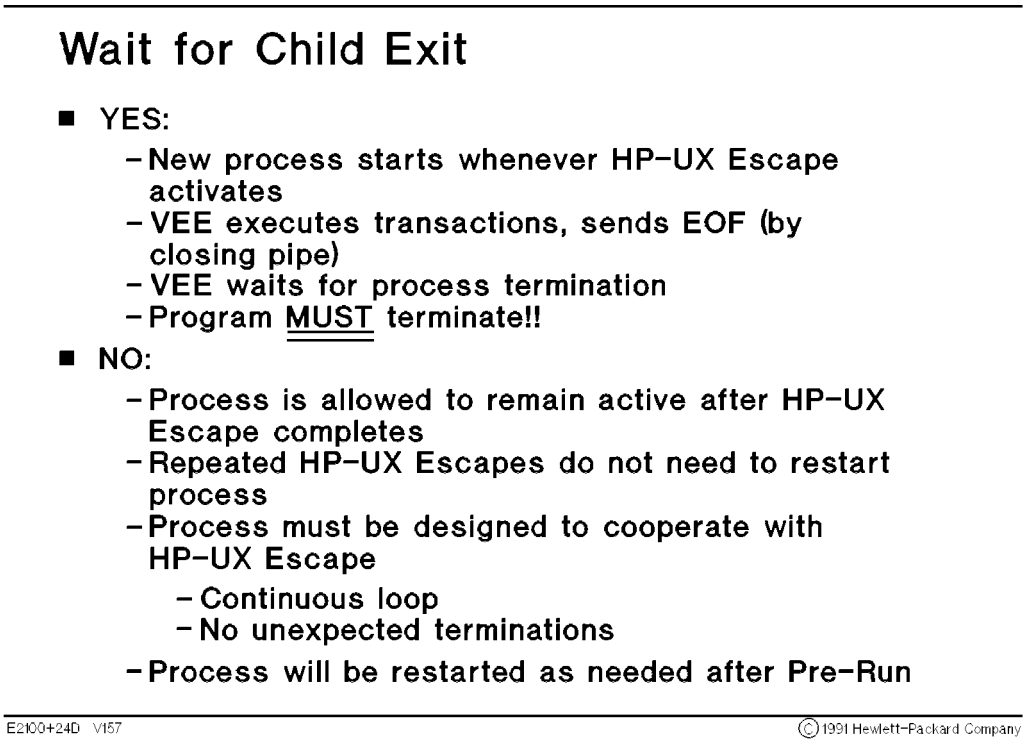
You can call the child process through a shell or directly from HP VEE. By calling through a shell you have the advantage of being able to interpret metacharacters such as `*`. You can also set up pipes or redirection.

However, by calling through a shell you also have the added overhead of another process execution time. It also increases the time need to start the process, since you read the `.*rc` and `.profile` files to start an additional process for the shell.

Screenshot - HP-UX Escape Select Shell



This screen shot shows how you can select one of three shells to use with the `HP-UX Escape` object.



When do you `Wait for child exit`? And what does that really mean? When do you want to exit?

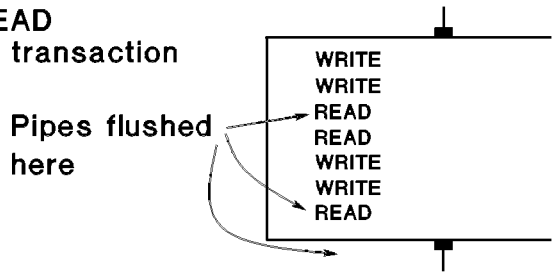
- `Wait for child exit: YES`
  - HP VEE starts a new process each time it activates this `HP-UX Escape` object.
  - HP VEE executes all of the transactions contained in the `HP-UX Escape` object.
  - HP VEE then *waits* for the new process to terminate. The process *must* terminate for the `HP-UX Escape` object to complete operation.
- `Wait for child exit: NO`
  - The process remains active after the `HP-UX Escape` object terminates.
  - Repeated activations of an `HP-UX Escape` do not need to restart a process.
  - The called process must be designed to cooperate with the `HP-UX Escape` object. It should be a continuous loop with no unexpected terminations.
  - HP VEE restarts the called process as needed after a `PreRun`.

Or more simply stated:

- Use `Wait for child exit: YES` when communicating with a forked shell or shell script.
- Use `Wait for child exit: NO` when communicating with a compiled program.

Pitfalls in Escape I/O

- Buffering must be disabled or buffers flushed
  - READ transaction may hang
  - Shell scripts cannot easily comply
  - C programs do:
    setbuf (stdout, NULL);
    OR
    fflush (stdout)
- VEE will flush its WRITE buffers
  - Before READ
  - After last transaction



E2100+24D V158

Here are some things that you should remember when working with HP-UX Escapes.

You must either disable or flush the buffers, otherwise, READ actions could hang. This is difficult to do with shell scripts. In C programs use the code

```
setbuf(stdout, NULL)
```

or

```
fflush(stdout)
```

HP VEE flushes its WRITE buffers before each READ action and when it finishes its last transaction. Note that HP VEE uses buffered writes and reads. Therefore, everything must be done with complete lines of data, not single characters.

Lab 8—Using HP-UX Escape to Unpack Data

Background

Binary data from an HP 3852 High Speed DVM has been stored in a file (`/labs/3852data`) of indefinite length. (You always wondered what anyone did with gigabytes of continuous 100KHz data. Well, here it is!)

You have a `c` program (`/labs/unpack.c`) that accepts one word of data from `stdin` and sends one `Real` voltage to `stdout`.

Task

1. Compile the `unpack.c` program.
2. Create a model to read ALL of the data in the `/labs/3852data` file.
3. Convert the data to voltage using `unpack`.
4. Plot the data to a strip chart for inspection.

You may alter the `c` program to give the best performance.

Hints

Use `Read BINARY a INT16` to read data from the file.

Use `Write BINARY a INT16` to write data to `unpack`.

Change `num=1` to reduce the overhead in `unpack.c`.

Add an indefinite loop and change `Wait for EXIT` to further improve throughput.

To compile a `c` program, use the the following command,

```
cc <file_name.c> -o <file_name>
```

10

Configuring & Customizing HP VEE

Configure & Customize

System Requirements

- HP VEE–Test – HP–UX application that requires 12MB file space and uses 5MB RAM (HP VEE–Engine uses 8 MB file space)
- SYSTEM – HP–UX 7.0 or later X11.4 windows

Recommended

345, 360, 375, 380, 400 (DIO)...

16 MBytes RAM

300 MByte Hard Disk

6 Plane, 1024x768, 1280x1024

Graphics

Remember....

RMB/UX, Multi–Tasking, VUE, Etc. will affect requirements!

E2100+24D V159

This slide lists the "Recommended Configurations" for your computer system hardware. Remember that these are minimum recommendations. For better performance, a faster CPU and disk drive and more RAM always help.

Installation

- Software shipped on
  - 1/4" cartridge tape
  - CD-ROM
- installation is performed by /etc/update
  - HP VEE-Test filesets VEE_TMAIN
    VEE_TIDS
    VEE_THELP
  - HP VEE-Engine filesets VEE_EMAIN
    VEE_EHELP
- HP VEE-Test requires device files to be created for each interface
  - run /usr/lib/veetest/vee_config on each system (cnode)

E2100+24D V160

The filesets used by `update` are listed here just for your information. The installation process takes care of this for you.

File System

- /usr/lib/veetest/

veetest	the actual executable (or veeengine)
hpidc	id compiler program
vee_config	program to set your /dev files
pcltrans	program needed for printing
xwd2sb	program needed for printing
./instruments	directory with .cid files *HP VEE-Test ONLY
./help	directory with help files
./examples	example HP VEE models
./config	directory of default config files
./lib	library of useful objects

- /usr/bin/veetest — symbolic link to run the /usr/lib/veetest/veetest executable

E2100+24D V161

These are the actual files used by HP VEE. Those of most interest to you are the `examples` directory which contains many example HP VEE models. Browse through this directory and look at some of this models at your leisure.

Another very useful directory is the `./lib` directory, which contains many useful `UserObjects` or other small models which you can merge into your models.

HP VEE Files that Assist With Customization

- .veeio
  - Config I/O file
  - Should be stored in default user directory
  - HP VEE includes a default version (d.veeio) at /usr/lib/vee/config/
- .Xdefaults

  File to customize colors, fonts, paletes, etc.
  - HP VEE includes several default versions at /usr/lib/vee/config/
- .veerc
  - Contains preferences for trig mode, auto line routing, printer configuration,waveform setting, number format

E2100+24D V162

HP VEE uses several files to remember your customization features.

`.veeio` contains information on your instrument configuration.

`.Xdefaults` contains information for the Xwindows system.

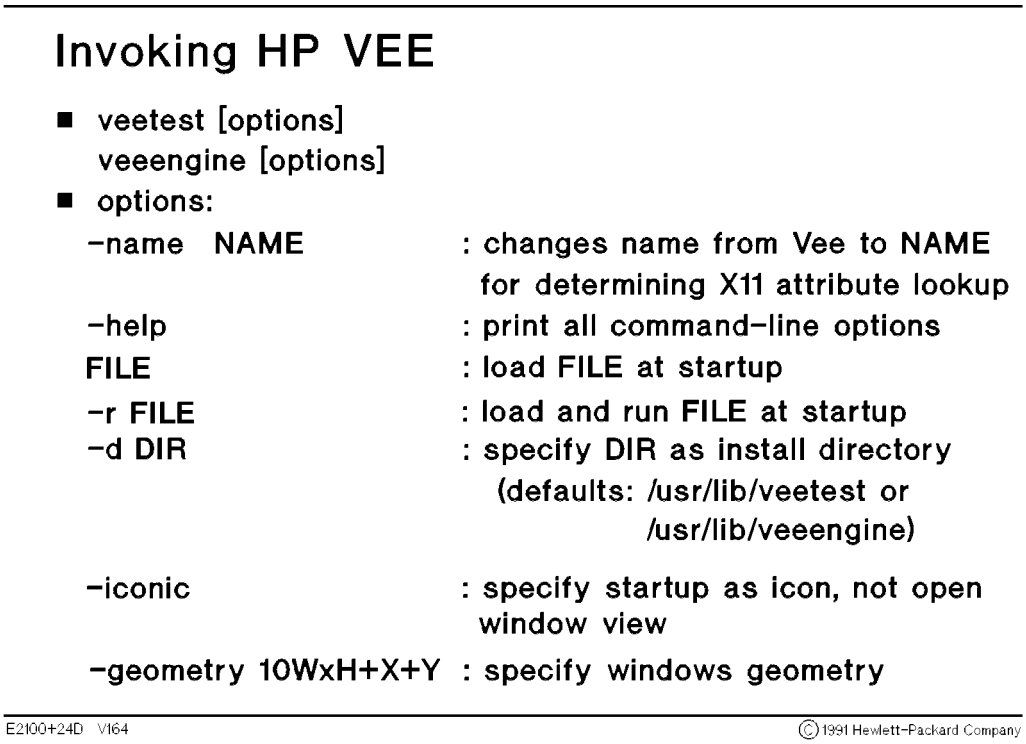
`.veerc` contains your preferences for HP VEE setup.

HP-UX Configuration

- HP VEE imposes no special requirements on kernel configuration, except:
  - X11 must be installed
  - hpib, gpio, and serial drivers are needed for HP VEE-Test I/O
- each HP VEE process requires >6MB swap space available

E2100+24D V163

Remember, that when you're setting up your system, you must install Xwindows and the drivers for all of the interfaces that you might access. You also need to allocate sufficient swap space on your disk drive. *Each* HP VEE process requires at least 6 Mbytes of swap space.



HP VEE permits several command line options when calling it. This slide lists the command line options. Note that typing `veetest -help` will print these command-line options on your terminal.

Remember that you can invoke HP VEE across a network, using a PC or an Xterminal as a display device using the `-display displayname` option.

Other Customizing Files

$HOME/.Xdefaults

/usr/lib/X11/app-defaults/Vee

- contain X customization strings for colors and fonts

Examples:

```
Mwm*Vee*clientDecoration : none
Vee*geometry :    1024X768+0+0
```

- creates full-size windows on 1024 X 768 display

E2100+24D V165

You can also use the X11 defaults file to help you customize your HP VEE work environment color and fonts. This slide lists the command to control the window manager decoration (window border) and geometry.

Configuration and Customization

- Printers
  - Current screen or entire workspace
  - Spool destination, resolution, dot expansion, color, orientation
- Global Preferences
  - Trig mode
  - Auto line routing
  - Number formats
  - Waveform settings

E2100+24D V166

Finally, within the HP VEE `File --> Preferences` menu selection, you can select several preference and configuration items, such as the `Trig Mode` angle preference (degrees, radians, or gradians) and number display formats.

You also specify the printer configuration, spooling, color, orientation, etc.

11

Instrument Control Interfaces

Instrument Control Interfaces

Computers now use many different interfaces to connect to various peripherals and instruments. We'll discuss the three most common for instrument control, HP-IB (IEEE 488), RS-232 and GPIO.

An Introduction To IEEE 488.2

- Review IEEE 488 Standard
- Review IEEE 488.2 Standard

E2110+24D V167

In this first section we do a quick overview of the IEEE 488 standard. We will also discuss a newer standard accepted by the IEEE in June, 1987, IEEE Standard 488.2.

Objectives of 488

- Define a general purpose, limited distance system
- Specify device-independent mechanical, electrical and functional interface
- Specify terminology and definitions
- Enable interconnections of different manufacturers equipment
- Permit direct communication without routing messages through a control unit
- Define system with minimum restrictions on performance characteristics
- Define asynchronous communications system with wide range of data rates
- Define a low cost system

E2110+24D V168

First, lets look at the objectives of IEEE 488.

It defines a general purpose, limited distance, digital interface. It defines the "device-independent" mechanical, electrical and functional aspects. It specifies and defines terminology used in describing the interface.

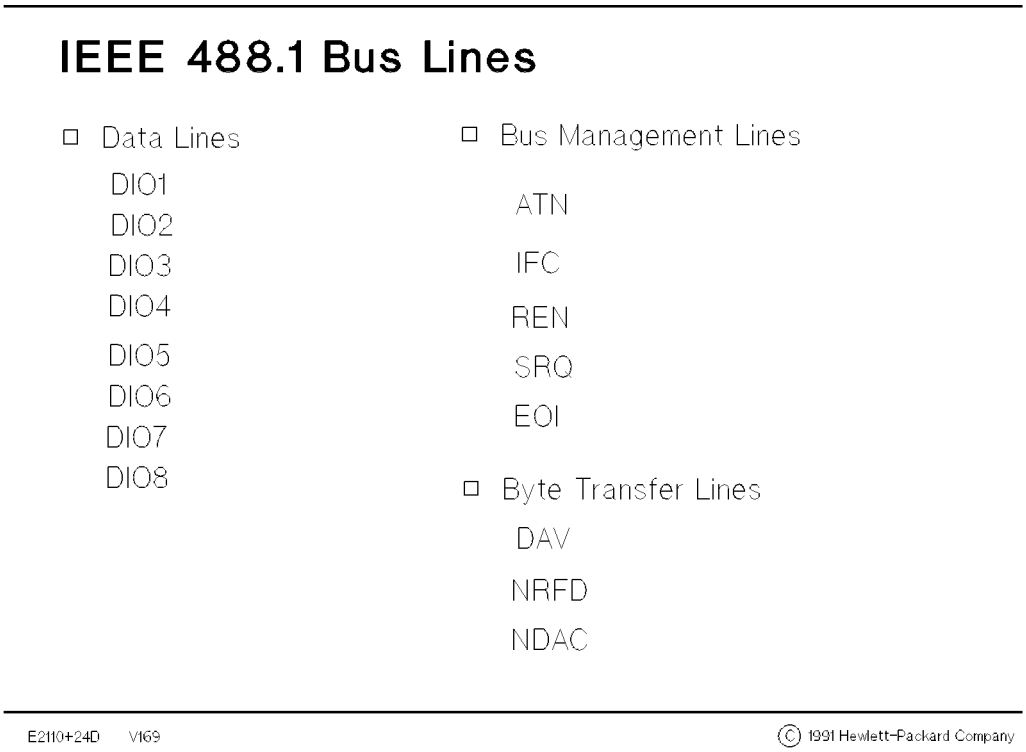
It permits different manufacturers to interconnect equipment.

The devices can communicate without sending the messages through a control device.

There are minimal restrictions or definitions on device performance characteristics. In fact, the performance restrictions only apply to the interface.

The interface uses an asynchronous handshake system with a very wide range of data transfer rates. It will work with slow and fast devices.

Finally it provides a relatively low cost interfacing system.



Here are the bus lines defined by IEEE 488.1:

- Data Lines
  - DIO1
  - DIO2
  - DIO3
  - DIO4
  - DIO5
  - DIO6
  - DIO7
  - DIO8
- Byte Transfer Lines
  - DAV—Data Valid
  - NRFD—Not Ready For Data
  - NDAC—Not Data Accepted
- General Bus Management Lines
  - ATN—Attention
  - IFC—Interface Clear
  - REN—Remote Enable
  - SRQ—Service Request
  - EOI—End Or Identify

IEEE 488 Key Specifications

- 15 Devices max
- Star or linear interconnection
- 16 Signal lines
- Byte serial, bit parallel messages
- 2 Metres per device, 20 Metres Total
- 1 MByte/sec maximum data rate
- 31 Primary addresses (992 secondary addresses)
- Multiple controllers (pass control)
- Hardware interface circuits (TTL, Schottky, Tristate)

E2110+24D V170

Now lets look at the Key Specifications of IEEE 488.1.

There can be a maximum of 15 devices connected together by IEEE 488.1 That includes the controller. They can be connected in a star or linear fashion, or a combination thereof. It uses a total of sixteen signal lines, eight data lines, three handshake lines and five bus management lines.

Data is transfered in a "Byte Serial, Bit Parallel" fashion, namely, entire bytes of data are transfered at a time.

When connecting a system together, the maximum length of cable allowed is 2 meters per device, with a total of 20 meters maximum in the total system.

The system is designed for a maximum 1 MegaByte per second data transfer rate.

Each device within the system is assigned a unique address. There are thirty one primary addresses and nine hundred ninety-two secondary addresses available.

There is also a provision to pass control amoung the various devices in the system.

Finally, it defines two types of electronics for the interface, open collector TTL or Schottky Tristate.

IEEE 488 Bus Device Functions

- Listener (receives data)
- Talker (sends data)
- Controller (assigned talkers and listeners)
- System controller (clear bus, put devices in remote mode)

E2110+24D V171

IEEE 488 also defines four basic bus device functions.

A LISTENER device receives data off of the bus, by performing the handshake operation.

A TALKER device places data on the bus and initiates the handshake operation.

The CONTROLLER (the controller in charge at this time) assigns who is a talker and who is a listener. Note that there may be more than one listener at a time.

The SYSTEM CONTROLLER can clear the bus, and put devices in Remote Mode. There can only be ONE System Controller in a measurement system, but there can be many controllers. However, there can only be one controller in charge at a time.

IEEE 488 Device Capability Subsets

SH	Source Handshake
AH	Acceptor Handshake
T	Talker
L	Listener
SR	Service Request
RL	Remote Local
PP	Parallel Poll
DC	Device Clear
DT	Device Trigger
C	Controller
E	Driver Electronics

E2110+24D V172

IEEE 488 also defines the device capability subsets. The devices may use all, part, or nothing out of the total set of capabilities. Each subset is carefully defined.

The source and acceptor handshakes allow the device to send and receive bytes on the bus. However, this does not mean that talker and listener capabilities exist.

Talker capability subsets include the ability to source data, respond to a serial poll, talk only and unaddress on "My Listen Address".

Listen capability subsets are similar, ability to accept data, listen only and unaddress on "My Talk Address".

Service Request is all or nothing. Either you can pull the SRQ line or you can't.

Remote/Local again is an all or nothing subset. In addition, there is an option to not include Local Lock Out. As you recall, Local Lock Out prevents the user from re-enabling the device front panel with a return-to-local button on the device.

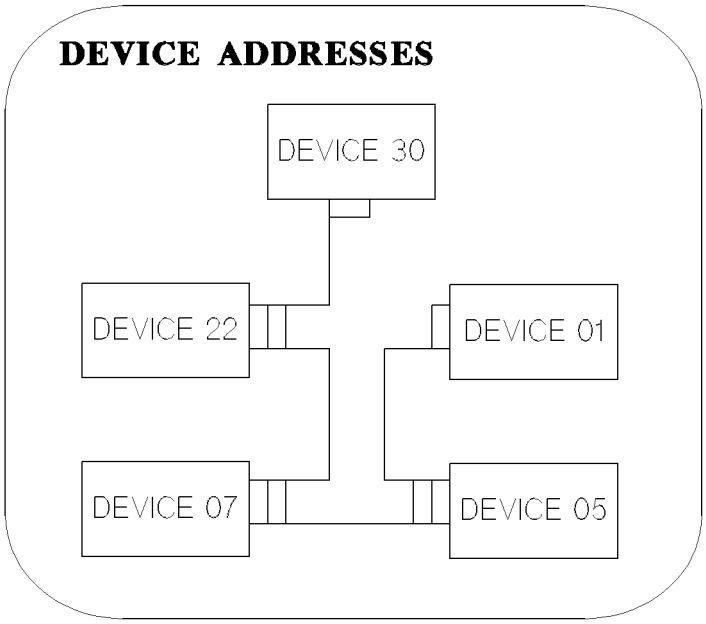
With Parallel Poll, you can have no capability, local configuration or remote configuration. The difference being, whether it can be programmed by the controller.

Device Clear is again, all or nothing with the option of omitting Selected Device Clear.

Device Trigger, all or nothing.

Controller Subsets are very complex. IEEE 488.1 defines 29 of them. Essentially, it covers the ability to be system controller, clear the interface and take charge, send remote enable, respond to service requests, pass control and those sorts of things.

Finally, Open Collector or Tristate bus driver electronics.



E2110+24D V173

Again, IEEE 488 devices each have a unique bus address. Devices can be connected in a linear fashion as shown here or in a star configuration or some combination.

That's ALL that IEEE 488.1 defines. Anything else that you understand to be IEEE 488 is something that an individual manufacturer has added to their implementation of 488.

IEEE 488

IT DOES NOT DEFINE:

- Status reporting
- Data format
- Data coding
- Minimum capabilities
- Message protocol

E2110+24D V174

IEEE 488 does NOT define:

Status Reporting. It does define a Status Byte and bit 6 within that byte. But, its up to each device designer to decide what data is reported in that byte, and what each bit means.

Data Coding and Formats. IEEE 488 simply states that any commonly recognized binary or alphanumeric code may be used. The ASCII charts that you may have seen that include the commands and addresses for IEEE 488 are only used for the convenience of the user. They are not part of the Standard. Therefore, you can represent data in anyway that you like.

Minimum Required Capability. Devices that conform to IEEE 488 can be anything from a Listen Only printer to a full blown computer acting as system controller. Infact, there could be devices that simply implement the handshakes.

Message Protocol. IEEE 488 only defines how to pass one byte from one device to another. What do you do if you are interrupted in the middle of a data transfer? What if a device doesn't terminate its data the way another device wants it to terminate? What does a device do when its buffers are full and the controller is trying to send it another device command? IEEE 488 does not define anything of this nature. Each device designer gets to make these decisions. As you know by experience, this doesn't always work.

IEEE 728 Recommended Practice for Code and Format Conventions

DEFINED:

- Data messages
  measurement, program, status, display
- Separators
- Headers

- Loosely defined with too many choices
- Wide open to "personal interpretation"

E2110+24D V175

In an effort to standardize the data formats and codes the IEEE defined the IEEE 728 Standard. Notice, that this is a RECOMMENDED standard, not a required standard. It defined data messages of various types, data headers and separators. However, it was loosely defined and left the designer with way to many choices. It was wide open to personal interpretation.

Device Dependent Commands
What Does DCL
(Device Clear) Do?

- "Return to a pre-defined, device-dependent state"
- Some clear I/O buffers
- Some change the function state of the device
- Some do a complete device reset

E2110+24D V176

Another example of the difficulties encountered when using IEEE 488 is Device Clear. What should a device do when it receives the DCL command? According to the standard it "returns to a pre-defined, device-dependent state." Well, that leaves the field wide open. The device designer can decide to do anything that appears to be good for him. Some devices clear their I/O buffers. Other devices change the function state of the device, setting it to do a totally different function. Others do a complete self-test and reset.

Problems Encountered Using IEEE 488

- Minimum capability set
- Data format
- Data codes
- Message protocol
- Commands
- Status

E2110+24D V177

In summary, here are some of the problems encountered when you use IEEE 488.

There is no required minimum set of capabilities.

IEEE 488 does not define data formats or codes. IEEE 728 gives you too many choices.

Other than transfering one byte, it does not define a message protocol.

IEEE 488 does not define a set of common commands and some of the bus commands defined by it leave too much room for interpretation.

IEEE 488 does not define a status reporting structure other than implementing the SRQ bit in the Status Byte.

IEEE Solution

- IEEE 488 renamed 488.1
- IEEE 728 obsoleted
- IEEE 488.2 defined

E2110+24D V178

To resolve these problems, the IEEE formed Committee 981. This commitee, made up of representatives from both instrument manufacturers and purchasers, drew up a new standard to be used in conjunction with IEEE 488. Since it was very closely tied to 488, rather than giving it a new number they gave it a suffix, IEEE 488.2 and renamed IEEE 488 to 488.1. In addition, they obsoleted IEEE 728.

IEEE 488.2

- Required IEEE 488.1 Capabilities
  (Everyone can talk, listen and be serial polled)
- Data Formats
  (e.g. Numbers all look the same)
- Message Protocol
  (Bus hangups are minimized)
- Common Commands
  (e.g. *IDN? Identifies the instrument)
- Status Model
  (Status byte usuage is consistent)

E2110+24D V179

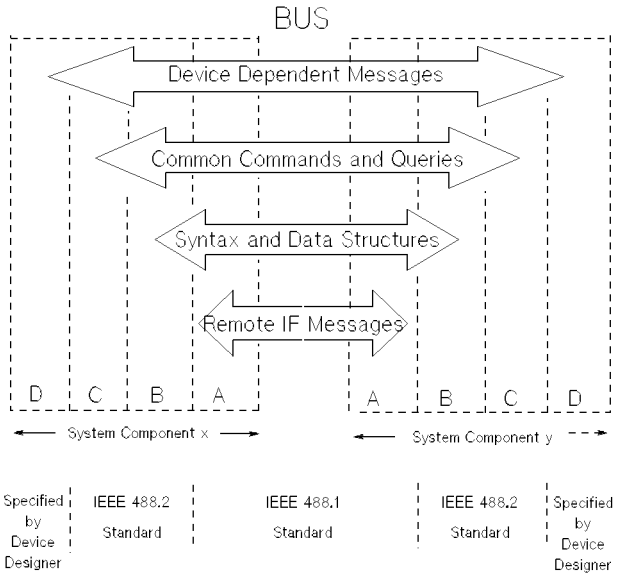
So what does IEEE 488.2 define? In a nutshell, it defines a required minimum set of interface function capabilities. These required capabilities define that every device will at least be able to talk, listen, and be serial polled. They may have capabilities in addition to these, but they will all have this minimum set.

IEEE 488.2 also defines a set of Data Codes and Formats. This means that all numbers passed back and forth between devices will look the same. We'll take a more detailed look at these formats in just a few moments.

This new standard also defines a Message Protocol. This protocol minimizes bus hangups by defining what to do in the "exception cases", where something interrupts the normal flow of bytes between devices.

It also defines a set of Common Commands. These commands execute functions common to every bus device, such as Reset or Identify.

IEEE 488.2 also defines a common Status Model. This provides for consistent usage of the Status Byte. We'll discuss this at more depth later in this presentation.



E2110+24D V180

We can look at the relationship of IEEE 488.1 and 488.2 graphically as well.

The bottom layer is IEEE 488.1. It provides the basic communication of bytes between devices. This is the mechanical, and electrical interface between devices.

The next two layers are defined by IEEE 488.2. Now that we can pass bytes between devices we need to define what those bytes mean. The syntax and data structures included in IEEE 488.2 provide the framework for the data.

The next layer are the common commands and queries defined by 488.2. Again, these are commands common to every device.

The top layer contains the commands necessary to control the individual device. These commands are defined by the device itself and are unique to the device. IEEE 488.2 does not address these commands.

From this slide you can see the layered structure of the device interface and how these two standards work together.

DEVICE COMPLIANCE CRITERIA

- IEEE 488.1 requirements
- Message exchange requirements
- Syntax requirements
- Status reporting requirements
- Common commands
- Synchronization requirements
- System configuration requirements
- Controller requirements
- Documentation requirements

E2110+24D V181

Let's begin examining some of the details of IEEE 488.2.

First, what does it take to put an IEEE 488.2 seal of approval on a device?

It must have the minimum set of IEEE 488.1 capabilities, meet message exchange, syntax and status reporting requirements. It must implement a set of required common commands. It must also meet requirements on system synchronization and configuration.

If the device implements controller capabilities, it must meet certain requirements in this area. Now, this does not mean that all devices must be controllers, just that if it is a controller, it must implement a minimum set of controller capabilities.

Finally, the device documentation must indicate all of the capabilities that the device implements.

IEEE 488.2 DATA CODES

- ASCII-Bit Code
- 8-Bit Binary Integer
- Binary Floating Point

E2110+24D V182

Lets look at the data codes and formats defined by IEEE 488.2

IEEE 488.2 standardizes on the ASCII 7-Bit Code. This is the standard code for passing text and data between devices. We'll look at the format for this data in just a moment.

In some cases, its more efficient to move data in a binary code rather than converting to ASCII. For those cases, IEEE 488.2 defines an 8-bit code and a Binary Floating-Point Code. The 8-bit code is ordinary 8 bit coding, with the lowest bit corresponding to the lowest numbered data line. For floating point numbers, IEEE 488.2 recommends using the IEEE 754-1985 Floating Point Format.

SYNTAX REQUIREMENTS

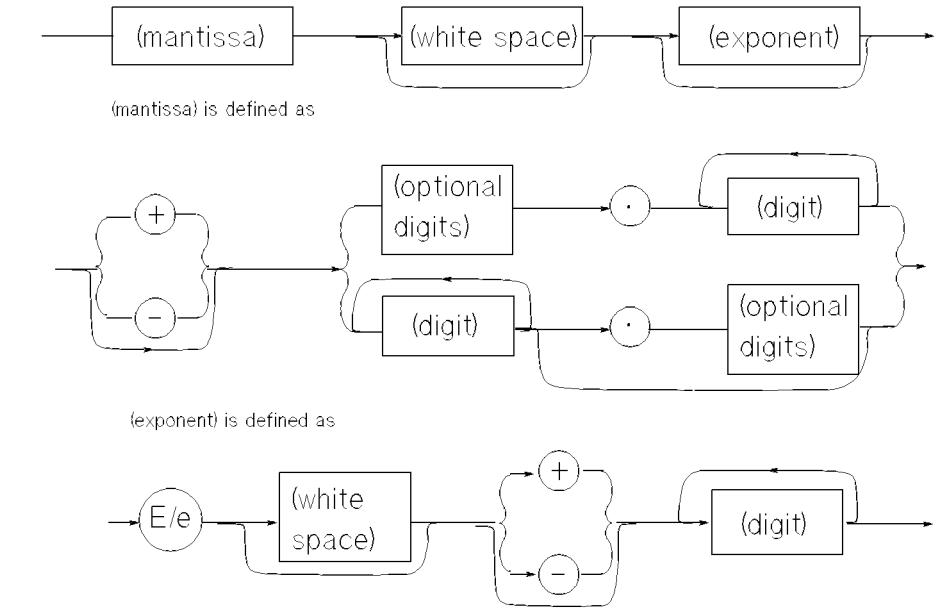
"Forgiving Listening, Precise Talking"

Device Listening Functional Elements

Device Talking Functional Elements

In defining data formats, IEEE 488.2 uses a concept that may be new to some of you as far as relating to device data exchange. This concept is "Forgiving Listening, Precise Talking". This concept embodies the idea of accepting various data formats when listening, or receiving data, but being very precise when talking, or sending data. This allows older devices to still be able to communicate with the newer devices.

Here are some examples of Forgiving Listening and Precise Talking.



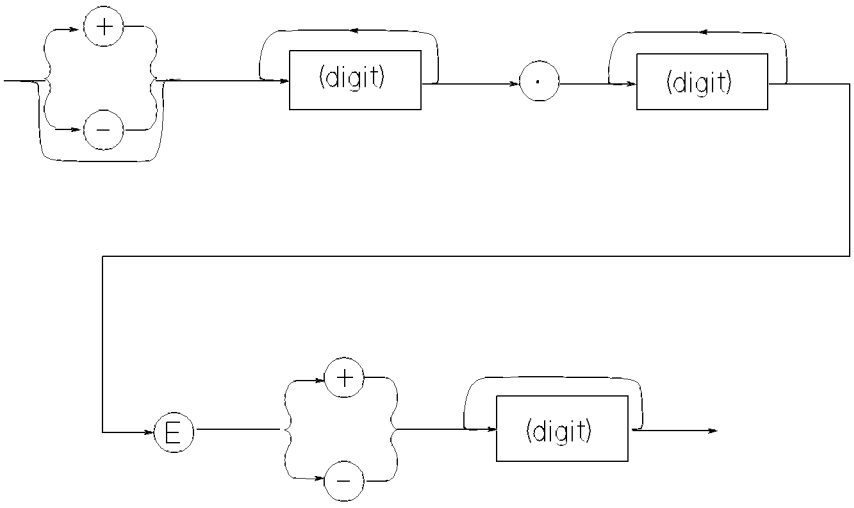
E2110+24D V184

This is an example of Forgiving Listening. This is the definition of the Listener Format for a Decimal Number. The two basic elements are the mantissa and the exponent, separated by white space.

The mantissa may or may not have a leading sign. Note that there may or may not be digits preceeding and trailing the decimal point. This leaves a great deal of flexibility in accepting data.

The exponent is very similar. The leading "E" may be in either upper or lower case. White space and the sign are optional, followed by the exponent digits. Again, a great deal of flexibility.

Finally, white space is defined as any ascii character, 0 through 32 (decimal) except line feed (newline). That includes all of the control characters. Again, its very flexible.



E2110+24D V185

Contrast the Forgiving Listening with this example of Precise Talking.

This is the definition of the Floating Point response data for IEEE 488.2. The initial sign is optional. After that the choices are very carefully defined. There must be at least one digit preceeding and trailing the decimal point. The "E" separating the mantissa from the exponent must be upper-case. No white space is permitted. There must be a sign for the exponent and at least one digit in the exponent. Everything here is very carefully defined in contrast with the listening format where there was much more flexibility. By precisely defining the format for talkers, it will be easier to parse and interpret the data transfered across the bus.

STATUS REPORTING MODEL

- IEEE 488 Defined:
  - Status byte and service request bit

E2110+24D V186

Lets turn now to the Status Reporting Model. You may recall that IEEE 488 defined a status byte, and bit 6, the Service Request Bit within that byte. The rest of the status byte, and how it operates is left up to the device designer.

STATUS REPORTING MODEL

- IEEE 488.2 Defines:
  - Master summary status bit
  - Event status bit
  - Message available bit
- Standard Event Status Register

Power on	Device dependent error
User request	Query error
Command error	Request control
Execution error	Operation complete

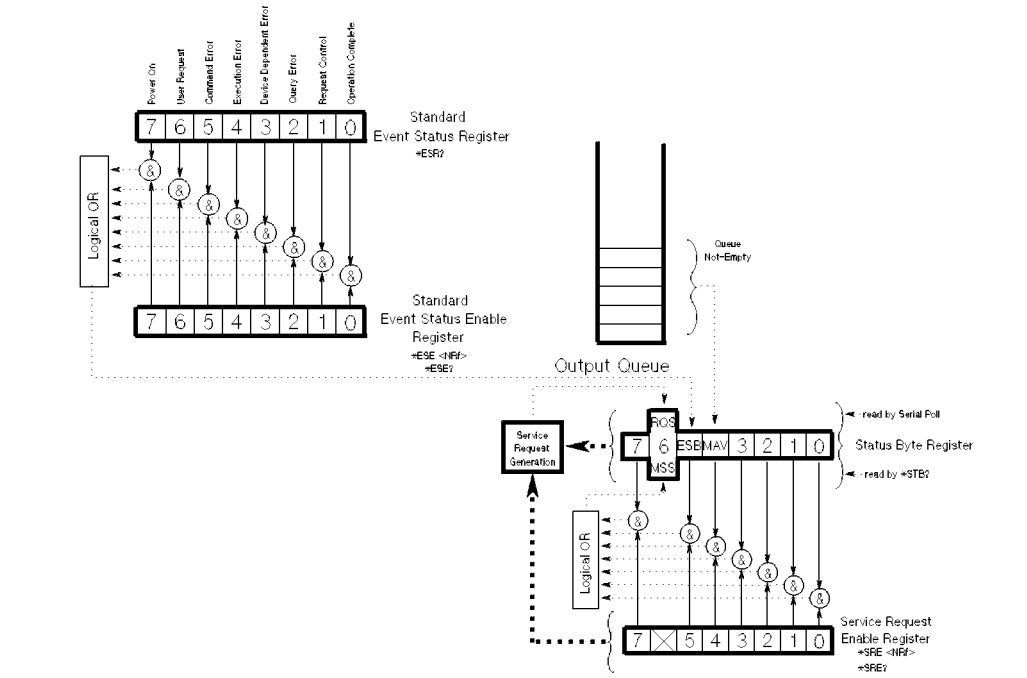
Standard Event Status Enable Register
Output Queue

E2110+24D V187

IEEE 488.2 defines the meaning of two additional bits within the Status Byte and gives additional duty to bit 6. Bit 6, when read with a new command defined by IEEE 488.2, becomes the Master Summary Status Bit. It lets you know that something of interest has occured. The other two defined bits are the Event Status Bit and the Message Available Bit.

The Event Status Bit summarizes the Standard Event Status Register. This newly defined register contains information, standard to every device, such as command errors, or request for control of the bus. You can see the bits listed here on the slide.

The Message Available Bit indicates that data is available in the Output Queue, another feature defined by 488.2.



E2110+24D V188

Looking at the graphical representation of the new status models allows you to see how these different bits and registers interact. Notice that both the Standard Event Status Register and the Status Byte have a corresponding "Enable" register to determine which bits are reported in the Request Service Bit or Master Summary Status Bit.

The Output Queue status is reported in the MAV bit of the Status Byte. IEEE 488.2 only requires a depth of 1 byte in the queue, but it must be implemented as a queue.

As you can see, the status model leaves 5 bits undefined in the Status Byte. Devices are free to use these bits to report information. However, they must follow the same model of either an event status register or a queue, like those used for the Standard Event Status Register and Output Queue. The register may only be one bit wide, but it must have an enable register and a summary bit. This means that the Status Byte reports only a summary of other status structures.

IEEE 488.2
COMMON COMMANDS

Defined using ASCII characters

Preceeded by an "*"

Queries end with "?"

E2110+24D V189

IEEE 488.2 defines a set of common commands that each device will implement. These commands are defined as ASCII characters. All of the commands have an asterisk as the first character. Queries all end with a question mark. The thing to note about these commands are that they are passed across the bus in the data mode. These are NOT new bus commands, but rather a set of common device commands.

IEEE 488.2 REQUIRED COMMANDS

(ATN FALSE)

SYSTEM DATA

*IDN	Identification

INTERNAL OPERATIONS

*RST	Reset
*TST?	Self-Test Query

SYNCHRONIZATION

*OPC	Operation complete
*OPC?	Operations complete query
*WAI	Wait to continue

E2110+24D V190

This slide and the next slide list the Required Common Commands. Every IEEE 488.2 device must recognized and execute these commands. There are commands for identification, reset and self-test, sychronization and status. The 488.2 standard defines what the devices will do and won't do for each one of these commands.

Some commands that you might not be familiar with are the Synchronization and the Status and Event commands.

The synchronization commands provide a method to synchronize the operation of several different devices. The Operation Complete Command and Query instruct the device to indicate when it completes an operation.

The Wait command causes a device to complete all pending instructions before it begins executing any newly received instructions. This is useful for more powerful devices that can pipeline commands.

IEEE 488.2 REQUIRED COMMANDS

(ATN FALSE)

STATUS & EVENT

*CLS	Clear status
*ESE	Event status enable
*ESE?	Event status enable query
*ESR?	Event status register query
*SRE	Service request enable
*SRE?	Service request enable query
*STB?	Read status byte query

E2110+24D V191

The status commands, listed on this next slide, allow the user to access the newly defined status structures that we discussed earlier.

IEEE 488.2

- Required IEEE 488.1 Capabilities
  (Everyone can talk, listen and be serial polled)
- Data Formats
  (e.g. Numbers all look the same)
- Message Protocol
  (Bus hangups are minimized)
- Common Commands
  (e.g. *IDN? Identifies the instrument)
- Status Model
  (Status byte usuage is consistent)

E2110+24D V192

So to summarize, IEEE 488.2 defines a required minimum set of 488.1 interface capabilities. In other words, every device can talk, listen and be serial polled. It defines data codes and formats, and a message protocol. It defines a set of common commands and defines the operation of those commands. And it defines a common status model.

For more information on IEEE 488.1 and 488.2 contact your local HP Sales Office and ask for the "Tutorial Description of the Hewlett-Packard Interface Bus", part number 5021-1927.

EIA DEFINED:

- Mechanical Characteristics
- Electrical Characteristics
- Interchange Circuits & Functions
- Relationship to Standard Interface Types
- Similar to CCITT V.24 & V.28

E2110+234D V193

In 1963, the Electronic Industry Association (EIA) established a standard for the interface between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) that uses a serial binary data interchange. The latest revision of this standard, which has been in effect since 1969 and reaffirmed in 1981, is known as RS-232-C. The standard covers:

- Mechanical Characteristics of an interface.
- Electrical Characteristics of an interface.
- Interchange circuits with descriptions of their functions
- Relationship of interchange circuits to standard interface types

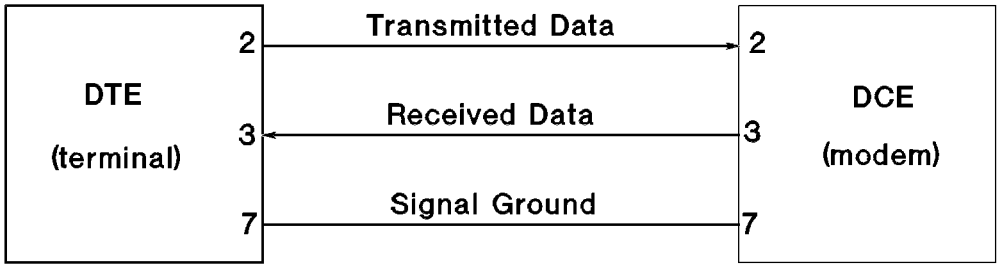
The Comite Consultatif International Telephonique et Telegraphique (CCITT) also established standards that correspond to RS-232-C. While these standards, CCITT V.24 and CCITT V.28, are very similar to RS-232-C, they are not identical.

The mechanical specification defines 22 pins and designates three pins as unassigned, but does not specify a connector. However, industry accepted one as a defacto standard. DTE's (Terminals, computers, etc.) use the male connector. DCE's (modems) use the female connector. Cables should be no longer than 15.24 meters (50 feet).

You can purchase a copy of the RS-232-C standard from:

Electronics Industries Association, Engineering Department, 2001 Eye Street, N.W., Washington, DC 20006

RS 232

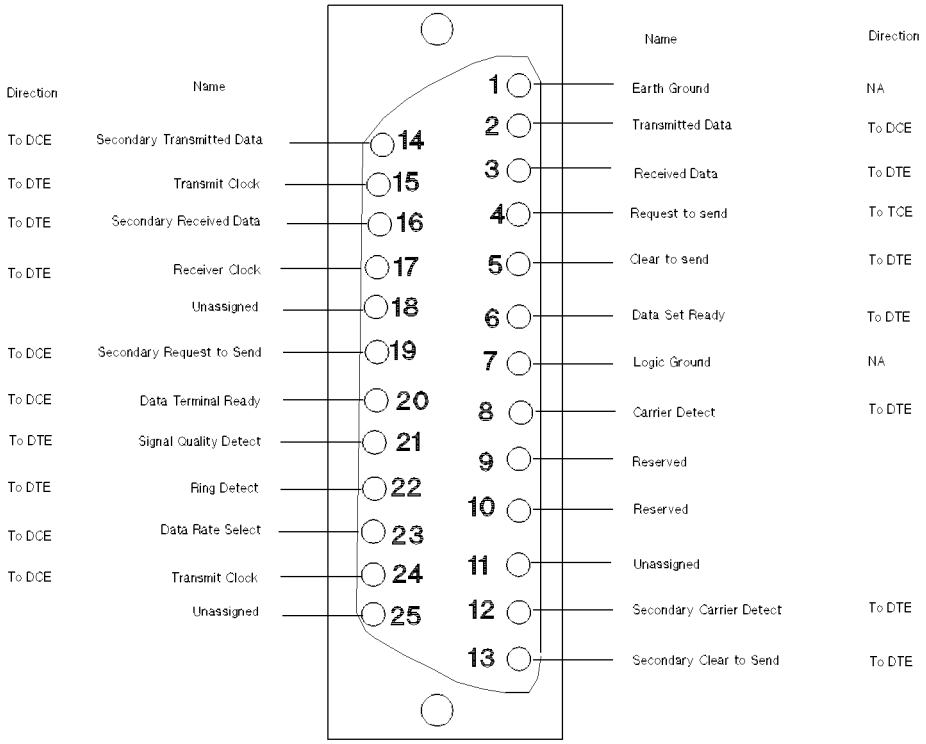


E2110+24D V194

Although RS-232 defines 25 lines, you really need to understand the operation of only three of them, Transmit Data, Receive Data and Signal Ground. The data moves along these lines. The others provide a means of handshaking between devices and detecting status.

The important thing to note with RS-232 is which direction the data is going. Note that for DTE devices, data flows OUT of pin 2 and into pin 3. Just the opposite occurs for a DCE device. Therefore, if you have difficulty moving data via RS-232, check pins 2 and 3 to see if the data is going the right direction for that device.

RS 232 Pin Assignments



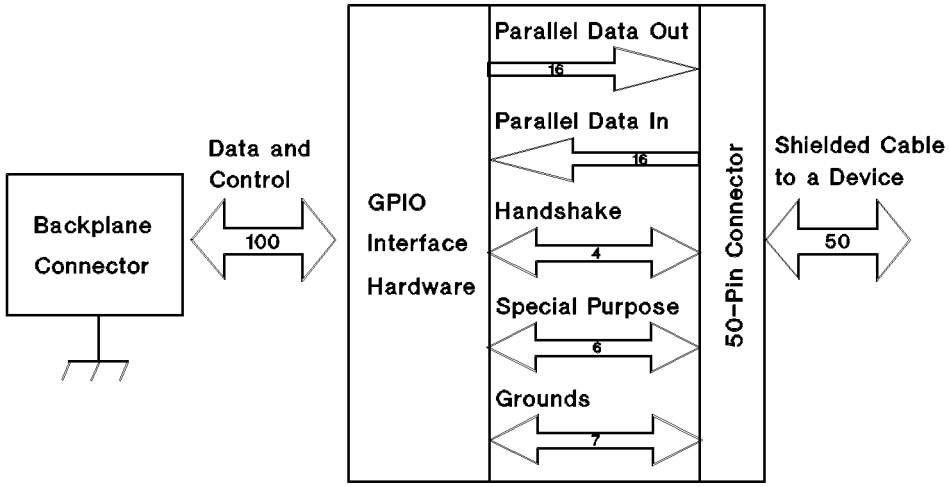
E2110+24D V195

For those of you that are curious, this slide shows the definitions and directions for all of the RS-232 pins.

RS-232-C Connector Pin Assignments

Pin	Name	Direction
1	Earth Ground	N.A.
2	Transmitted Data	To DCE
3	Received Data	To DTE
4	Request to Send	To DCE
5	Clear to Send	To DTE
6	Data Set Ready	To DTE
7	Logic Ground	N.A.
8	Carrier Detect	To DTE
9	Reserved	
10	Reserved	
11	Unassigned	
12	Secondary Carrier Detect	To DTE
13	Secondary Clear to Send	To DTE
14	Secondary Transmitted Data	To DCE
15	Transmit Clock	To DTE
16	Secondary Received Data	To DTE
17	Receiver Clock	To DTE
18	Unassigned	
19	Secondary Request to Send	To DCE
20	Data Terminal Ready	To DCE
21	Signal Quality Detect	To DTE
22	Ring Detect	To DTE
23	Data Rate Select	To DCE
24	Transmit Clock	To DCE
25	Unassigned	

GPIO Parallel Interface



Block Diagram of the GPIO Interface

E2110+24D V196

The GPIO interface provides the most flexibility of all interfaces for communicating with a variety of devices. This interface sends and receives up to sixteen bits of data with a choice of several handshake methods. It also provides external interrupt and definable signal lines for additional flexibility.

Just to keep things simple, remember that there are several ways to move data in and out of GPIO interfaces. When you want to use it READ THE BOOK that accompanies the interface.

12

Using Drivers for Instrument Control

Using Drivers for Instrument Control

HP VEE Device I/O

- Instrument "State" Drivers
  - Developed by HP for 170 instruments
  - Easiest HP VEE instrument control
  - Most interactive
- Instrument "Component" Drivers
  - Allow efficient access to instrument driver
- Direct I/O
  - For devices and instruments with no pre-developed drivers
  - Fast, flexible, and powerful
  - Transaction interface consistent with other HP VEE I/O objects

E2110+24D V197

HP VEE communicates with instruments in three ways.

It uses HP Instrument drivers; we call them state drivers. These are the same drivers developed for HP ITG. They provide easy interactive instrument control. They are called state drivers because they keep track of the instrument's state or function settings.

HP VEE also uses the instrument drivers in a fashion called Component drivers. These same drivers can also communicate more effeciently by not keeping track of the instrument state and not displaying the instrument information.

For those devices which do not have a driver, or for the very fastest I/O with instruments, HP VEE provides Direct I/O. This object is the fastest and most flexible, but requires the user to provide the actual instrument commands, where the two methods above do not require this. Direct I/O uses the same transaction interface that we've learned about previously.

HP Instrument Drivers

- **Used in ITG and HP VEE-Test**
- **Text file that defines:**
  1. **Instrument components (or functions)**
  2. **Bus mnemonics to set components**
  3. **User interface for front panel interaction**
- **Also contain function interrelation (coupling)**

HP instrument drivers provide access to most programmable functions available on the instrument

Coupling and the proper order of components allows incremental state programming

E2110+24D V198

HP VEE uses the same instrument drivers as HP ITG. So if you've written a driver for ITG, you can use the same driver with HP VEE. You can also use the driver writing tools provided for HP ITG to write new drivers.

These instrument drivers are text files that define:

- Instrument components or functions
- Instrument commands or mnemonics that control the instrument functions.
- User interface for front panel interaction.

The drivers also contain information on how various instrument functions interact. This interaction is called coupling.

HP instrument drivers provide access to most of the programmable instrument functions available in the device.

These drivers also permit you to use incremental state programming since it tracks the state of the instrument.

Instrument Drivers

- All drivers are compiled to allow fast loading
- All HP ITG drivers are supported except 3852 (user subs)
- HP VEE drivers default to Incremental On which only works for state programmable (supplied) drivers
  - Incremental Off may work for some "homemade" drivers

E2110+24D V199

The HP VEE product includes all of the HP ITG drivers in compiled form. Compiled drivers load faster. HP VEE supports all of the instrument drivers currently available, except the HP 3852 driver, since it uses HP BASIC user subs.

Note that all HP VEE instrument drivers default to `Incremental On` mode. This mode only works for state programmable drivers, like the drivers supplied with HP VEE. `Incremental Off` mode may work for some “homemade” drivers that you write.

State vs. Component Drivers

- "State" Drivers
  - Show complete graphical panel
  - Use when working with full instrument states
- "Comp" Drivers
  - Don't show full panel
  - Use for setting specific components
  - Efficient, yet still isolate instrument knowledge to driver
  - Only maintain state of specific components

E2110+24D V200

Now let's take a look at the difference between State drivers and Component drivers.

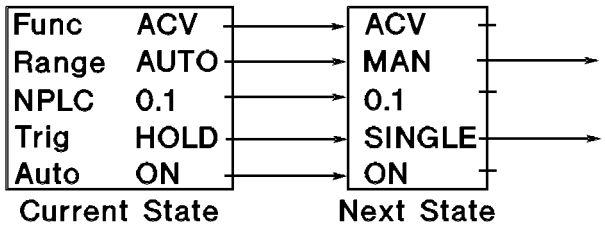
The State drivers are equivalent to the ITG Recall function. They show the complete graphical user interface panel. Use these objects when you want to work with full instrument states.

The Component drivers are equivalent to the ITG Set and Get functions. These objects do not display the user interface panel. Use them to set specific components and functions of an instrument. For example to set the function of a DVM to DC Volts or AC Volts.

The Component drivers are more effecient than the state drivers since they don't maintain the state of specific components. They also do not require the user to know the specific details of instrument commands.

Incremental State Programming

- The software package maintains a state table of current instrument settings
- Users can request a single component or entire instrument state to be sent
- With Incremental Mode ON, only required components are sent to instrument



E2110+24D V201

With incremental state programming, HP VEE maintains a state table of current settings for the instrument. You can then request that HP VEE sends an individual component or function change or an entirely new instrument state. When you have `Incremental Mode ON`, HP VEE sends only those components required to update the state of the instrument.

State Drivers

- Error Checking ON
  - For instruments that allow it, after an action list is sent, the driver requests the instrument error status
- Error Checking OFF
  - Speeds execution some, but leaves the user exposed to unreported errors

  There are better ways to optimize than turning off error checking

E2110+24D V202

State drivers also check for errors when sending commands to instruments by requesting the instrument's error status. To improve the speed of exectution, turn error checking off. This leaves you vulnerable to unreported errors.

We'll show you some better ways to improve performance later.

HP VEE Comp Drivers

- Only required functions are added to object
- Only added functions have state maintained
- Because State Lookup is NOT done for every function, comp drivers execute much faster than state drivers
- Assumes incremental OFF, error checking OFF

E2110+24D V203

With HP VEE component drivers, you only add the functions you need to the object. You simply add an input or output terminal, choosing from the list of available functions. This driver only tracks the state of the functions you add. Since `Comp` drivers do not look up the entire state of the instrument they execute faster than state drivers. These drivers assume that incremental state programming and error checking are disabled.

HP VEE Instrument Drivers Summary

- State drivers for users wanting full graphical panels
- Comp drivers to set/get specific components to optimize driver performance
- State drivers and comp drivers can be mixed and matched
- Multiple instances of same driver (state or comp) to same instrument share state

E2110+24D V204

So to summarize:

- State drivers provide a full graphical user interface panel.
- Component drivers set and get specific instrument components to optimize performance.
- You can use both state and component drivers in the same model.
- Multiple instances of the same driver (either state or component) to the same instrument share the same state table.

Using Instrument Drivers

- First step is creating HP-UX device files
  - Performed by vee-config
- Next step is to Configure I/O
  - Allows user to specify which instruments are available, the instrument address, timeout, etc.
  - All instrument configuration must be accessed under Configure I/O
  - Two menus under CONFIG
    1. Direct I/O Configuration
    2. Instrument Driver Configuration

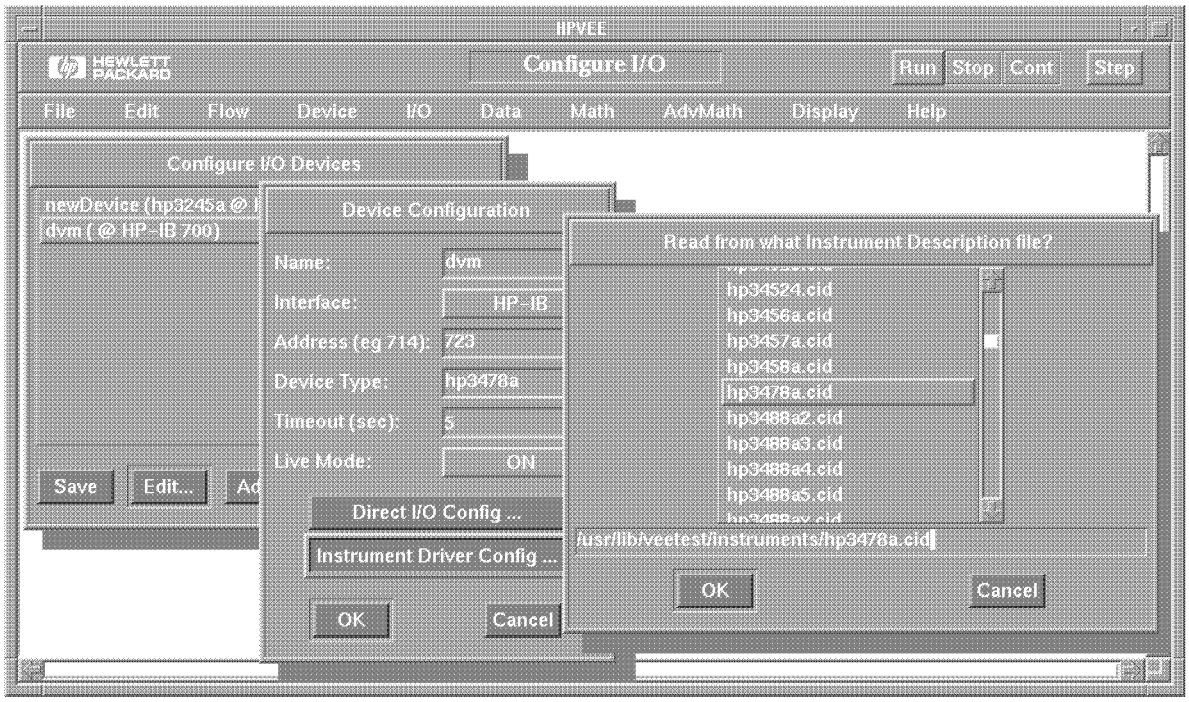
E2110+24D V205

So how do you really use these instrument drivers? First, you must do some work to set up your system.

When you install HP VEE, the installation process runs a utility, `vee_config`, which creates HP-UX device driver files for each interface that you will be using. However, `vee_config` does not verify that the actual drivers are configured in your HP-UX kernel. Be sure to have your system administrator verify that your kernel configuration contains the proper interface drivers. Also note that you *must* run vee_config on each `cnode` of a diskless system to configure the `/dev` files for that system.

Next use the `Configure I/O` function to specify the instruments that you will be using. With `Configure I/O` you specify the instrument interface, address, timeout and so forth. You must do this for each instrument. There are two menus under `Configure I/O` to configure instrument drivers and direct I/O.

Screenshot - Configure I/O Devices



Here is an illustration of configuring instruments with `Configure I/O`.

Using Instrument Drivers
Device Configuration

- **NAME** – Must be unique to this config
  – What will appear on the object title
- **INTERFACE** – Choose interface type – HP–IB, Serial, GPIO
- **ADDRESS** – 0 – if no instrument is present
  – 714 – Bus Address (7) and Instrument Address (14)
- **DEVICE TYPE** – Descriptive name
  – Defaults to Driver File Name
- **TIMEOUT** – Instrument Timeout in seconds
- **LIVE MODE** – ON – Instrument is present
  – OFF – No instrument attached
  (If address is 0, live mode is OFF)

E2110+24D V207

So what do you do to configure and instrument driver?

`NAME` a unique name for this instrument object. This name appears in the instrument title.

`INTERFACE` specify the interface type: HP-IB, Serial (RS-232), GPIO

`ADDRESS` 0 - for no instrument connected to system.

714—interface address (7) and instrument address (14)

`DEVICE TYPE` a descriptive name for the instrument. This defaults to the instrument driver file name.

`TIMEOUT` instrument timeout, in seconds.

`LIVE MODE` `ON`, instrument connect to system.

`OFF`, no instrument connected to system. If address is 0, `LIVE MODE` is `OFF` automatically.

Using Instrument Drivers
Instrument Driver Configuration

- **ID FILENAME** – Select ID to associate with instrument
- **SUB ADDRESS** – Defaults to –1
  – Only required for some instruments
  – Not HP-IB secondary address
  – Slot or card address
- **INCREMENTAL MODE** – ON – Only send required commands
  – OFF – Send all commands
- **ERROR CHECKING** – ON – Ask the instrument for any errors
  – OFF – Don't check for instrument errors

E2110+24D V208

`ID FILENAME` Designate Instrument Driver file to use with instrument.

`SUB ADDRESS` Defaults to `-1`.

Only required by some instruments, designates slot or card address.

This is *NOT* HP-IB secondary address.

`INCREMENTAL MODE` `ON`—Only send commands required to update state of instrument.

`OFF`—Send all commands.

`ERROR CHECKING` `ON`—Query instrument for errors after sending commands.

`OFF`—Don't check for errors.

Using Instrument Drivers Within Models

- Unconnected ID panel used interactively
- Terminals allow ID functions to be controlled
  - All components known within ID available via data input/output terminals
  - Field input/output allows selection of any feature present on panel

E2110+24D V209

You can use instrument drivers within a model to interactively control control an instrument. It need not be connected to any other objects.

By adding terminals to an instrument driver, the model can control the instrument. All instrument components known by an instrument driver may be controlled or queried via input or output terminals. You can also add terminals by selecting them from the instrument soft front panel display. To do so, select `Add Terminals --> Select Input Component` or `Add Terminals --> Select Output Component` from the instrument's object menu. Then simply click on the instrument field that you want to become a terminal.

Lab 9—HP3478 Frequency Response

Task 1

Measure and graph the input frequency response of the HP 3478A over the range of 0.1Hz to 10MHz. Use the instrument drivers for the HP 3314A and the HP 3478A.

Task 2

Characterize the speed difference between state and component drivers.

13

Using Direct I/O

Using Direct I/O

Direct I/O

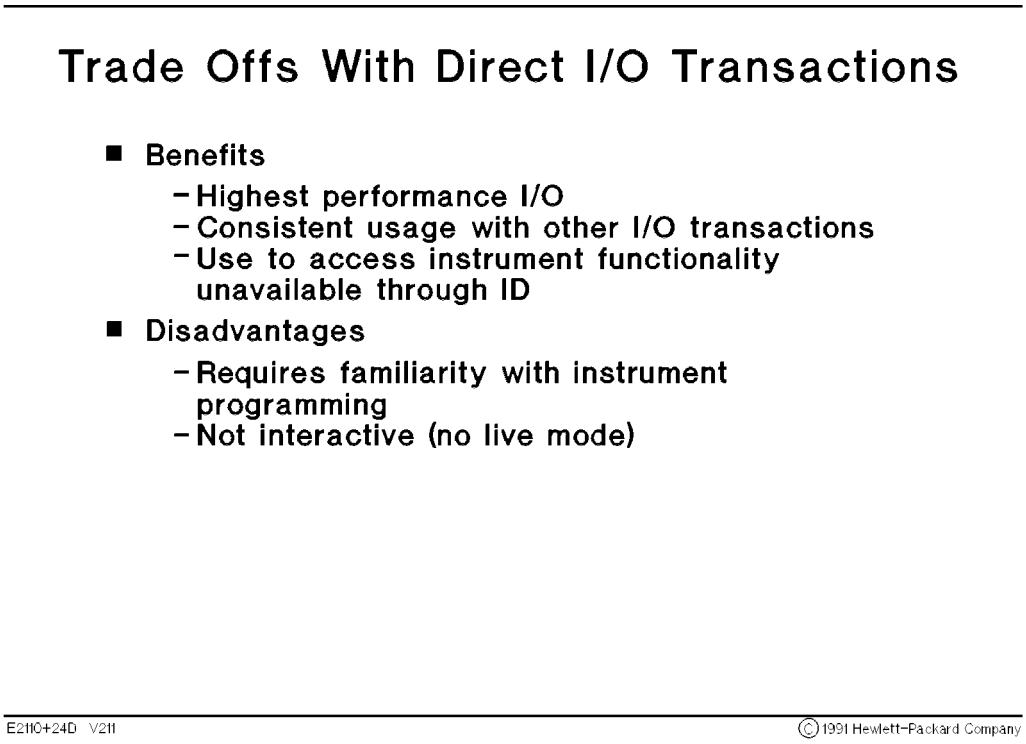
- **Full instrument I/O functionality via transaction objects**
  - **READ and WRITE data in all formats**
  - **SEND for fine control of data and command**
  - **EXECUTE for control of interface and device**
  - **Wait**

E2110+24D V210

But what if HP VEE does't have an instrument driver for your favorite instrument? HP VEE can still communicate with it using `Direct I/O` objects. These objects use the transaction box construct that we've seen before.

`Direct I/O` provides the following actions:

- `READ` and `WRITE` data to your instrument in all formats.
- `SEND` gives fine control of data and commands.
- `EXECUTE` controls the instrument interface and your device.
- `WAIT` until you're ready for the next command.



What are the trade offs of using `Direct I/O`?

`Direct I/O` provides the following benefits:

- Highest performance I/O
- Consistent interface with other I/O transactions
- Access to instrument functionality unavailable through instrument drivers

However, it also has the following disadvantages:

- You must be familiar with the instrument programming commands or have access to documentation.
- It is not interactive, ie., no live mode control.

Configuring Direct I/O

- Single instrument has separate configuration for Driver and Direct I/O
- Direct I/O configuration specifies
  - Terminators and EOL sequence
  - Formatting of array data
  - Conformance to IEEE 488 or 488.2
  - Information needed to save and restore instrument learn string

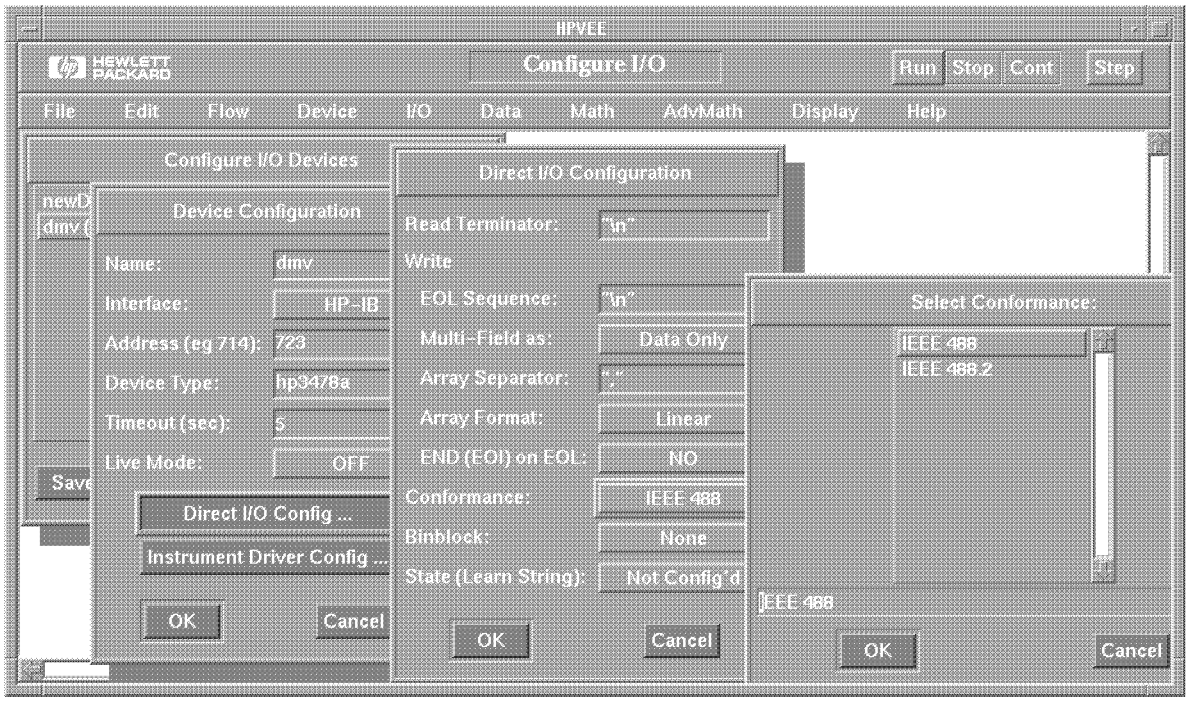
E2110+24D V212

Even though you have configured an instrument driver for a particular device, you must also configure the instrument for `Direct I/O`.

`Direct I/O` configuration specifies:

- Terminators and EOL sequence
- Format of array data
- Conformance or non-conformance to IEEE 488 or 488.2 (HP-IB)
- Information need to save and restore an instrument learn string

Screenshot - Direct I/O Configuration



This screen shot illustrates configuring a device for `Direct I/O`.

Transactions for Direct I/O

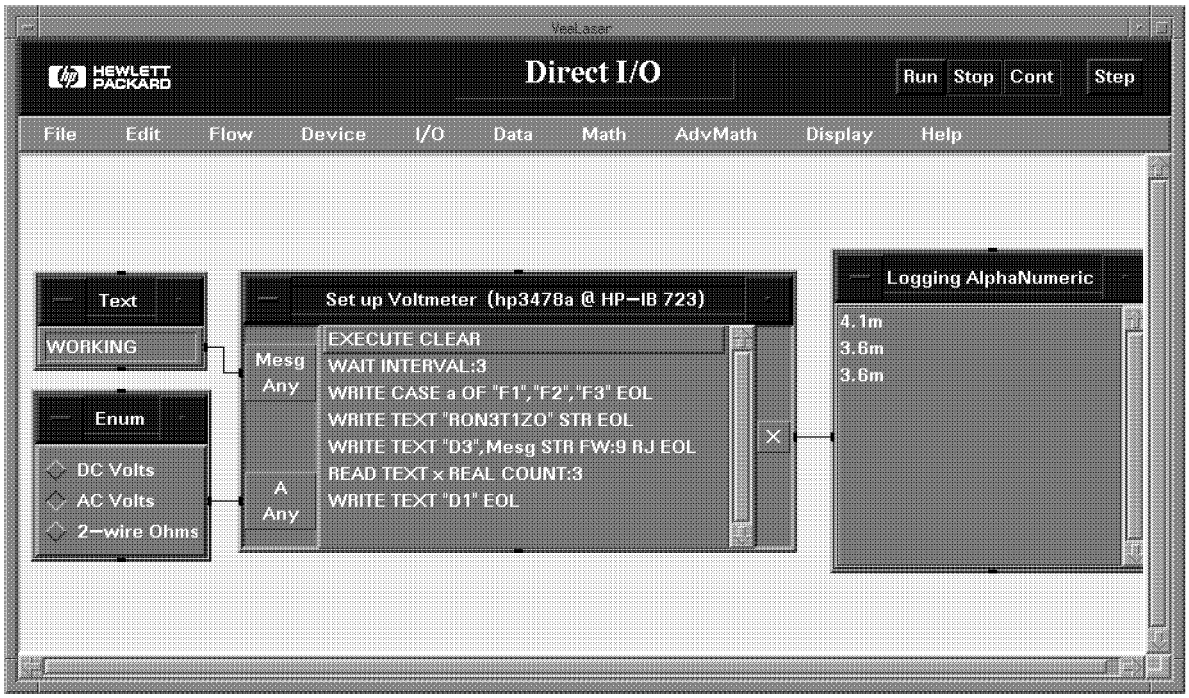
- **EXECUTE** – Sends addressed commands
  – CLEAR, TRIGGER, LOCAL, REMOTE, RESET
- **WAIT UNTIL SPOLL MASK** – Allows SPOLL operation and result compare within an instrument object
- **READ IOSTATUS** – Returns 2–bit value of GPIO, STS0 and STS1
- **WRITE** – STATE – Allows uploaded learn string to be written
  – IOCONTROL – Allows control of PCTL, CTL0, CTL1 lines

E2110+24D V214

Lets look at some of the details fo the `Direct I/O` actions.

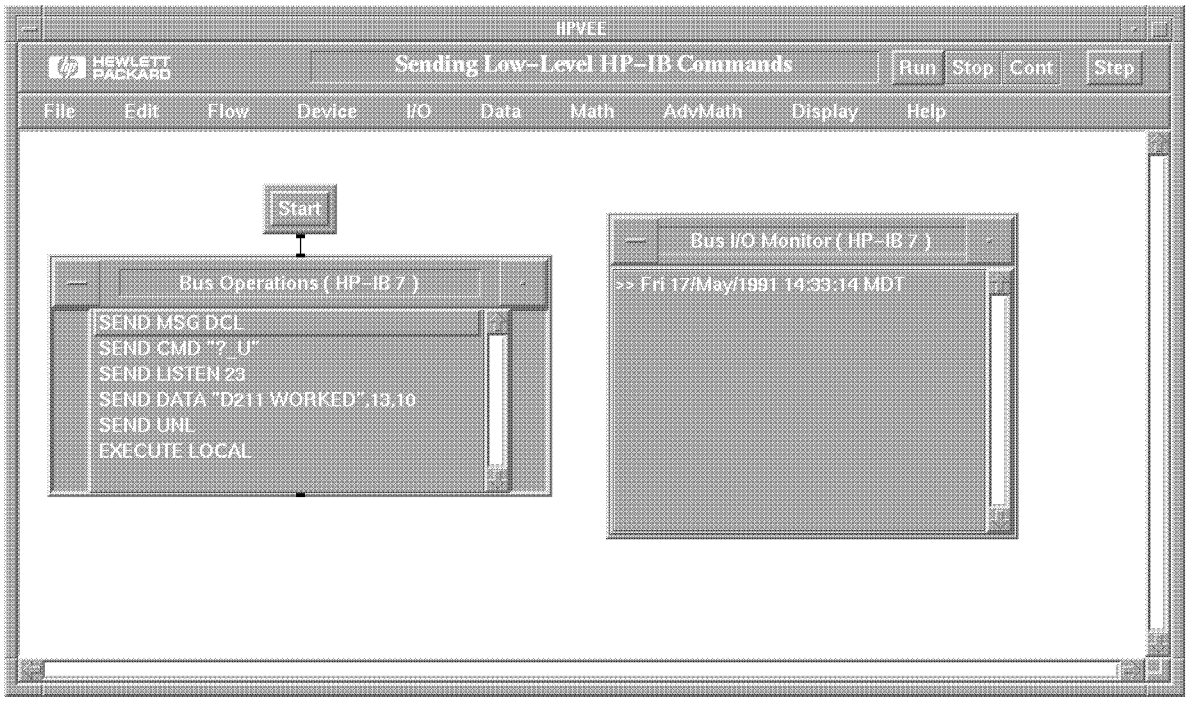
`EXECUTE`	Sends addressed commands `CLEAR`, `TRIGGER`, `LOCAL`, `REMOTE`, `RESET`
`WAIT UNTIL SPOLL MASK`	Performs an HP-IB serial poll, compares the result against a mask, then continues.
`READ IOSTATUS`	Reads the 2-bit value of the GPIO interface lines, `STS0` and `STS1`.
`WRITE`	`STATE` writes an uploaded learn string to the device.
	`IOCONTROL` controls the state of GPIO interface `PCTL`, `CTL0` and `CTL1` lines.

Screenshot - Using Direct I/O



This screen shot illustrates using `Direct I/O` to control an instrument.

Screenshot - Sending Low Level HP-IB Commands



This screen shot illustrates sending low level HP-IB commands with `Direct I/O`.

Using Advanced HP-IB Functions

- EXECUTE – Sends non-addressed (global) bus commands
  - ABORT, CLEAR, TRIGGER, REMOTE, LOCAL, LOCAL LOCKOUT
- SEND – Allows custom command/data transactions to be created
  - COMMAND – Data sent with ATN TRUE (command)
  - MESSAGE
    - IEEE-488 defined mnemonic commands Sent with ATN TRUE
    - DCL, TCT, etc.
  - DATA – Data sent with ATN FALSE (data)
- TALK, LISTEN, UNLISTEN, UNTALK, MTA, MLA, SECONDARY

E2110+24D V217

HP VEE also provides `Advanced HP-IB` functions for advanced control of HP-IB.

`EXECUTE` sends non-addressed (global) bus commands to all devices. `ABORT`, `CLEAR`, `TRIGGER`, `REMOTE`, `LOCAL`, `LOCAL LOCKOUT`

`SEND` sends custom command or data transactions.

`COMMAND` sends data with `ATN` line TRUE (HP-IB command)

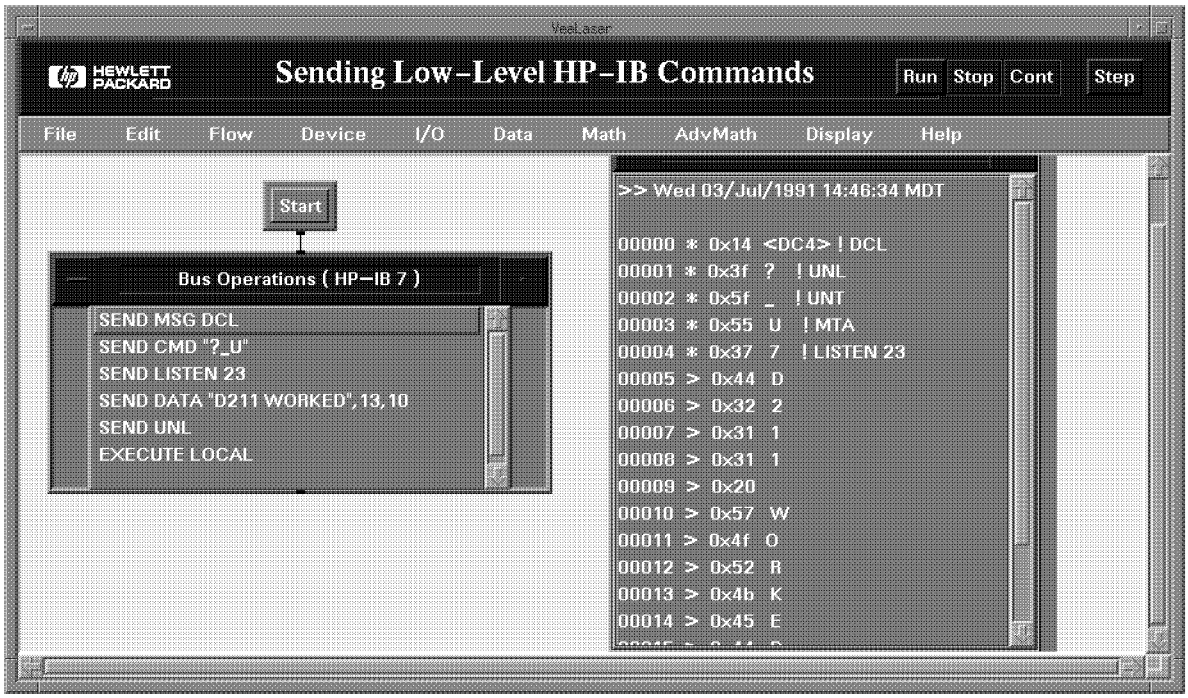
`MESSAGE` IEEE-488 defined mnemonic commands sent with `ATN` line TRUE. `DCL`, `TCT`, etc.

`DATA` data sent with `ATN` line FALSE (HP-IB data)

You can also use the commands:

`TALK`, `LISTEN`, `UNLISTEN`, `UNTALK`, `MTA`, `MLA`, `SECONDARY`.

Screenshot - Sending Low Level HP-IB Commands



This screen shot illustrates sending low level HP-IB commands with `Advanced HP-IB --> HP-IB Bus Operations`.

Using Advanced HP-IB Functions

- **HP-IB Serial Poll** – Command sequence initiated by controller which causes addressed instrument to return status byte
- **Wait For SRQ**
  - Suspends operation of current thread until SRQ line is asserted
  - SRQ shared by all instruments on bus
  - Must usually follow with SPOLL to determine source
  - Operation of independent threads continues

E2110+24D V219

`Advanced HP-IB` also provides some high level HP-IB functions.

`HP-IB Serial Poll` causes the controller to poll the addressed instrument for its status byte.

`Wait for SRQ` suspends the current thread until the `SRQ` line is asserted. Remeber, `SRQ` is shared by all instruments on the bus. You must poll each instrument to determine which one requested service.

All other independent threads continue operation even though the thread containing the `Wait for SRQ` objects suspends operation until it receives the `SRQ` bus message.

Maintaining Instrument State

- Direct I/O instrument objects can upload the state (learn string) of instruments
  - State is maintained with the object
    Multiple objects share same learn string
  - Does not interact with driver state
- Typical use:
  - Set up instrument with State Driver or front panel
  - Upload learn string
  - Use learn string to preset instrument state
  - Use Direct I/O to make incremental changes

E2110+24D V220

So how do you maintain the state of an instrument?

`Direct I/O` objects can upload an instrument state or learn string from the instrument. The object maintains the state and shares it with other objects. It does not interact with the instrument driver state.

To do this you would:

- Set up the instrument with a `State Driver` or the instrument front panel.
- Upload the learn string.
- Use the learn string to preset the instrument state.
- Use `Direct I/O` to make incremental changes to the instrument state.

Bus Monitor

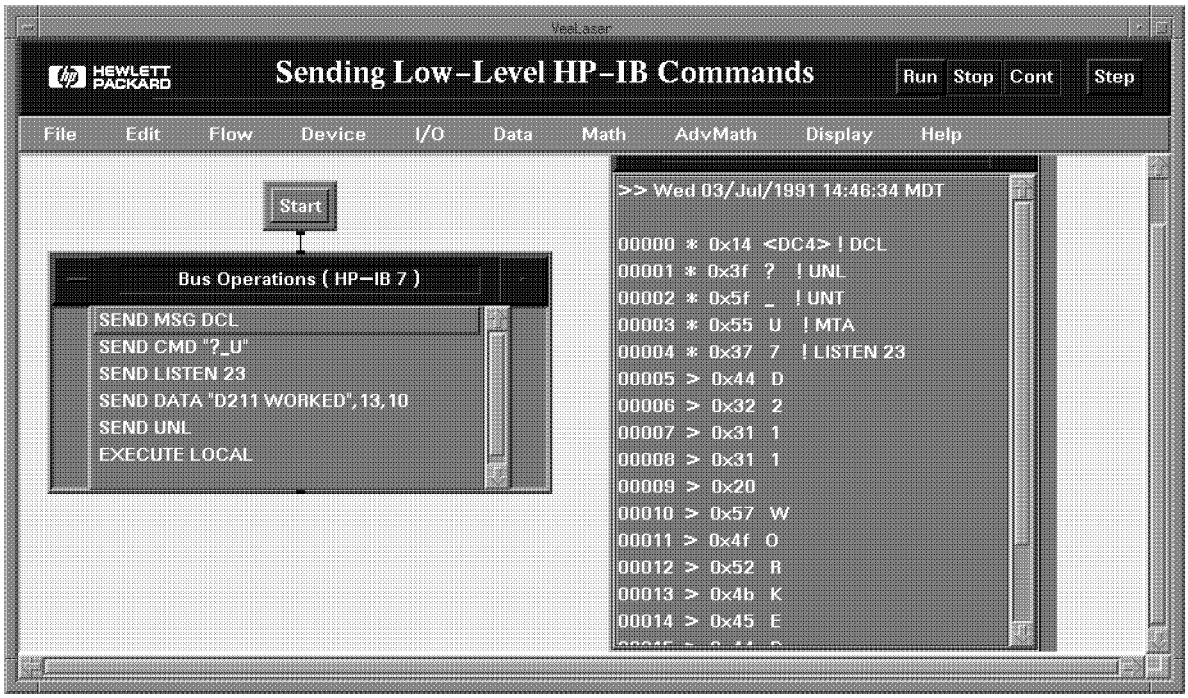
- Works with HP-IB, GPIO, RS-232
- Records all traffic
  - Generated by Driver or Direct I/O
  - Received by controller
- Data is timestamped, displayed in text or hex, I/O direction indicated, and command bytes interpreted

E2110+24D V221

To help you debug your instrument control models, HP VEE provides the `Bus Monitor` object. The `Bus Monitor` works with all of the interfaces configured in your computer, recording traffic generated by instrument drivers or `Direct I/O`, or received by the controller. The monitor timestamps the data, displays it as hex digits, indicating the I/O direction. It also interprets command bytes for you.

Note The `Bus Monitor` only records I/O traffic related to HP VEE. It does not monitor or record traffic generated or received by other devices or programs.

Screenshot - Bus Monitor



This screen shot shows an example of data displayed on the `Bus Monitor`.

Lab 10a—Custom Instrument Panel

Task

Create an object that acts as a custom instrument panel for the HP 3478A. The finished model should allow interactive control of the following functions:

- Function
- Range
- Display
- Trigger
- Autozero

As any value is entered, display the resultant measurement. The finished model should be within a `UserObject` (on a panel). Use at least one `Enum` object.

Lab 10b—Instrument Interrupts

Task

Create a model that allows the operator to:

1. Manually take measurements with an HP 3478A.
2. Use `Direct I/O` to establish the instrument's initial setup.
3. Initiate readings from the front panel of the instrument.
4. Append each reading to a file, along with a timestamp.
5. Graph the data at any time with the press of a button.

Hint

To set the HP 3478 for Front Panel SRQ, send the command `KM20`.

14

Using Named Pipes & HP BASIC/UX

Using Named Pipes & HP BASIC/UX

Interprocess Communication

- Multiple HP-UX processes to work in concert on a single problem
  - Individual processes are less complex
  - Individual processes may be optimized for task
  - Operating system facilities used instead of being reinvented
    - Data buffering
    - Process priority
    - Virtual memory
    - Concurrency of operation
- Benefit: Complex systems built from less-complex modules
  Less coupling means easier maintenance

E2110+24D V223

You will occasionally want several processes to run at the same time, passing information back and forth to each other. In this manner, each process is less complex and may be optimized for an individual task. You also take advantage of operating system facilities, instead of reinventing each task, such as:

- Data buffering
- Process prioritization
- Virtual memory
- Concurrent operations

By doing so, you build complex systems from less-complex modules which are easier to maintain.

Why IPC?

- Data exchange or sharing between processes
- Synchronization of concurrent processes
- IPC trades added complexity of data handling for reduced complexity of program structure
  - Synchronizing data exchange between programs easier than handling asynchronous control events in single process

E2110+24D V224

So why use Interprocess Communication? It provides an effective method for transfering data between programs. You can also use it to synchronize concurrent processes. By doing so, you trade the added complexity of passing data between programs for the reduced complexity of a large program structure. Its easier to synchronize data exchange between programs than to handle asynchronous control events in a single program.

IPC Facilities

- HP VEE implements HP-UX file system IPC only
  - Ordinary files
  - Pipes
- Other methods are very specialized - Access via HP-UX Escape object if required
  - Shared memory
  - Semaphores
  - Message queue
  - Signals

E2110+24D V225

HP VEE implements interprocess communication through the HP-UX file system only. It uses either normal files or pipes.

You may use other methods through the `HP-UX Escape` object if you find them necessary. These other methods include:

- Shared memory
- Semaphores
- Message Queue
- Signals

Using Files for IPC

- "Unlimited" capacity
- Unlimited number of processes may access
- Programs must agree on arbitrary conventions for format and synchronization
  - Auxiliary files often used as lock files
- Excellent performance if reader and writer can share file buffer
  - Lightly loaded system
  - Moderate amounts of data
- Poor performance if physical file system involved

E2110+24D V226

When you use HP-UX files for communication, you have unlimited data capacity, limited only by the system resources. An unlimited number of processes can access these files as well. Each program must agree on arbitrary conventions of format and synchronization. For example, you may use auxilary files for lock files or to indicate the state of file information.

This method provides excellent performance if both the Reader and Writer can share the file buffer. This happens on a lightly loaded system or with small amounts of data. Performance degrades rapidly when the physical file system becomes involved.

Using Pipes for IPC

- Pipes enforce FIFO message order
  - Multiple processes may write or read
  - Data can be read once ONLY
- Named pipes are created and accessed as part of file system
  - mknod mypipe p (sysadmin only)
  - mkfifo mypipe (user)
  - Exist independently of any processes
- Arbitration for multiple readers on pipe
- Pipes must exist locally (not NFS mounted)

E2110+24D V227

HP VEE provides access to pipes as an alternate communication link. Pipes enforce FIFO message order. Multiple processes can read from or write to pipes. However, you must remember that you can only read the data once.

The file system creates and accesses named pipes. Use the command

```
mknod mypipe p
```

or

```
mkfifo mypipe
```

to create the pipe `mypipe`. It then exists independent of any process.

Note that the pipes used with HP VEE must exist locally. They cannot be NFS mounted.

Using Named Pipes

- Capacity of pipes is limited (4K–8K typical)
  - Writing to full pipe "blocks" writer
  - Reading from empty pipe "blocks" reader
- Synchronizing is reliable if only one each reader/writer
  - Kernel suspends processes until both reader and writer exist
  - Blocking will synchronize later if needed

E2110+24D V228

Here are a few things to remember while using pipes:

- Pipes have a limited capacity, typically 4 to 8 Kbytes.
- Writing to a full pipe “blocks” the writing processes, causing it to wait until another process reads data from the pipe.
- Reading from an empty pipe “blocks” the reading processes, causing it to wait until another process writes data to the pipe.
- Synchronizing processes with pipes is only reliable if there is only one reading process and one writing process. The kernel suspends processes until there is both a reader and a writer. The kernel will block one of the processes if needed for synchronization.
- Pipes must exist locally. They cannot be NFS mounted.

To/From Named Pipes

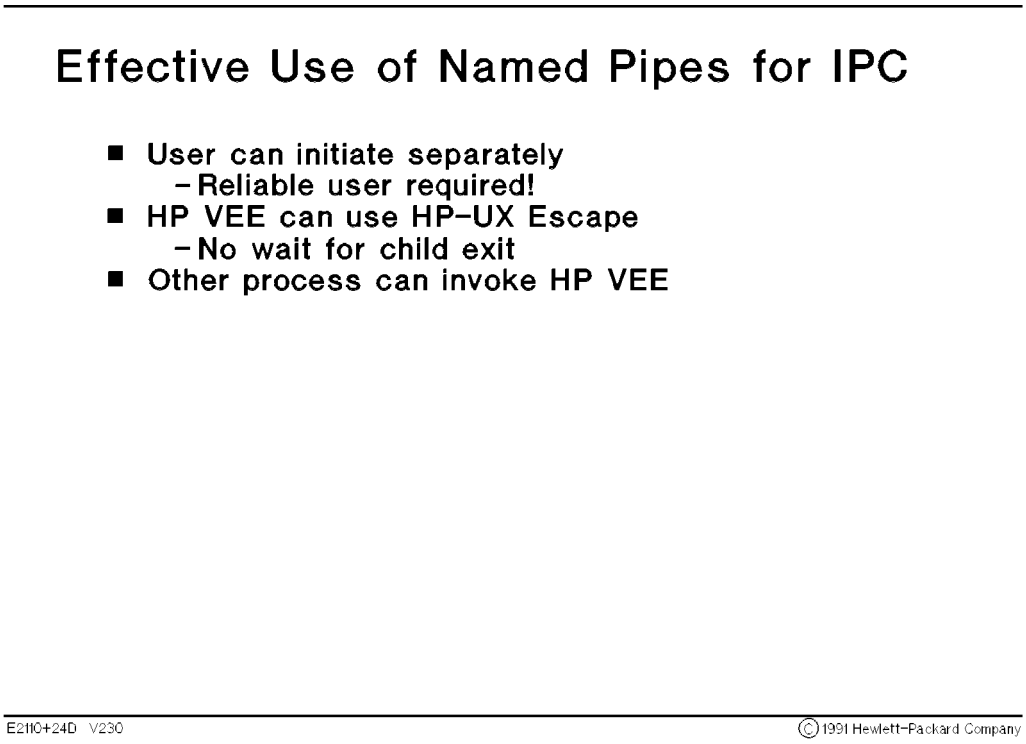
- Pipes are automatically created by first attempt to open
  - Read pipe opened read-only
    - Allows EOF detection
  - Write pipe opened write-only
- Pipes are closed upon termination of entire model
  - Not after each object deactivates
  - Never deleted
- Pipes opened as "blocking"
  - Needed for synchronization
  - Can hang waiting for data or space available

E2110+24D V229

HP VEE uses the `To/From Named Pipe` object to communicate with pipes. It automatically creates the pipe on the first attempt to open the pipe. HP VEE opens a read pipe as read-only, allowing it to detect an `EOF`. It also opens a write pipe as write-only.

HP VEE closes pipes upon termination of the entire model, rather than at the termination of an object. It also never deletes a pipe.

Also note that HP VEE opens pipes as "blocking," for synchronization. This may hang your HP VEE process or other processes while they wait for data or space in the pipe.



To most effectively use named pipes with HP VEE remeber:

- The user can initate named pipes and processes separately
- You need a *reliable* user process.
- HP VEE can start other processes with the `HP-UX Escape` object. Remember to select `Wait for child exit: NO`.
- Other processes can start HP VEE

Requirements for Cooperative IPC

The Non-VEE Process Should:

- **Open pipes read-only or write-only**
- **Look for EOF conditions on each read**
- **Trap SIGPIPE to help diagnose mysterious failures**
  - **Issued by kernel if write attempted after reader closes**
- **Use UNbuffered write operations, or flush buffers prior to any READ after WRITE**
- **Use single reader/single writer model**
  - **Ideally, interleave read/write operation**

E2110+24D V231

To be successful when using IPC's with HP VEE the non-HP VEE process should:

- Open pipes as Read-Only or Write-Only, *not* bi-directional
- Check for `EOF` condition on each read.
- Trap a `SIGPIPE` signal to help diagnose mysterious failures. The kernel issues this signal when you attempt to write after the reader closes.
- Use *unbuffered* write operations or flush the buffers prior to any read after a write.
- Use a single reader/single writer model, ideally interleaving the read/write operation.

Escapes to HP BASIC/UX

- Performed as two steps to avoid multiple BASIC bootup
  - Initialize HP BASIC/UX
    - Invokes BASIC
    - Loads and runs requested program
  - To/From BASIC/UX
    - Equivalent to To/From Named Pipe

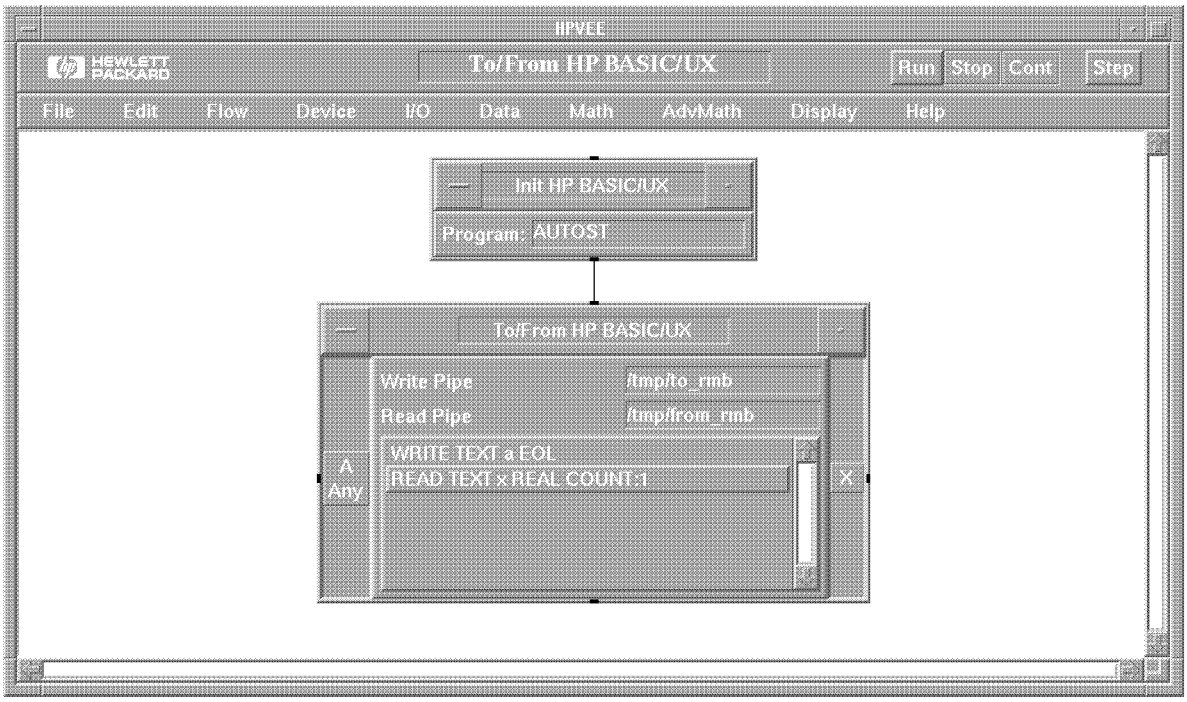
E2110+24D V232

HP VEE also provides a specialized `To/From Named Pipe` object to communicate with HP BASIC/UX. It calls HP BASIC/UX in a two-step process to avoid booting HP BASIC multiple times.

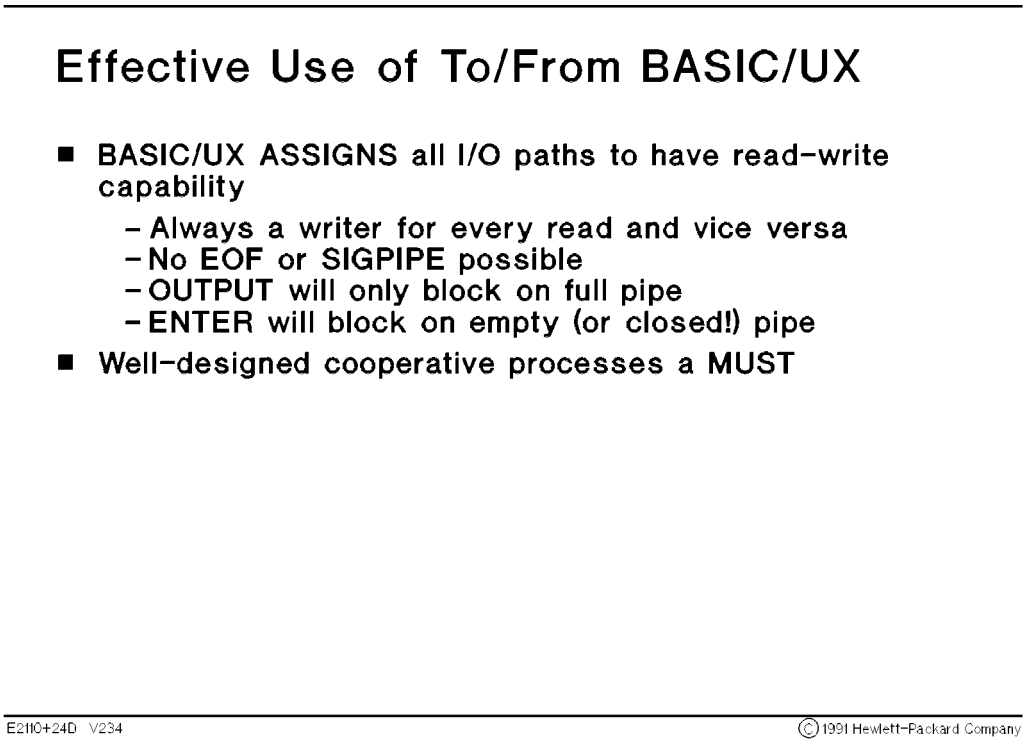
`Initialize HP BASIC/UX` starts HP BASIC and then loads and runs the requested program.

`To/From HP BASIC/UX` is simply a `To/From Named Pipe` object with the parameters filled in to work with HP BASIC/UX.

Screenshot - To/From HP BASIC/UX



This screenshot shows how the two HP BASIC/UX objects work together.



For more effective use of `To/From HP BASIC/UX` remember the following:

- HP BASIC/UX `ASSIGNS` all I/O paths as Read-Write capable.
- You must always have a writer for every reader and vice versa.
- Checking for `EOF` and `SIGPIPE` is not possible.
- An `OUTPUT` with only block on a full pipe.
- An `ENTER` will block on an empty or *closed* pipe.

Remember to be especially careful to write well-designed and cooperative processes.

Using TRANSFER with Named Pipe

- Creates subprocess (rmbxfr) to read/write pipe
- Main BASIC/UX process remains unblocked
- Use EOR/EOT interrupts to signal data availability
- Use ON DELAY to act as watchdog timer
- Advantages:
  - No blocking
  - Multiple sets of TRANSFER and pipe, can connect to single RMB
- Disadvantages:
  - Not as easy to implement as OUTPUT/ENTER

E2110+24D V235

Perhaps the most effective means of transferring data between HP VEE and HP BASIC is the `TRANSFER` process within HP BASIC. `TRANSFER` creates a subprocess, `rmbxfr` to read from and write to the pipe. The main HP BASIC/UX process remains unblocked. Because of this you can use EOR/EOT interrupts to signal data availability. You can also use `ON DELAY` as a watchdog timer to timeout an errant `TRANSFER` process.

This technique has the following advantages and disadvantages:

- Advantages:
  - No process blocking
  - You can have multiple sets of `TRANSFER` and pipes connected to a single HP BASIC process.
- Disadvantages:
  - Its not as easy to implement as `OUTPUT/ENTER`.

Lab 11—Exploring Named Pipes

Background

Interprocess communication with named pipes involves cooperation between two HP-UX processes as well as the kernel.

A `c` program (`/labs/ToFromPipe.c`) is supplied which reads data from a pipe, prompts for input, and sends the response to another pipe.

Tasks

1. Create a model to talk to the `c` program.
2. Explore the effects of starting and stopping the `c` program and HP VEE independently.
3. What is the largest amount of data ever held in the pipe?
4. Write an HP BASIC/UX program that emulates the `c` program. How does its behavior differ from the `c` program? Why? Test the hypothesis by modifying the `c` program.

Lab 12—Benchmarking HP BASIC/UX Escapes

Task

Benchmark the data throughput rate of a named pipe.

1. Write an HP BASIC/UX program which writes progressively larger binary blocks of data to a HP VEE model through a pipe. The program should write in response to a prompt from the HP VEE model.
2. Plot the throughput rate versus block size. Block size may be up to 1MByte in size.

15

Instrument Application Development Techniques

Instrument Application Development Techniques

Application Development: Building Complex Models Visually

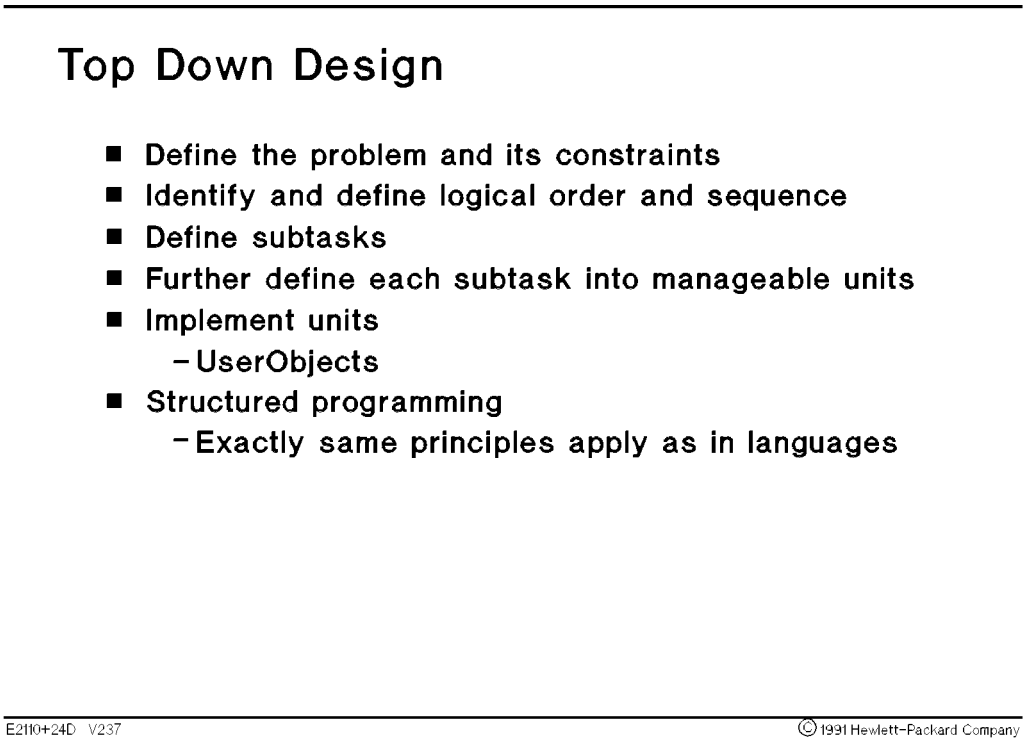
Benefits of Using HP VEE

- Time spent solving the problem – no time spent remembering syntax
- Development time is decreased
  - No edit, compile cycle
  - Changes made quickly
- Multifunctionality of objects based on data types and shapes
- Inherent user interface
  - Visual orientation
- Automatic Data Typing

E2110+24D V236

As we explained previously, HP VEE helps you create solutions to your programming problems quickly and easily. You spend more of your time actually defining and describing your problem, rather than trying to find syntax errors or missing semicolons. You will also find that your development time is much shorter since there is no edit-compile cycle. You simply build your model on the screen and then execute it. The HP VEE paradigm also provides an inherent user interface.

These same ideas apply when working with instrument control applications.



As before, remember the points of top-down design.

- Define the problem, including all of its contraints.
- Identify a logical order and sequence for each of the tasks within the problem.
- Define each subtask within each task. Note how well this fits into HP VEE's paradigm of nested `UserObjects`.
- Continue defining each subtask into manageble units.
- Now implement each of these tasks as a `UserObject` within HP VEE.

Levels of Complexity: HP VEE-Test

- Test and measurement: data flow
- Sequential flow along data path
  - Sequence determined by data type, shape
- Few objects; mostly I/O and display
- Usually more emphasis on instrument, particularly with direct I/O
- Data acquisition: high sample rates
  - Optimization
- More analogous to the Basic world

E2110+24D V238

Test and measurement models lend themselves well to HP VEE since it implements data flow directly. Think of how data needs to flow from one part of your system to another, or from one function to another.

When working with instruments you will find it easiest to work with just a few objects as you begin developing your model, mostly instruments and display objects. Get the instruments setup and collecting data properly. Then develop the data analysis portion of your model.

When developing with high speed data acquisition systems, work to optimize the speed of the I/O functions. You may want to develop external programs to speed the collection or conversion of data.

Lab 13—Develop Your Own Application

Task

Create an HP VEE-Test model applicable to your own work assignments. Use any objects necessary. You may incorporate objects, `UserObjects`, or models created in previous lab exercises, or models from the `examples` directory.

Remember to keep the user in mind as you create the user interface for this model.