TestPoint™ QuickStart

Limited Warranty

Capital Equipment Corporation (CEC) warrants the physical diskettes and documentation enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date. The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective diskettes or documentation, and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims. In no event shall CEC's liability exceed the purchase price of the product.

TestPoint QuickStart
Part number 04000-90100, vol. 1
Fourth edition
02 01 00 99
10 9 8 7 6 5 4 3 2 1

Capital Equipment Corporation
900 Middlesex Turnpike Building 2.
Billerica, Massachusetts 01821
(978) 663-2002

Setup

- Insert TestPoint CDROM.
- In the Windows Program Manager, choose File, Run from the menu.
- Enter "D:SETUP" (use appropriate drive letter or Browse)
- Follow the directions on the screen. You will need to choose a hard disk directory for TestPoint. The default is "C:\TESTPT".

The SETUP program copies in the TestPoint files and adds new icons to the Windows Program Manager.

Library Installation

GPIB and other libraries are optional. They are installed by running SETUP from the first library diskette, just as for the main TestPoint installation. You can choose to install any or all of the libraries.

System Requirements

- Microsoft Windows 95 or later
- A PC-compatible computer with at least an 80386 processor
- At least 8 megabytes of system RAM (16 recommended)
- At least 8 megabytes of available disk space
- A mouse for editing (not required for TestPoint runtimes)

Removing TestPoint (Uninstall)

To reverse the installation process, and remove the TestPoint files from your system:

First, go to the Windows Program Manager and minimize the TestPoint group window (use the down arrow button near the upper right corner of the group window). Then, select the TestPoint group icon, and use the Del key. Click OK to confirm that you want to delete the entire program group.

Next, go to DOS and run the batch file UNINSTALL:

```
C:\TESTPT>  UNINSTALL
```

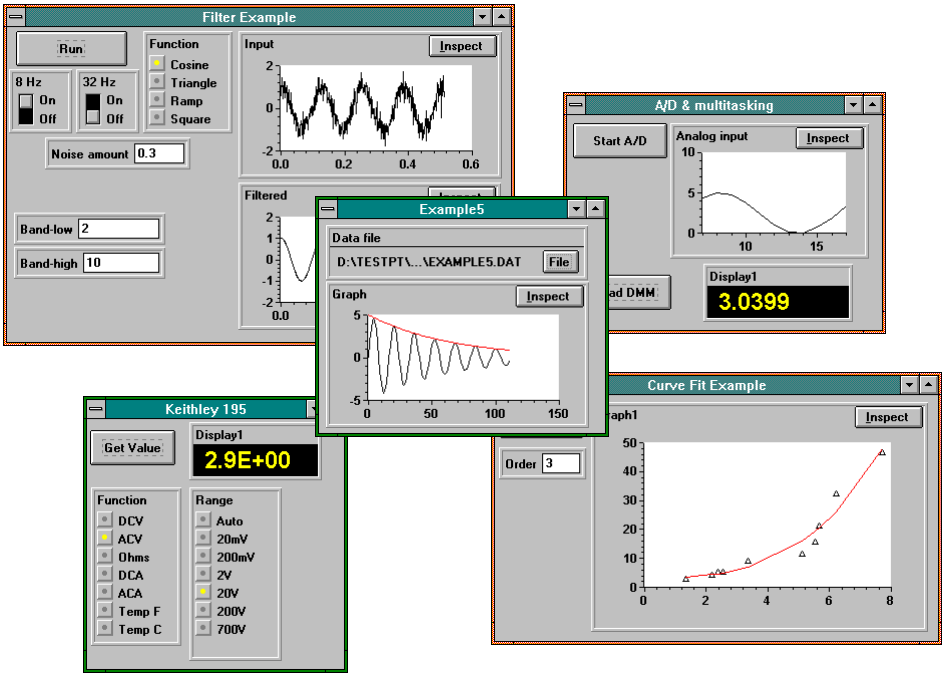
This will delete all files and directories for TestPoint.

Working Model Demo version

If you have the free working model demo version of TestPoint, certain features of the full product are not available:

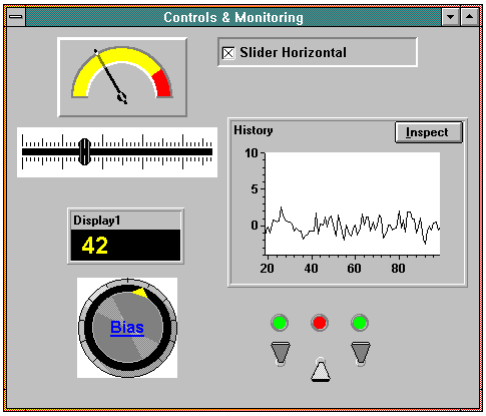
You cannot save application files.
You cannot control actual measurement hardware (GPIB,A/D,etc.).
You cannot use Dynamic Data Exchange.
You cannot call external code with the Code object.

Introduction



What TestPoint Is

TestPoint is a tool for creating custom test, measurement, and data acquisition applications.



TestPoint includes features for controlling external hardware, creating user interfaces, processing and displaying data, creating report files, and exchanging information with other Windows programs.

TestPoint is a new and unique alternative to conventional programming languages. In TestPoint, you select objects that model the components of your application, then combine the actions they make available using drag-and-drop mouse operations to create a custom solution.

TestPoint has all the power and flexibility of a programming language. There are none of the limitations associated with "choose from a menu" programs, because TestPoint has a full set of features for controlling program flow, accessing custom hardware, and extending the basic package with external code, add-ons, and data exchange links with other programs.

What TestPoint Can Do

Instrument Control

- Instrument libraries for hundreds of popular devices
- Support for GPIB (IEEE-488), RS232, and RS485 devices.
- Support for VXI devices, through a GPIB controller.

Data Acquisition

- Support for analog input and output (A/D, D/A), digital I/O, and custom I/O port devices.
- High-speed background A/D.
- Strip charts, bar indicators, numeric displays.
- Alarm limits.
- Disk logging.

Analysis

- FFT, digital filtering, waveform smoothing.
- Curve fitting, polynomials, interpolation.
- Statistics, including mean, deviation, median, histograms, ...
- Vector and array math, including inverse, determinant, ...

Presentation and Reports

- Line/symbol graphs, XY graphs, strip charts, bar graphs.
- Numeric and analog bar displays.
- Indicators and bitmapped pictures.
- Printed report generation

User Interface

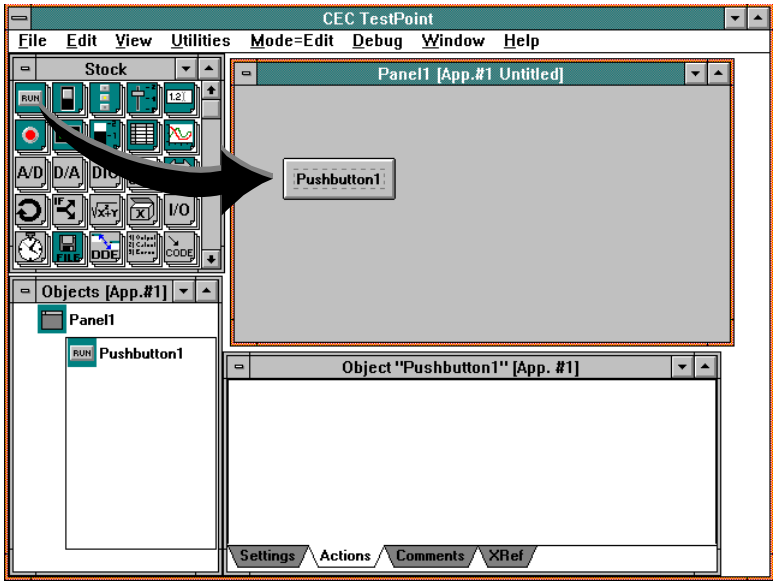
- Pushbuttons, switches, selectors, sliders, entry fields.
- Multiple panels.
- Works with industry-standard custom controls (ActiveX).

Dynamic Data Exchange / Object Linking & Embedding

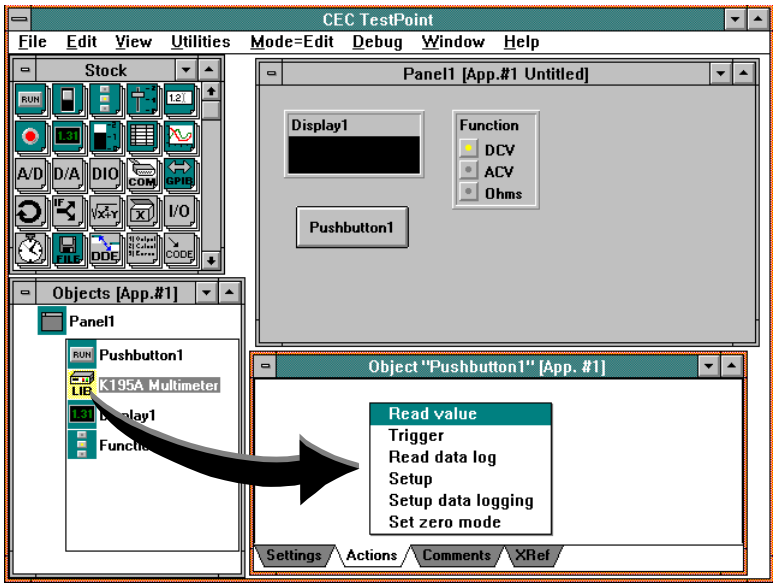
- Send/receive numbers, text, and charts to spreadsheets, word processors, databases, and math programs.
- Display graphs and pictures processed by other applications.

How You Work With TestPoint

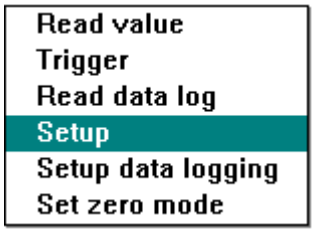
You begin by simply dragging TestPoint objects like buttons, displays, graphs, GPIB devices, A/D boards, math formulas, and disk files from a stock window into your application:



Then, you drag the objects to an action list:



and choose actions from a menu, to tell TestPoint the steps involved in carrying out your application's task:



TestPoint creates action lines, in a clear, readable format, which describe the actions you've chosen:

1)	Setup	K195A Multimeter	Function="DCV"
2)	Read value	K195A Multimeter	
3)	Set	Display1	to K195A Multimeter

There's none of the extra work associated with conventional programming languages. You don't have to declare variables, type long statements, carefully spell out language keywords, and follow syntax rules about semicolons and "end" statements. Instead of making you do the tedious work to meet the computer's needs, TestPoint lets you concentrate on choosing the objects and actions that make up your application.

While making application authoring easy, TestPoint doesn't give up any of the power or flexibility of a programming language. TestPoint matches and exceeds the capabilities of tools like C, Pascal, or Visual BASIC, as well as "wiring-diagram" style tools.

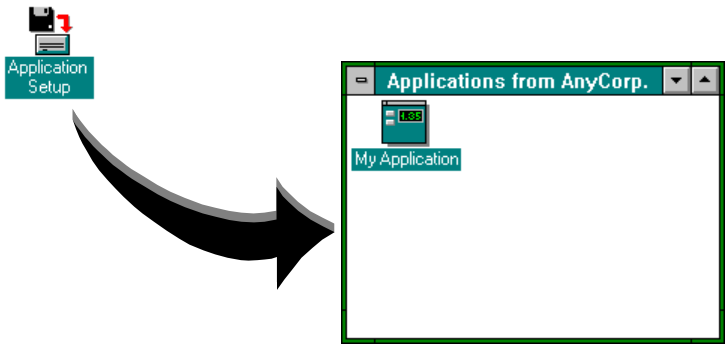
TestPoint uses the latest approach to program structuring by grouping actions and data into objects. Objects support all types of data: numbers, strings, vectors, arrays, and lists. But data types don't get in your way. Automatic data typing and conversion handle most cases, and manual data formatting and conversion are available when needed by clicking the mouse.

TestPoint action lists support sequential execution, loops, and conditionals, as well as the ability to respond to events and interrupts and to support multitasking. TestPoint action lists can activate other action lists, to give fully modular programming.

How TestPoint Works For You

When you've built your application, TestPoint packages it as a runtime, by just choosing a menu command.

A TestPoint runtime is an independent, executable Windows application, complete with professional installation program.



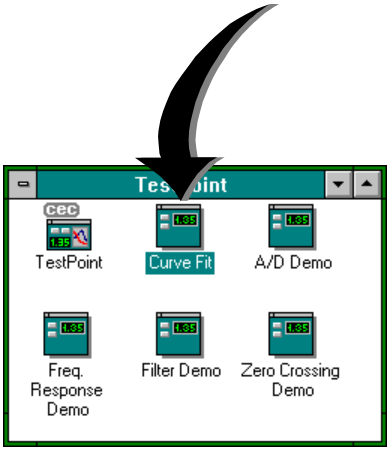
You can use it, give it to colleagues or your production line, or sell it as a commercial product. Your users don't need TestPoint to run the application, and there are no runtime fees or license restrictions.

Tutorial

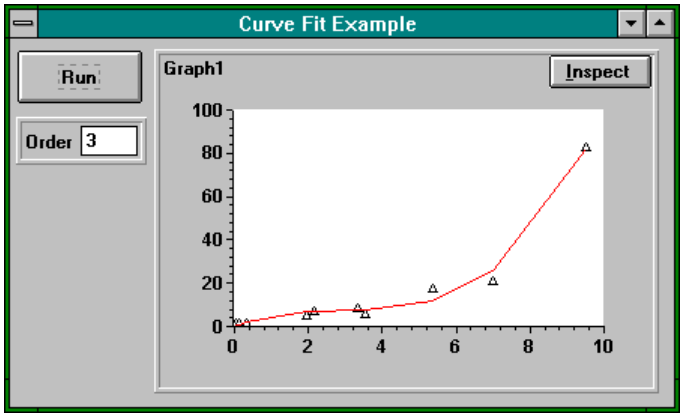
Run some sample applications

If you haven't installed TestPoint yet, do so now. Instructions are at the front of this book.

Double-click the mouse on the "Curve Fit" icon in the Windows Program Manager:

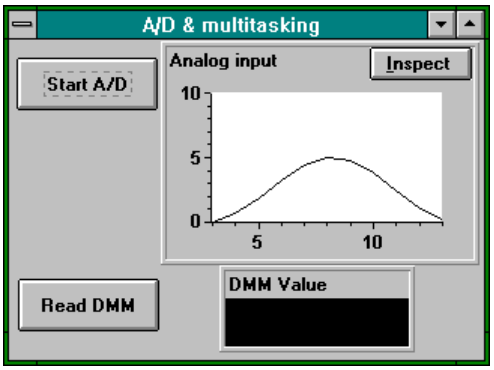


Now, click on the "Run" button:

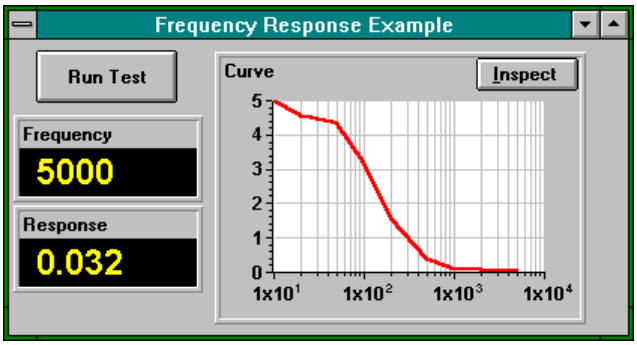


This sample application fits a polynomial equation to a set of data points and graphs the result. You can choose the order of the polynomial by entering a number in the "Order" field.

Now, try the "A/D demo" application, and click on its "Start" button to see analog input and strip charting:



Double-click the "Freq. Response" icon. Note that this application starts automatically and steps through a set of frequencies, then graphs the measured voltage responses:

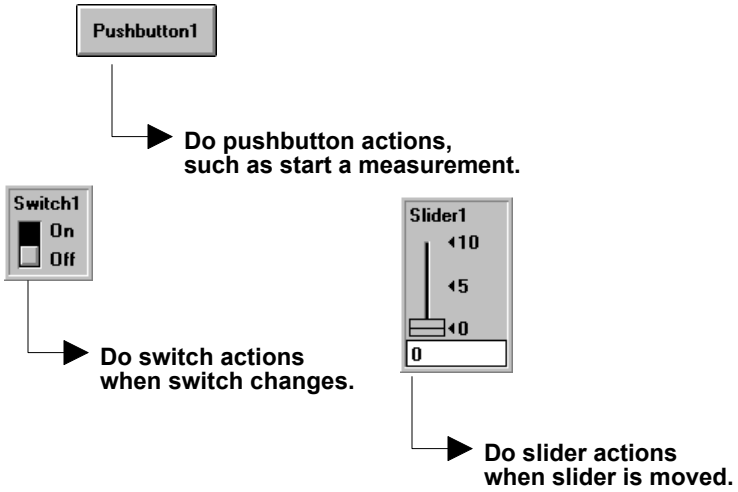


These examples show a few of the capabilities of TestPoint. They also show you how a TestPoint application will look when it is complete.

Event-Driven Programming

TestPoint applications run in Microsoft Windows, and they respond to the mouse or keyboard. The user of an application can enter values into TestPoint panels in any order desired. TestPoint responds to the inputs based on the instructions given when the application was built.

This is known as **event-driven programming**, because the program responds to events from the user, rather than asking the user for lines of input text. There is no "main" program in TestPoint. Whichever event occurs first determines the TestPoint action list that is executed.



Event-driven programming is one of the few concepts you need to master to use TestPoint, so it's worth taking a few moments to follow what goes on when a TestPoint application is running.

TestPoint, like most other Windows programs, spends most of its time idle, waiting for an event to occur. An event may be a key press, a mouse movement or click, a timer interval, or some type of hardware signal such as an A/D board sampling clock.

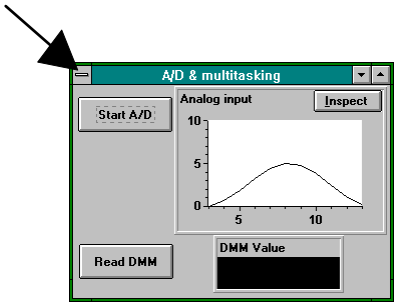
Each TestPoint object has an event it responds to, often by changing a data value and by executing a series of action steps specified by you, the application designer.

For example, the Data-Entry object responds to entry of a new value. The Slider object responds to movement of the slider using the mouse or the keyboard. The A/D object responds to analog input samples arriving in the computer.

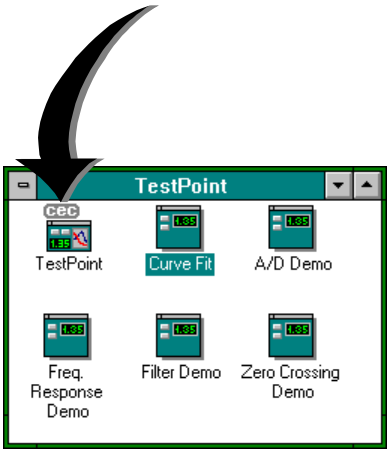
The steps that are executed when an event occurs are completely up to you. Each TestPoint object makes a selection of actions available, to provide control of hardware, math processing, graphing, and so forth. The order in which you combine these actions, triggered by events, determines the task carried out by your application.

Choose a tutorial path

If you have TestPoint applications running, exit them now by double-clicking on the close box at the upper left corner of each application's window.



Now, start the TestPoint development environment by double-clicking on the "TestPoint" icon:



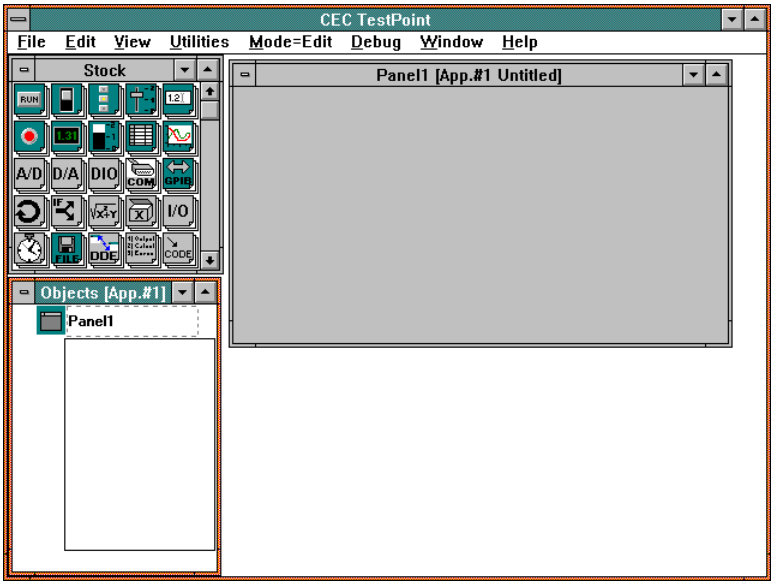
(You need to have closed the TestPoint applications, and you should install your execution control key on a printer port before starting TestPoint. If you have the free working model demo version of TestPoint, no key is used.)

The tutorial is divided into three paths. Each will teach you the fundamentals of building applications in TestPoint. Choose the application area which best fits your interests:

- **Automated Test** **see page 17**
- **Data Acquisition** **see page 35**
- **Analysis** **see page 55**

and skip ahead to the appropriate section...

When you start the TestPoint development environment, the editor window will appear, with a new, blank panel for you to fill in:



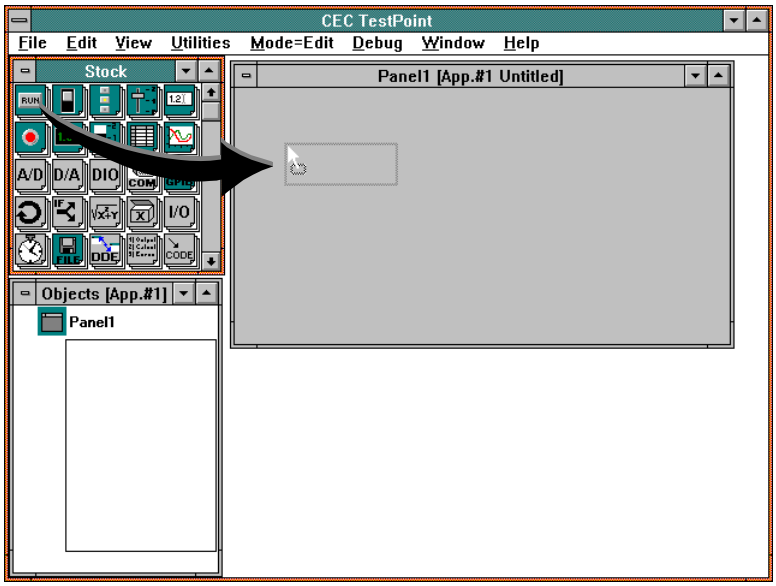
There are four areas in the TestPoint editor: the panel, the stock window, the object list window, and individual object windows which appear as you add objects. The use of these windows will be explained in more detail in the sections that follow. First, though, we'll run once through the steps of creating a new application.

The application we create will read a voltmeter when a pushbutton is clicked and display the result.

Add a pushbutton

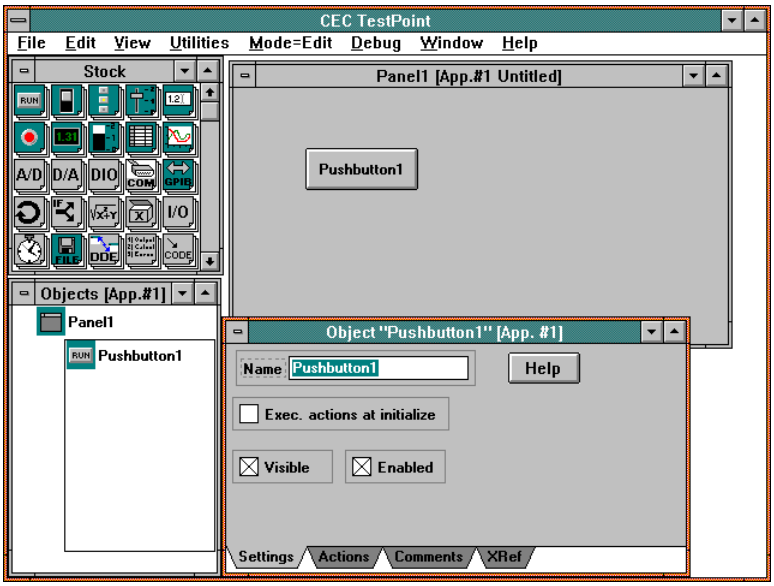
Drag a pushbutton  **from the stock window to the panel.**

(dragging means to place the mouse pointer over the source, push the left mouse button down and hold it, then move the mouse to the destination and release the button).



New objects are added to TestPoint applications by dragging and dropping from the stock.

When you release the mouse button, the pushbutton object will appear on the panel. A window titled "Object Pushbutton1" will also appear, to let you enter information about the new object.



The name of the new object starts out highlighted and ready to be replaced.

Type in a new name for the object: "Run Test".

The default values for the other settings are OK. You can click on any other window or on the "Actions" tab to dismiss the settings window.

You can double-click the object or its icon in the object window to re-open its settings window.

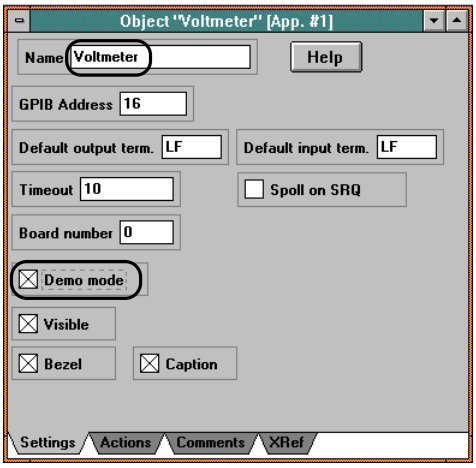
Note that the new object appears not only on the panel, but also in the object window, and that its action list appears automatically.

Add the GPIB voltmeter

**Drag a GPIB object 

from the stock to the panel.**

In the settings window, enter the name "Voltmeter", and also click the checkbox for "demo mode".



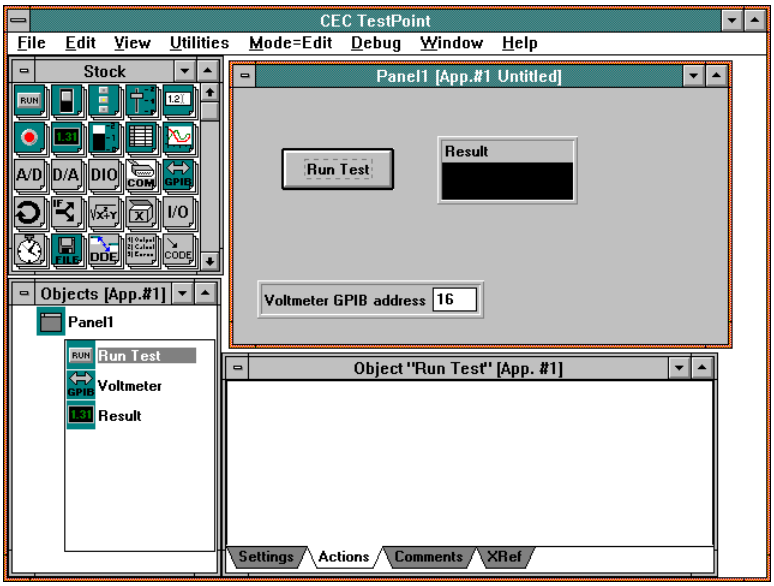
Demo mode is available as a setting on objects which access hardware I/O devices. It allows you to run your application without the hardware actually being present, for testing purposes.

Add a numeric display

Drag a display object  **from the stock to the panel.**

Enter a new name: "Result". Use defaults for the other settings.

The screen should now look like this:



Create the action list

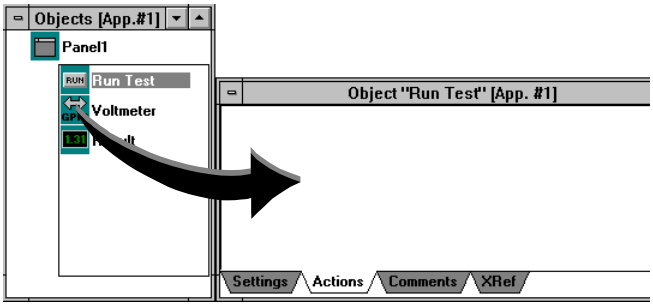
The action list window should be displaying the list for the "Run Test" pushbutton object, which is currently a blank list.

> **The action list for any object may be seen by clicking on the object in the object window with the right mouse button.**

An action list is a sequence of actions to be executed when an object receives an event. In the case of the pushbutton object, the activating event is the pushing of the button by the user (or by an action in another action list). For our example application, we need to carry out two steps when the user pushes the button: read the instrument, and display the result.

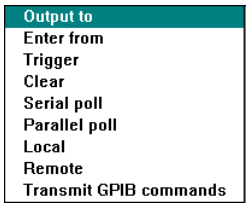
Read the voltmeter

Drag the Voltmeter object from the object window to the action list.



> **Action lines are created by dragging objects to the action list.**

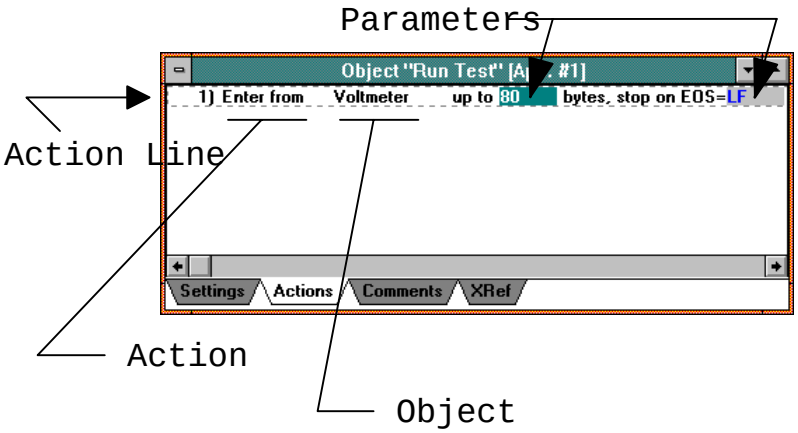
When the mouse button is released, you will see a popup list of action choices because the GPIB object has more than one action available.



Choose the "Enter from" action, which reads the device.

A text action line appears automatically, describing the action to be executed. Note that it has parameters which affect the action. In this case, the GPIB Enter From action, the parameters describe the maximum number of bytes to read, and the terminating EOS, or end-of-string, character.

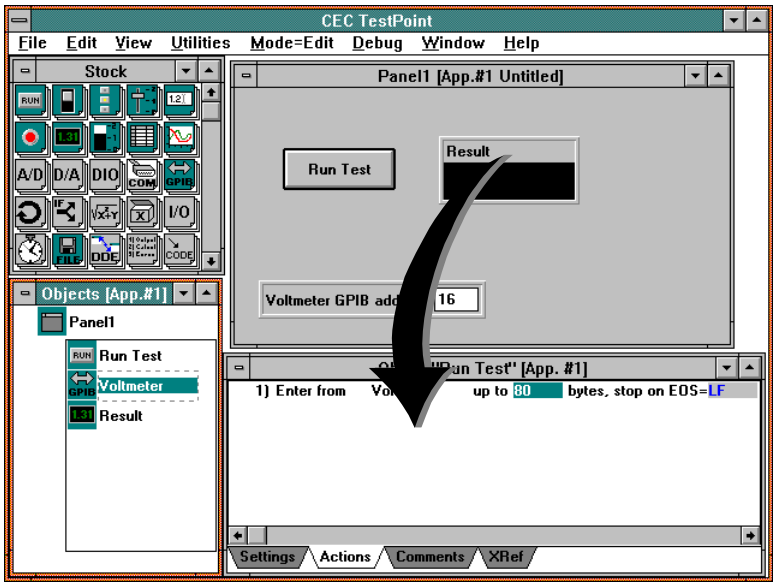
Click on the first parameter (256), and type in 80.



Constants may be entered by clicking on the desired parameter and typing on the keyboard.

Display the result

Drag the display object from the panel to the action list.



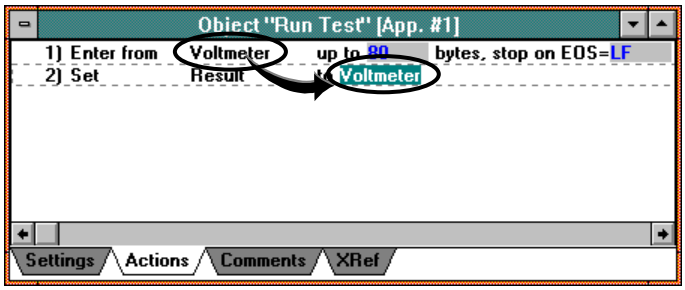
Note that you can drag from either the panel or the object window to create actions - there's no difference.

The new line has a blank parameter for the value to be put in the display. In this example, we want to display the value entered from the voltmeter. This value is stored as the data value of the voltmeter object.

Data values of objects may changed as the result of executing actions.

So, to set the display to the voltmeter value, we just drag the voltmeter object to the blank, shaded parameter area on action line 2.

Drag the Voltmeter object's name from line 1 to the blank on line 2.



Note that you can drag a reference to the voltmeter's data from the icon in the object window or the object name in a previous action line.

Objects can act like variables - they contain data values. You can use object data values in actions by dragging the object to a blank in an action line.

Run it

Use the Mode menu command to switch to Run mode.

Run mode changes the behavior of the panel. In Edit mode, using the mouse on panel objects selects them for moving, resizing, etc. In Run mode, the mouse acts as it will in the final application - it activates the object with an event.

Push the "Run Test" button.

A value will appear in the numeric display. Because demo mode was selected in the Voltmeter object settings, no actual GPIB read operation occurs. Instead, a random value is generated. You may push the button repeatedly to get new values.

Use the Mode menu command to switch back to Edit mode.

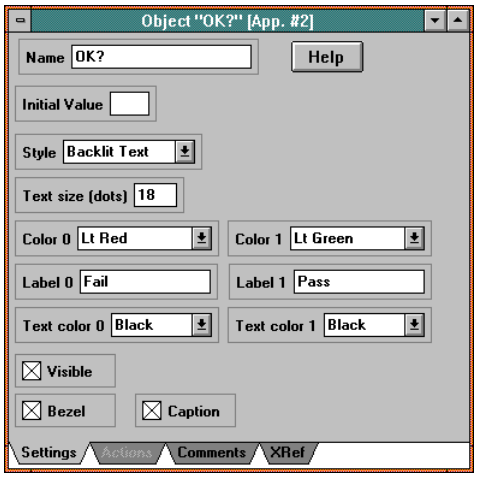
This example is also provided, already built for you, in the file **TUTORIAL\TEST1.TST**

Add Pass/Fail Range Checking

Once a measurement has been made, a common test requirement is checking the result against an allowed range and indicating a pass or fail. For this task, we need an indicator on the panel, and a way of calculating whether the value is in range.

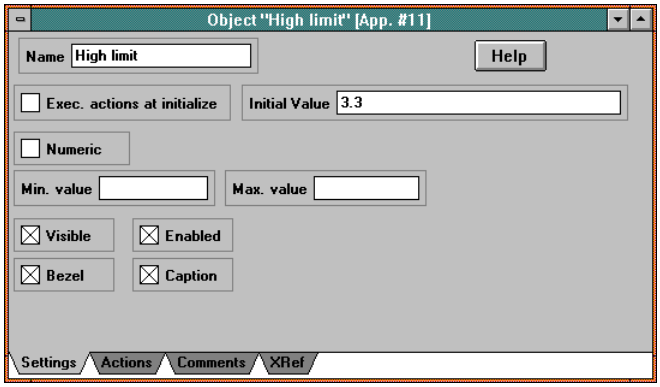
Indicator

Drag an indicator  from stock to the panel.
Name it "OK?", and select "Backlit text" as a style.



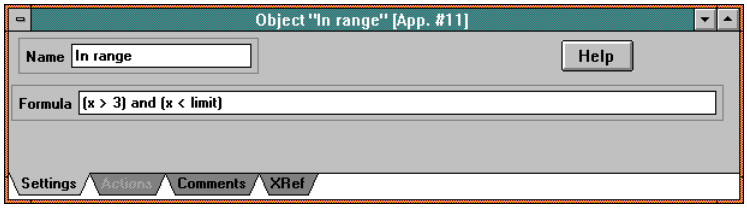
Data-Entry

Drag a data-entry object  from stock to the panel.
Name it "High limit",
and set its initial value setting to 3.3.



Math

Drag a math object  from stock to the object window.
Name it "In range", and enter:
(x > 3) and (x < limit)
as the formula.



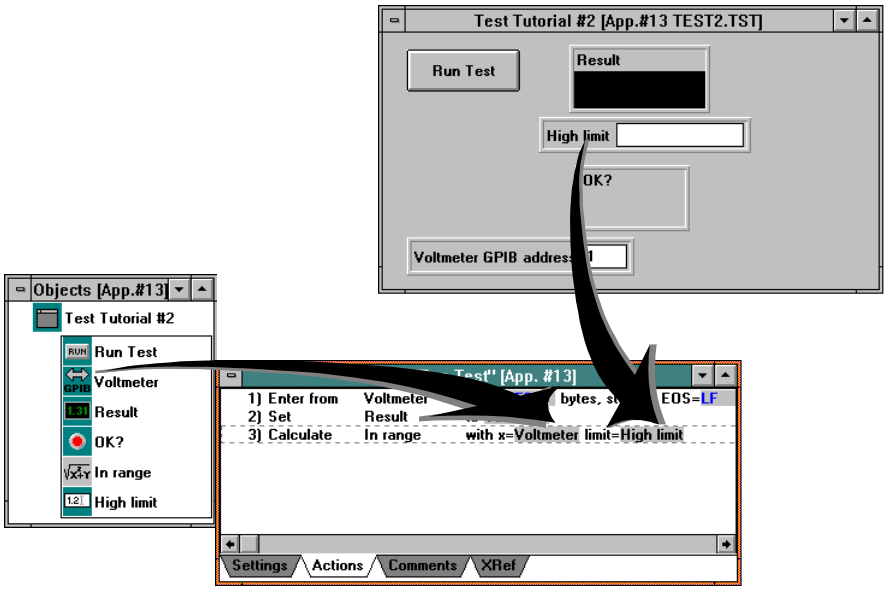
A math object can calculate any desired function of any number of variables, using a wide variety of built-in operations. In this example, we just want to compare a single value against a min and max constant range, and get a logical (true/false) result.

Now add the new steps to the action list for the test.

Click on the "Run Test" button with the right-hand mouse button, to view its action list.
Drag the "In range" math object to the list.
Choose the "Calculate" action.

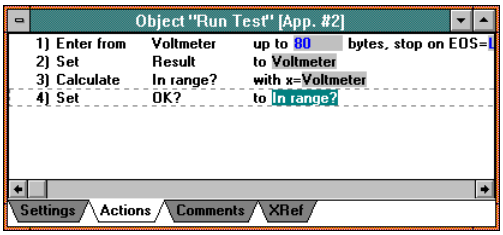
Note that the action line, created automatically for you, has parameter blanks for the values of "x" and "limit" in the formula.

Drag the Voltmeter object from the object window to the "x=" blank in the action list.
Drag the "High Limit" data-entry object from the panel to the "limit=" blank in the action list.



This new action line will calculate whether the data value of the Voltmeter object (which is the value read in from the instrument) is within range. The result of the calculation becomes the data value for the "In range" object.

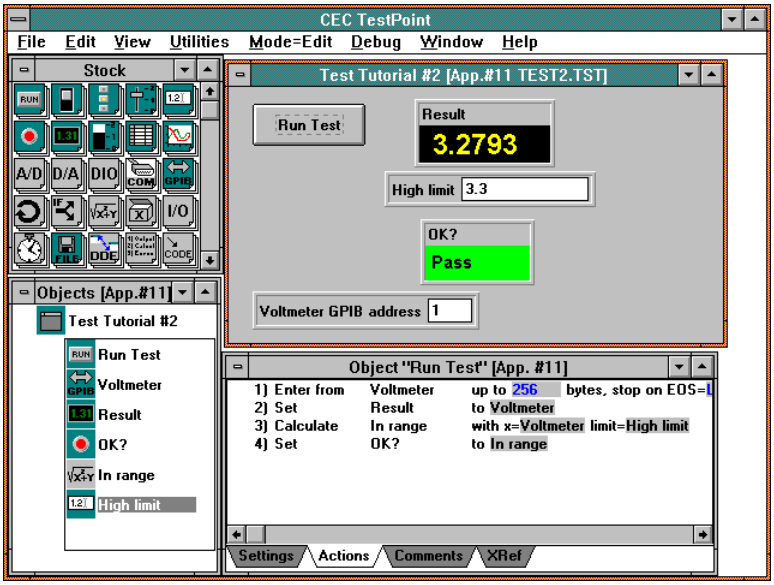
Drag the "OK?" indicator object to the action list.
Choose the "Set" action.
Drag the "In range" math object to the blank on the new action line.



Try running the application. Use the Mode menu command to select Run mode. Push the "Run Test" button. Try it multiple times. Depending on the value, the indicator will show Pass or Fail.

Try changing the limit value by typing into the data-entry field, and then pushing the button again.

Set the mode back to Edit.



This example is also provided, already built for you, in the file **TUTORIAL\TEST2.TST**

Add Another Control and Action List

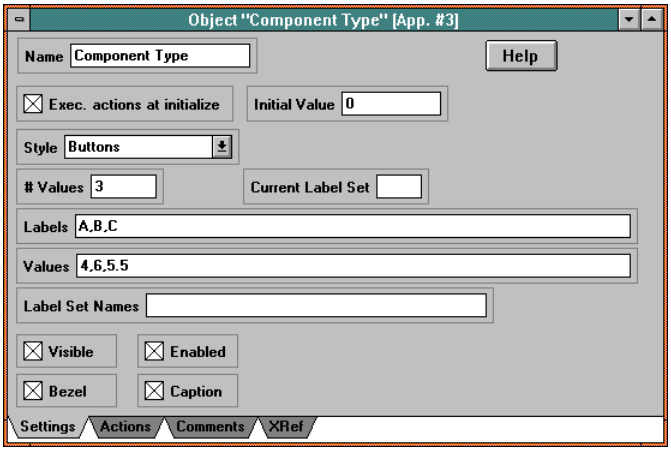
So far, the application consists of a single action list, which runs in sequence. Let's add a control for selecting the type of component being tested and also add a second device: a voltage source which provides an input to the unit under test.

Selector

Drag a selector object from the stock to the panel.

A selector object lets the user choose among a set of alternatives. The settings for the selector include the number of choices, the labels, and the values that correspond to those labels.

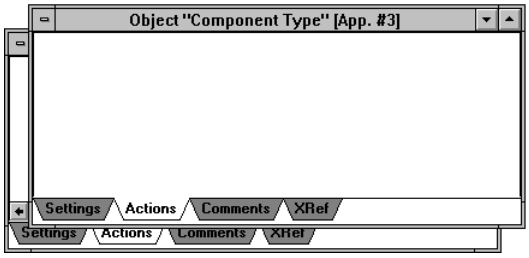
Enter "Component type" for the object name.
Enter 3 for number of values.
Enter "A,B,C" for the labels.
Enter "4,6,5.5" for the values.
Check the "Exec. actions at initialize" option.
Enter "A" for the initial value.



Drag another GPIB object from the stock to the panel, and name it "Source".
Set its "Demo mode" setting on.

Now, we want to set the voltage source's output whenever the component type is changed. Changing the component type selector causes its action list to be executed.

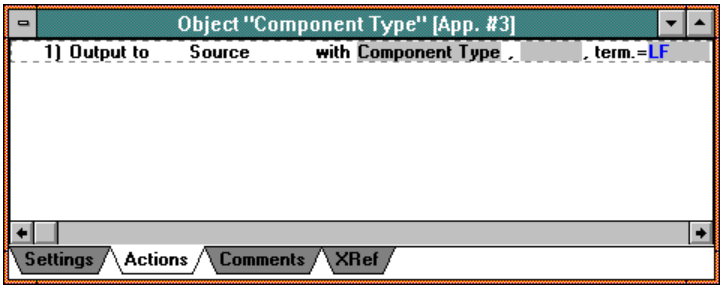
Click on the "Component type" selector with the right mouse button to view its action list.



There can be many separate action lists in an application. Each is executed when its associated object receives an event.

Note that this action list is empty. The action list for the "Run Test" pushbutton and the action list for the "Component Test" selector are independent.

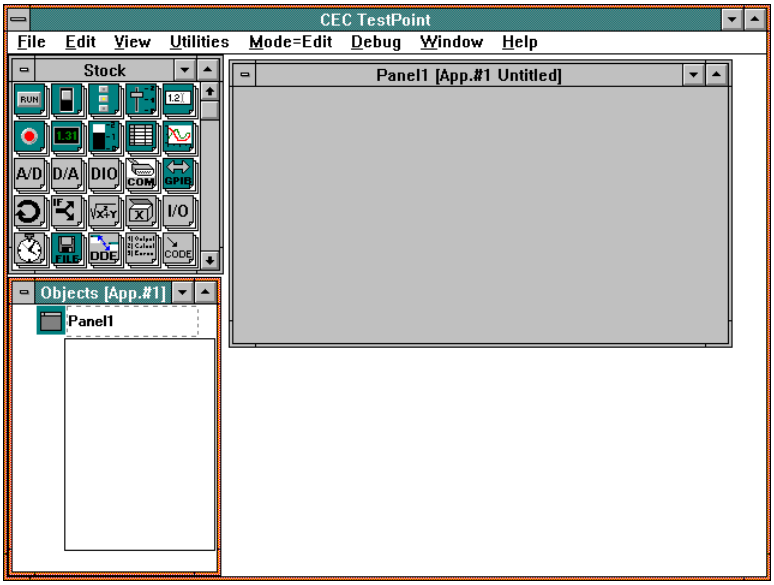
Drag the "Source" object to the action list.
Choose the "Output to" action.
Drag the "Component Type" selector to the first shaded parameter blank on the new action line.



The value of the "Component Type" selector is 4, 6, or 5.5 - the desired voltage, which is sent out to the voltage source device.

This example is also provided, already built for you, in the file **TUTORIAL\TEST3.TST**

When you start the TestPoint development environment, the editor window will appear, with a new, blank panel for you to fill in:



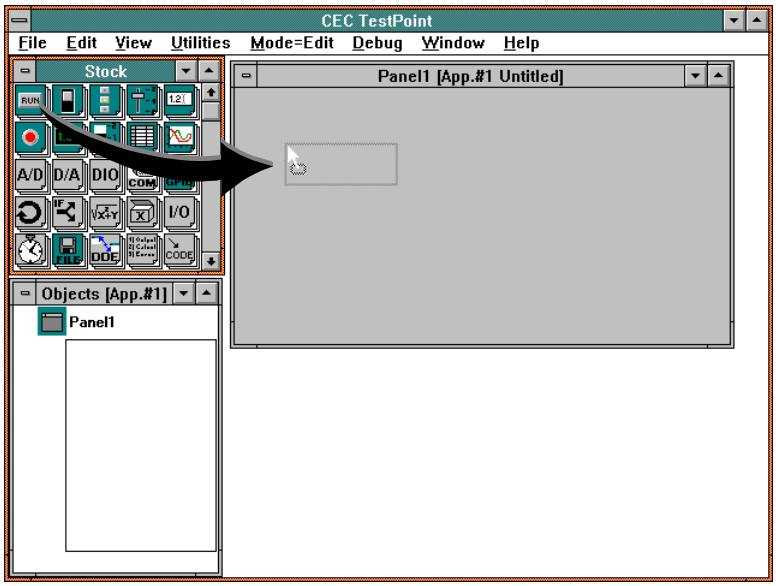
There are four areas in the TestPoint editor: the panel, the stock window, the object list window, and individual object windows which appear as you add objects. The use of these windows will be explained in more detail in the sections that follow. First, though, we'll run once through the steps of creating a new application.

The first application we create will take some analog samples from an A/D board and graph them, when a button is pushed.

Add a pushbutton

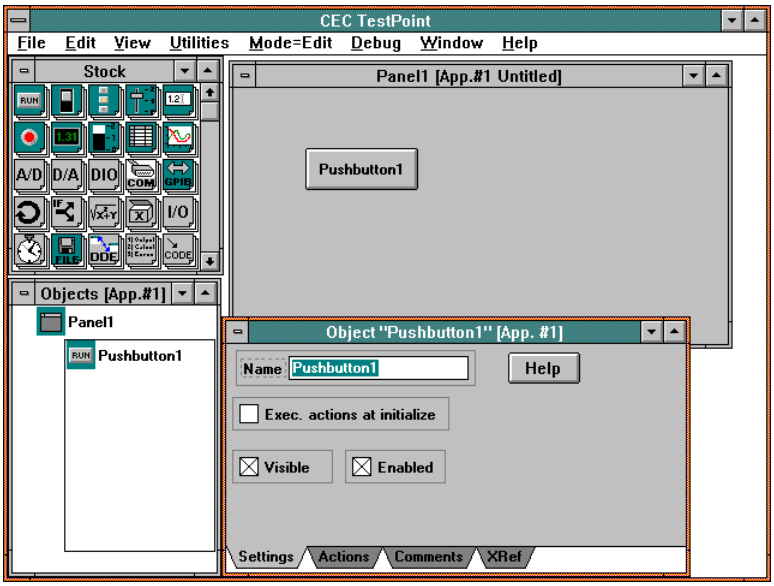
Drag a pushbutton  **from the stock window to the panel.**

(dragging means to place the mouse pointer over the source, push the left mouse button down and hold it, then move the mouse to the destination and release the button).



New objects are added to TestPoint applications by dragging and dropping from the stock.

When you release the mouse button, the pushbutton object will appear on the panel. A window titled "Object Pushbutton1" will also appear, to let you enter information about the new object.



The name of the new object starts out highlighted and ready to be replaced.

Type in a new name for the object: "Acquire".

The default values for the other settings are OK. You can click on any other window or on the "Actions" tab to dismiss the settings window.

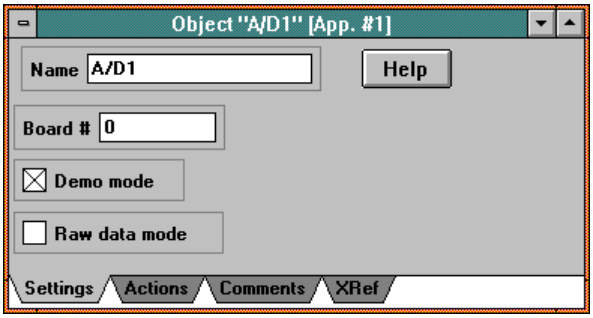
You can double-click the object or its icon in the object window to re-open its settings window.

Note that the new object appears not only on the panel, but also in the object window, and that its action list appears automatically.

Add the A/D board

Drag an A/D object  **from the stock to the panel.**

In the settings window, click the checkbox for "demo mode".



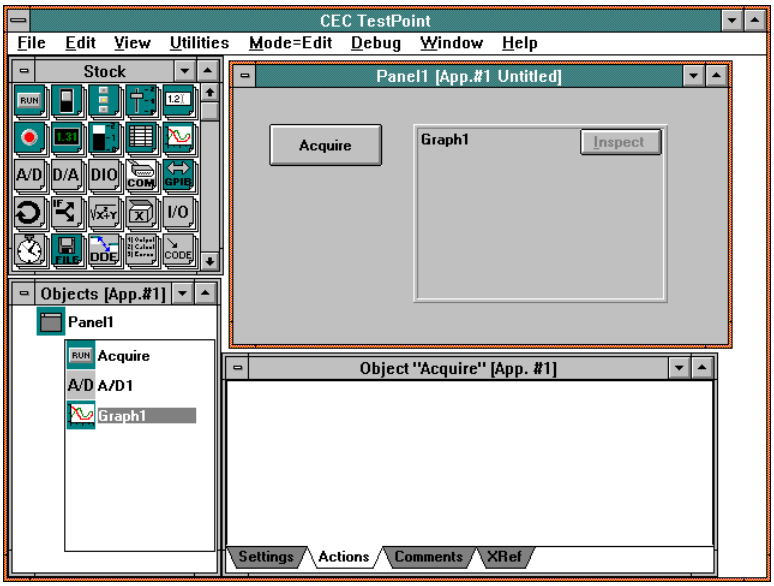
Demo mode is available as a setting on objects which access hardware I/O devices. It allows you to run your application without the hardware actually being present, for testing purposes.

Add a graph

Drag a Graph object  from the stock to the panel.

Use defaults for the settings.

The screen should now look like this:



Create the action list

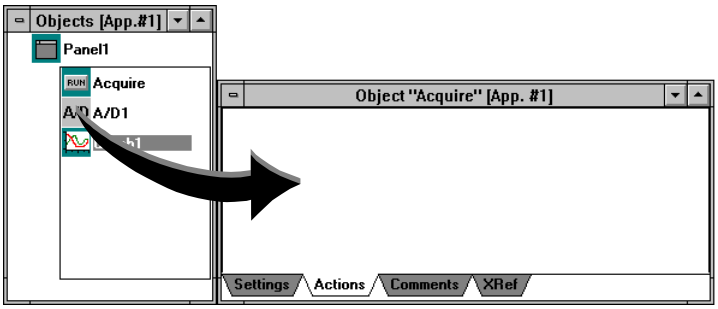
The action list window should be displaying the list for the "Acquire" pushbutton object, which is currently a blank list.

> **The action list for any object may be seen by clicking on the object in the object window with the right mouse button.**

An action list is a sequence of actions to be executed when an object receives an event. In the case of the pushbutton object, the activating event is the pushing of the button by the user (or by an action in another action list). For our example application, we need to carry out two steps when the user pushes the button: acquire the data, and graph it.

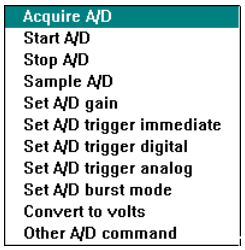
Acquire the data

Drag the A/D object from the object window to the action list.



> **Action lines are created by dragging objects to the action list.**

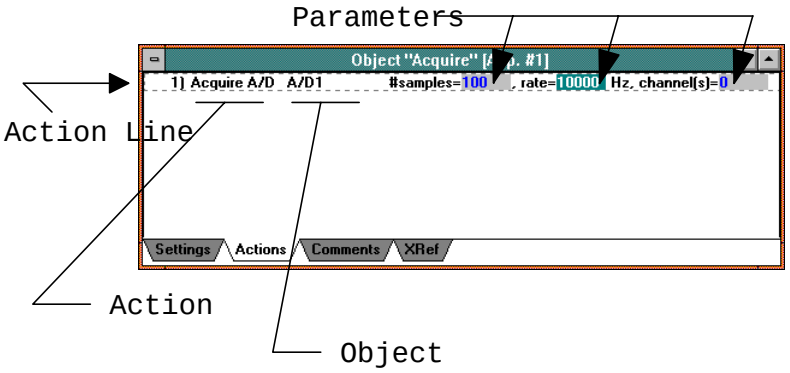
When the mouse button is released, you will see a popup list of action choices because the A/D object has more than one action available.



Choose the "Acquire A/D" action, which reads voltage samples.

A text action line appears automatically, describing the action to be executed. In this case, the Acquire A/D action has parameters that describe the number of samples desired, the rate at which the samples are to be taken, and the input channels to be sampled.

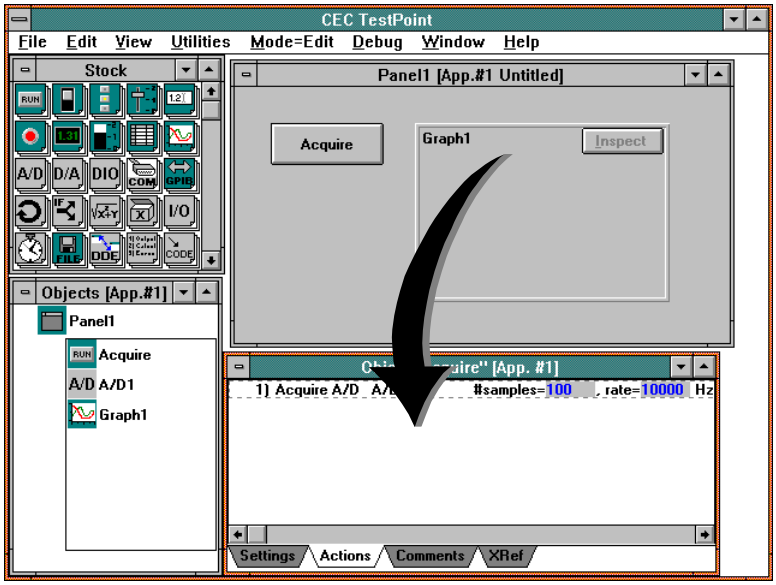
Click on the first blank (after #samples=).
Type in 100 from the keyboard.
Use the TAB key to get to the next parameter,
and enter 10000 for the rate.



Constants may be entered by clicking on the desired parameter and typing on the keyboard.

Graph the data

Drag the graph object from the panel to the action list. Choose the "Draw graph" action.



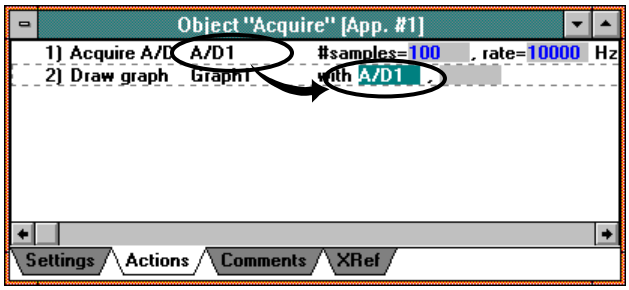
Note that you can drag from either the panel or the object window to create actions - there's no difference.

The new line has a blank parameter for the data to be graphed. In this example, we want to graph the data we just acquired from the A/D board. This data is stored as the data value of the A/D object.

Data values of objects may changed as the result of executing actions.

So, to graph the data, we just drag the A/D object to the blank, shaded parameter area on action line 2.

Drag the A/D object's name from line 1 to the blank on line 2.



A second blank appears on the action line because the Draw graph action is able to accept multiple data items to be graphed in a single action step. For this example, only one vector of data samples is being graphed.

Note that you can drag a reference to the object's data from the icon in the object window or the object name in a previous action line.

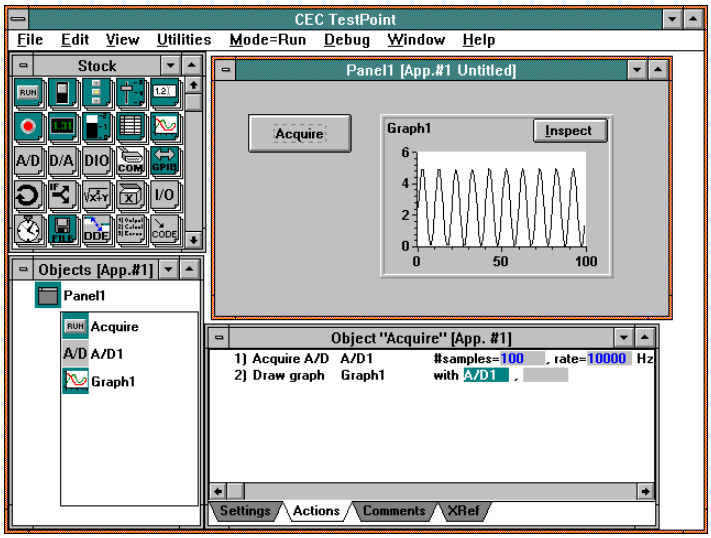
Objects can act like variables - they contain data values. You can use object data values in actions by dragging the object to a blank in an action line.

Run it

Use the Mode menu command to switch to Run mode.

Run mode changes the behavior of the panel. In Edit mode, using the mouse on panel objects selects them for moving, resizing, etc. In Run mode, the mouse acts as it will in the final application - it activates the object with an event.

Push the "Acquire" button.



Data will appear in the graph. Because demo mode was selected in the A/D object settings, no actual data sampling operation occurs. Instead, simulated data is generated.

Use the Mode menu command to switch back to Edit mode.

This example is also provided, already built for you, in the file **TUTORIAL\DATAACQ1.TST**

Change to Strip Chart

The first example we've built acquires the data and then graphs it. That's OK when the number of samples is small and the data rate high, but many data acquisition applications require slower sampling over a long period of time, with continuous monitoring of the results. For that purpose, we need to use background sampling and a strip chart.

Set the graph to strip chart

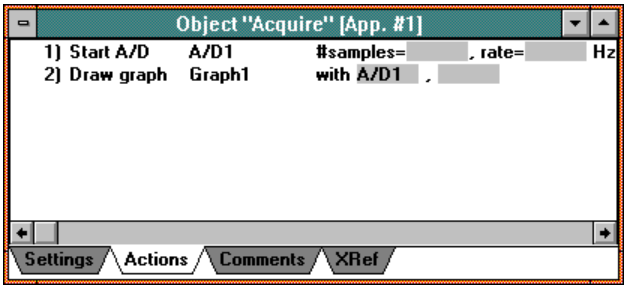
Double-click the graph object icon in the Objects window, to open its settings.
Change the "Mode" setting to "Strip Chart".

Use background sampling

Click on the words "Acquire A/D" in line 1 of the action list.
Use the Del key to delete this action line.

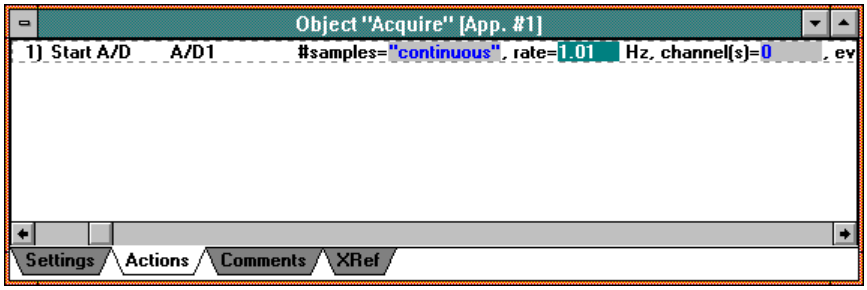
You can select action lines by clicking on the action words at the beginning of the lines.
After selection, action lines may be copied, cut, pasted, or deleted.

Drag the A/D object in again, and drop it above line 1.
Choose the "Start A/D" action.



Objects can be dropped anywhere in an action list to insert lines at a desired location.

Enter "continuous" for # samples, 1.01 for the rate, and leave the other parameters at their default values.

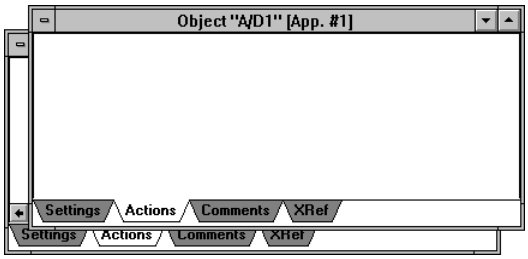


(Note: we use 1.01 Hz because the simulated demo data source is a 1000 Hz sine wave, and we don't want to sample it at exactly the same point in every cycle and get a flat line.)

The "Start A/D" action begins sampling, but does not wait until data arrives to continue executing the action list. So, we can no longer graph the data in line 2.

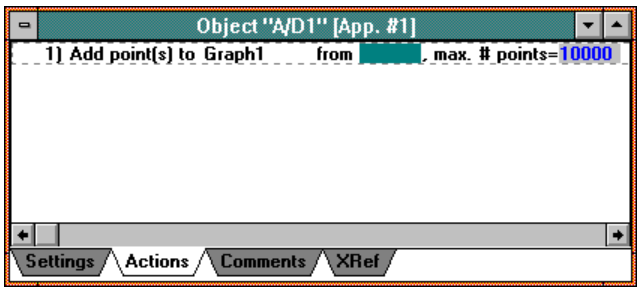
Graph in A/D action list

Click on the words "Draw graph" in line 2.
Use the Del key to delete this line.
Click on the A/D object icon with the right mouse button, to view its action list (which should be blank).



There can be many separate action lists in an application.
Each is executed when its associated object receives an event.

Drag the Graph object to the A/D action list.
Choose the "Add point(s) to" action.



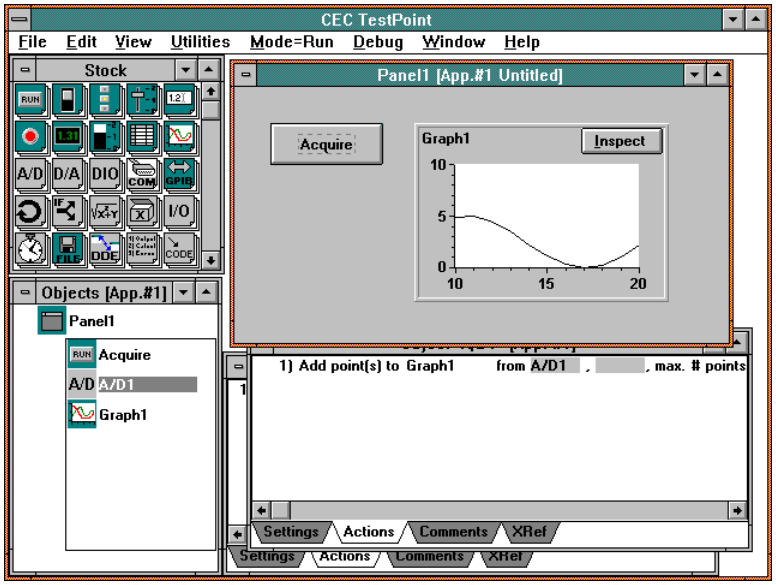
Drag the A/D object icon to the blank in line 1.

The A/D object runs its action list whenever samples taken by the "Start A/D" action arrive. Each time a sample arrives, we want to add that point to the graph.

You should now have two action lists: the button, which starts the A/D sampling, and the A/D's list, which adds points to the graph as they arrive.

Run the application

Switch to "Mode=Run" with the Mode menu command.
Push the "Acquire" button.



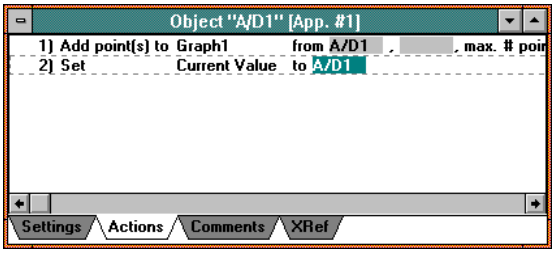
Watch as samples arrive, and the graph begins to scroll.

If you wish, you can use the "Inspect" button on the graph to make a copy of the strip chart data and review the data which has scrolled off the left side. Note that the graph itself continues to update while you review historical data.

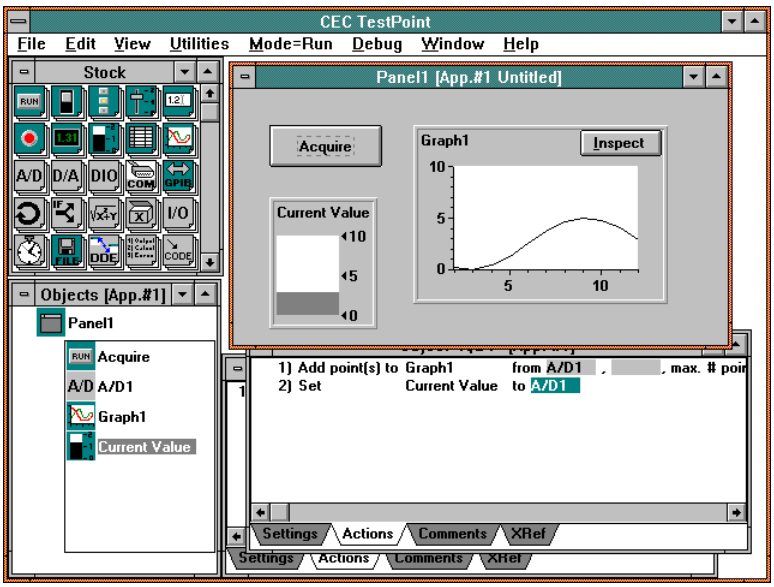
This example is also provided, already built for you, in the file **TUTORIAL\DATAACQ2.TST**

If you want a bar graph indicator

Drag a Bar  **object from the stock to the panel.**
Name it "Current value".
Drag it to the A/D action list.
Drag the A/D object to the new parameter blank.



Run it.



Use the Mode menu command to switch back to Edit mode.

Add Analog Output (D/A)

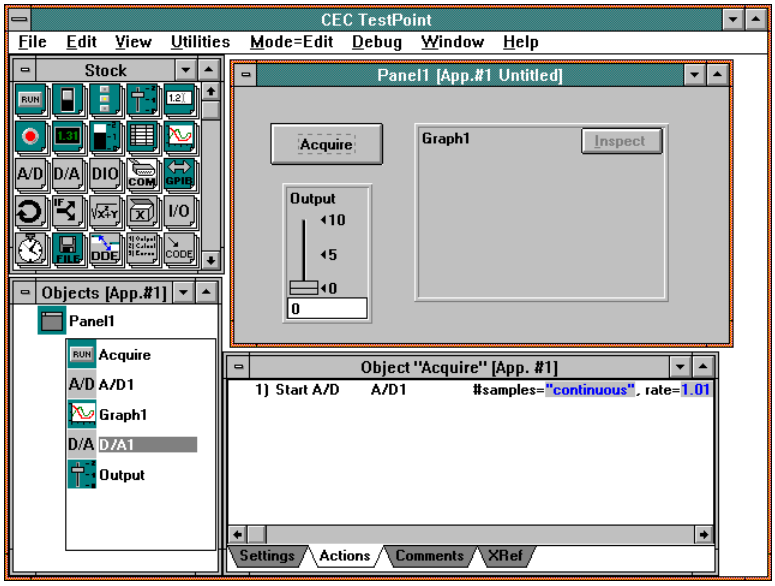
Next, let's add an analog output channel, controlled by a slider on the panel.

Add a D/A object

Drag a D/A object  from the stock to the Objects window. Set "Demo mode" on.

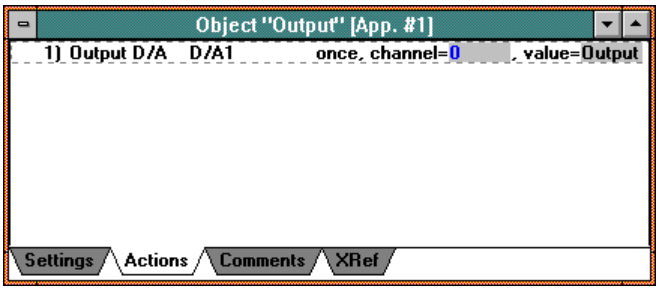
Add a slider control

Drag a Slider object from the stock to the panel. Name it "Output".



Build the action list

Click on the "Output" slider's icon with the right mouse button, to view its action list.
Drag the D/A object to the action list.
Choose the "Output D/A" action.
Drag the slider object to the "value" parameter.



The slider's action list executes whenever the slider value is changed, and sets the D/A output value to the slider.

Try running the application.
Push "Acquire".
Move the "Output" slider by dragging it.
Switch back to Edit mode.

Multitasking: you can use the "Acquire" button to start the strip chart, and use the "Output" slider to change the D/A output while the strip chart continues to run.

This example is also provided, already built for you, in the file **TUTORIAL\DATAACQ3.TST**

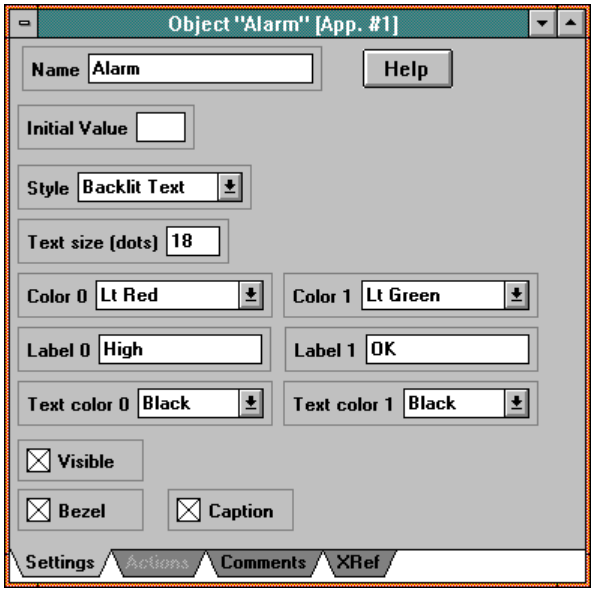
Add alarm limits

A common data acquisition application is the indication of alarms when a sampled value goes out of an allowed range.

To check the data value and display an alarm, we need a Math object and an Indicator object.

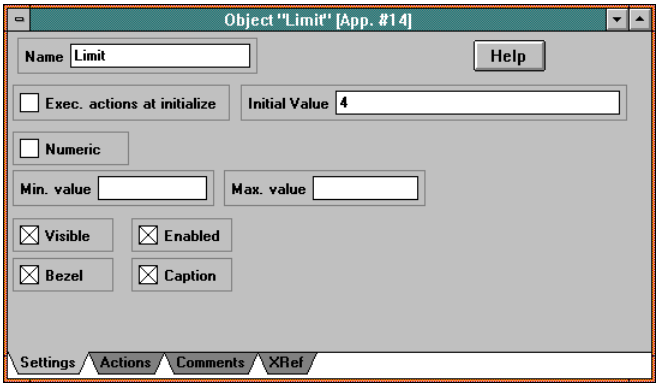
Add an indicator

Drag an indicator object from stock to the panel.
Name it "Alarm".
Set its style to "Backlit Text",
and Label 0 to "High", Label 1 to "OK".



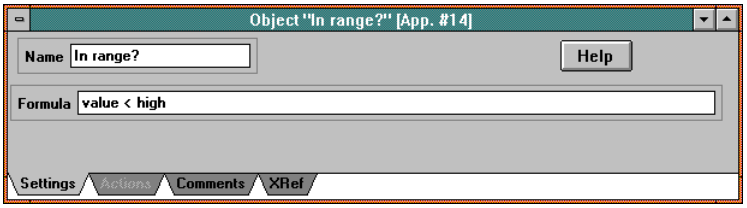
Add Data-Entry

Drag a data-entry object  from stock to the panel.
Name it "Limit",
and set its initial value setting to 4.



Add Math

Drag a math  object from stock to the Objects window.
Name it "In range?"
Set its formula to:
value < high



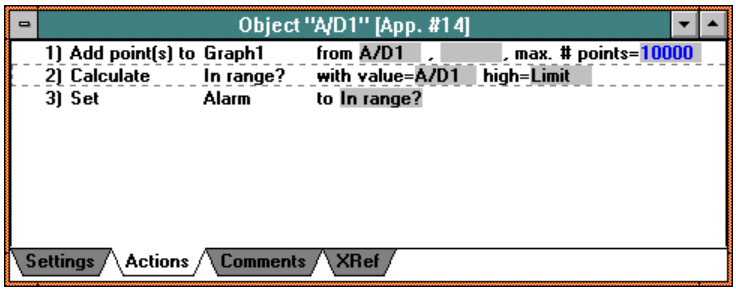
A math object can calculate any desired function of any number of variables, using a wide variety of built-in operations. In this example, we just want to compare a single value against a max range, and get a logical (true/false) result.

Add actions

Click on the A/D object icon with the right mouse button to view its action list.

Drag the "In range?" object to the action list.
Choose the "Calculate" action.
Drag the A/D object to the "value=" parameter, and the "Limit" object to the "high=" parameter.

Drag the "Alarm" object to the action list.
Choose the "Set" action.
Drag the "In range?" object to the parameter.



Whenever a sample arrives, the A/D action list executes. In addition to adding the point to the strip chart, the action list now checks if the value is less than the limit and sets the alarm indicator appropriately.

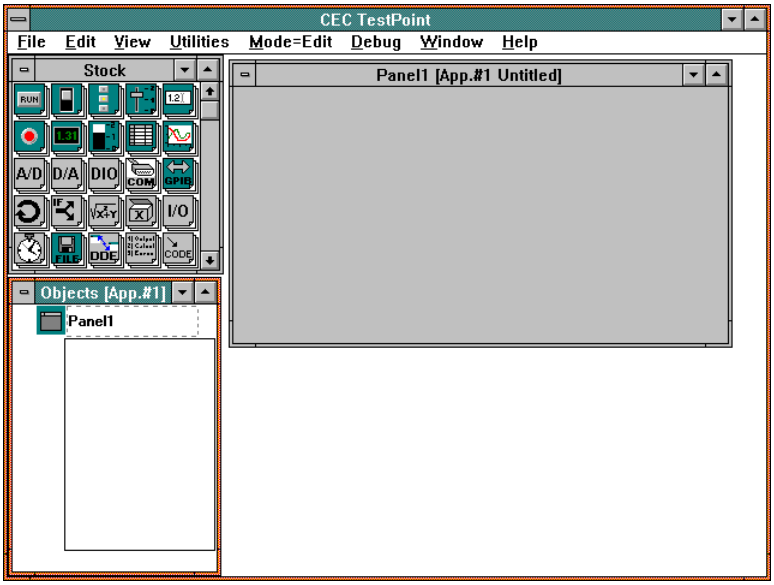
Try running the application.
Push "Acquire".

Watch as the alarm turns on and off as the input value changes. Try changing the limit value by typing into the data-entry field.

Switch back to Edit mode.

This example is also provided, already built for you, in the file **TUTORIAL\DATACQ4.TST**

When you start the TestPoint development environment, the editor window will appear, with a new, blank panel for you to fill in:



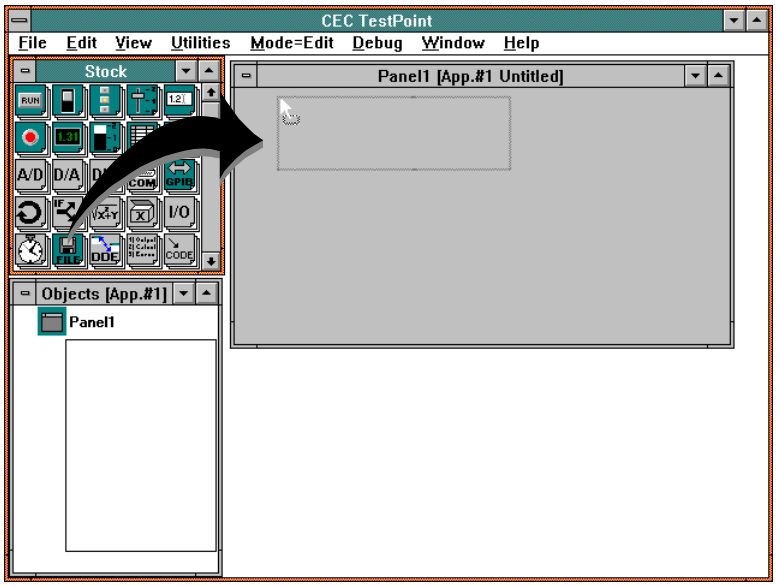
There are four areas in the TestPoint editor: the panel, the stock window, the object list window, and individual object windows which appear as you add objects. The use of these windows will be explained in more detail in the sections that follow. First, though, we'll run once through the steps of creating a new application.

The first application we create will read a data file containing numbers and graph the data in it.

Add a File object

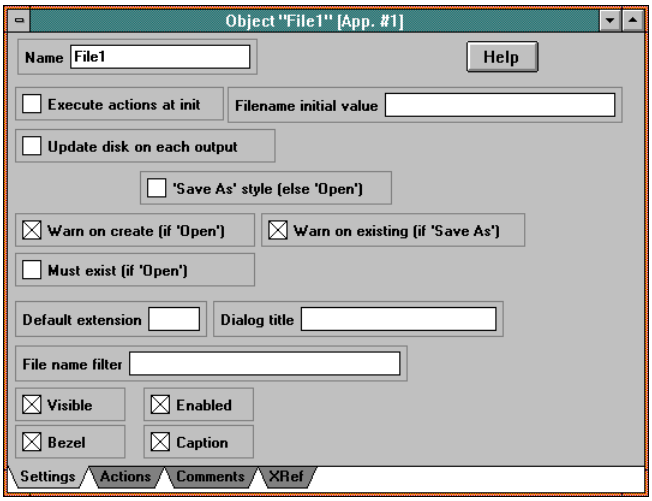
Drag a File object  from the stock window to the panel.

(dragging means to place the mouse pointer over the source, push the left mouse button down and hold it, then move the mouse to the destination and release the button).



New objects are added to TestPoint applications by dragging and dropping from the stock.

When you release the mouse button, the pushbutton object will appear on the panel. A window titled "Object File1" will also appear, to let you enter information about the new object.



The name of the new object starts out highlighted and ready to be replaced.

Type in a new name for the object: "Data file".
Enter this string into the File name filter setting:
Data files|*.DAT|
(using the vertical bar character as a separator).

The default values for the other settings are OK for now. You can click on any other window or on the "Actions" tab to dismiss the settings window.

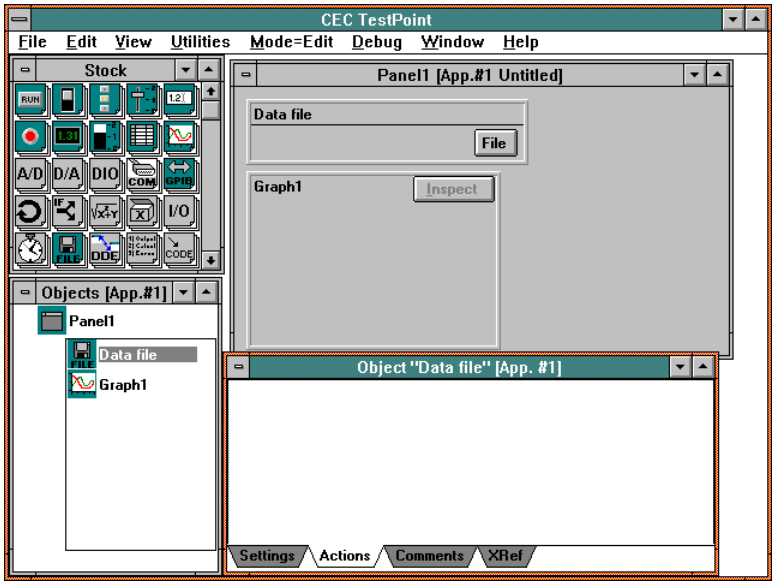
You can double-click the object or its icon in the object window to re-open its settings window.

Note that the new object appears not only on the panel, but also in the object window, and that its action list appears automatically.

Add the graph

Drag a Graph object from the stock to the panel. Use its default settings.

The screen should now look like this:



Create the action list

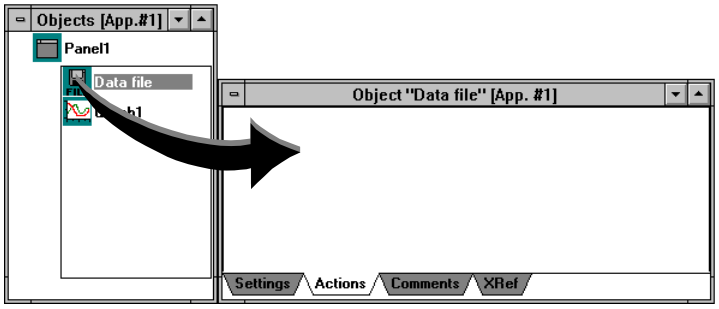
The action list window should be displaying the list for the "Data file" file object, which is currently a blank list.

> **The action list for any object may be seen by clicking on the object in the object window with the right mouse button.**

An action list is a sequence of actions to be executed when an object receives an event. In the case of the File object, the activating event is the selection of a new file by the user (or by an action in another action list). For our example application, we need to carry out two steps when the user pushes the button: read the file, and graph the data.

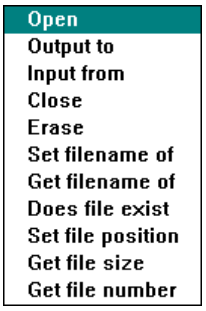
Read the file

Drag the "Data file" object from the object window to the action list.



> **Action lines are created by dragging objects to the action list.**

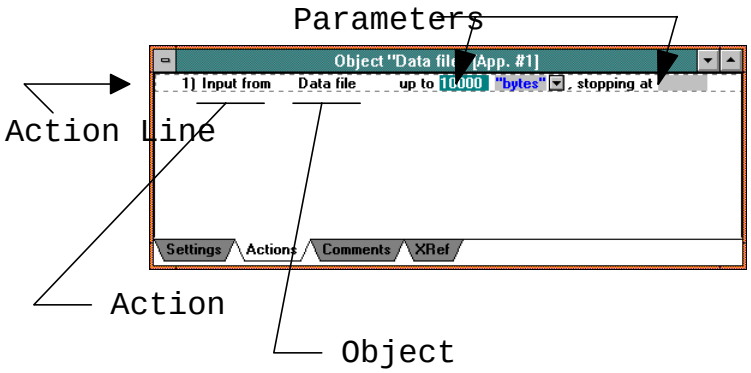
When the mouse button is released, you will see a popup list of action choices because the File object has more than one action available.



Choose the "Input from" action, which reads data from the file.

A text action line appears automatically, describing the action to be executed. Note that it has parameters which affect the action. In this case, the file "Input from" action, the parameters describe the maximum number of bytes to read, and an optional terminating character to stop the input.

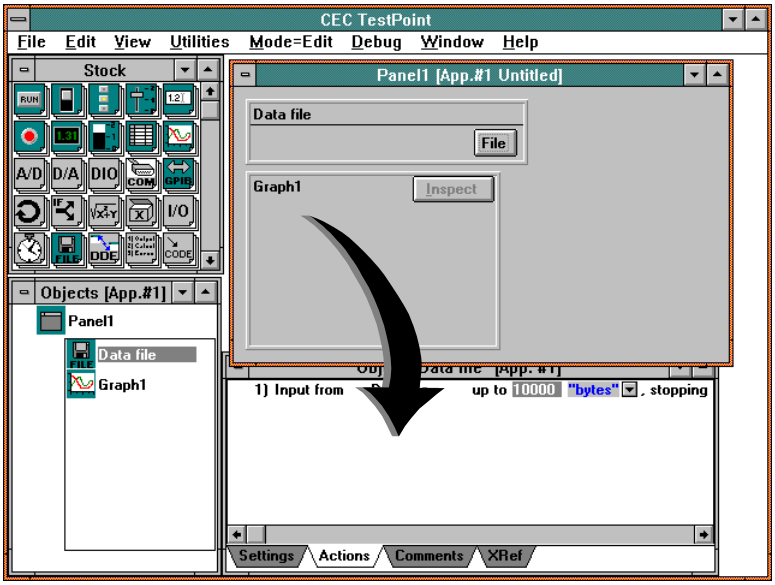
Click on the first parameter (32768), and type in 10000.



Constants may be entered by clicking on the desired parameter and typing on the keyboard.

Graph the data

Drag the graph object from the panel to the action list. Choose the "Draw graph" action.



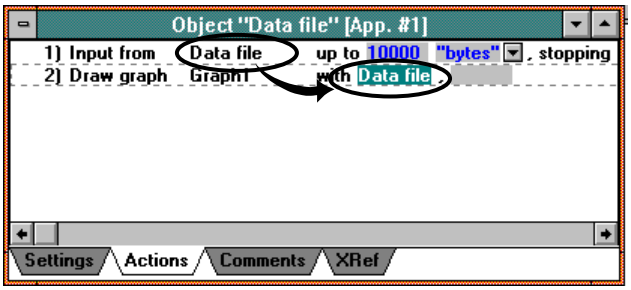
Note that you can drag from either the panel or the object window to create actions - there's no difference.

The new line has a blank parameter for the data to be graphed. In this example, we want to graph the data we just read from the file. This data is stored as the data value of the file object.

Data values of objects may changed as the result of executing actions.

So, to graph the data, we just drag the file object to the blank, shaded parameter area on action line 2.

Drag the File object's name from line 1 to the blank on line 2.



A second blank appears on the action line because the Draw graph action is able to accept multiple data items to be graphed in a single action step. For this example, only one data item is being graphed.

Note that you can drag a reference to the file's data from the icon in the object window or the object name in a previous action line.

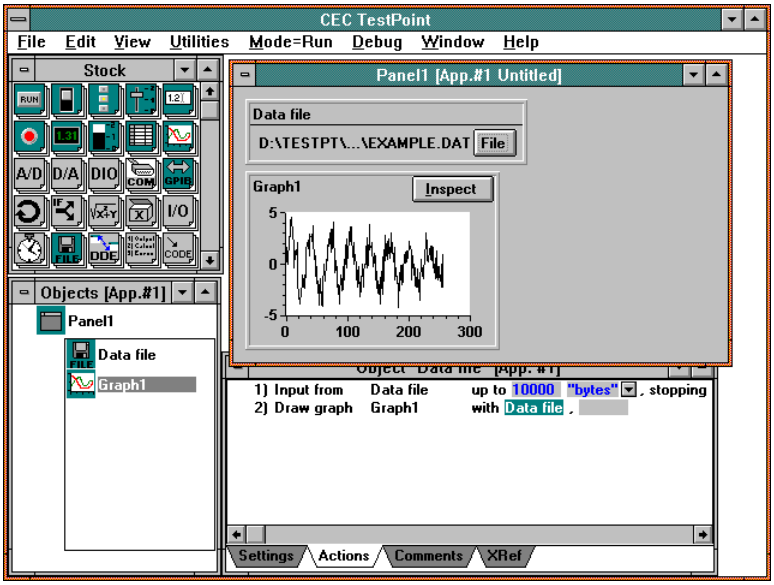
Objects can act like variables - they contain data values. You can use object data values in actions by dragging the object to a blank in an action line.

Run it

Use the Mode menu command to switch to Run mode.

Run mode changes the behavior of the panel. In Edit mode, using the mouse on panel objects selects them for moving, resizing, etc. In Run mode, the mouse acts as it will in the final application - it activates the object with an event.

Push the "File" button in the "Data file" object.
Choose the "example.dat" file in the tutorial directory.



Use the Mode menu command to switch back to Edit mode.

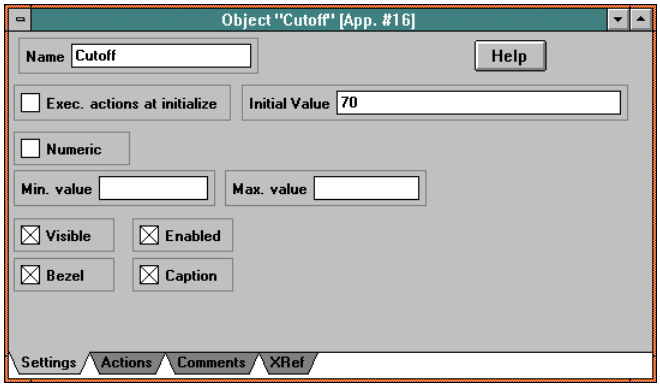
This example is also provided, already built for you, in the file **TUTORIAL\ANALYZE1.TST**

Filter the data

The example data in the file is typical of real-world acquired data. It contains noise which obscures the information. TestPoint includes digital filtering functions for this situation.

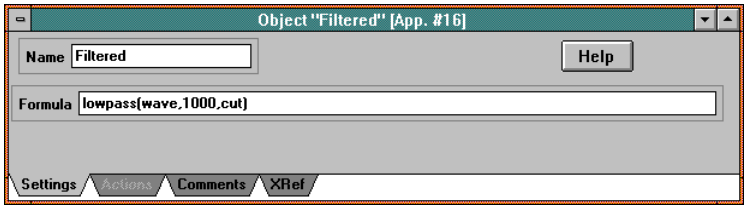
Add a data-entry field

Drag a Data-Entry object from stock to the panel.
Name it "Cutoff",
and set its initial value setting to 70.



Add a Math object

Drag a Math object from stock to the Objects window.
Set its name to "Filtered",
and its formula to:
lowpass(wave,1000,cut)

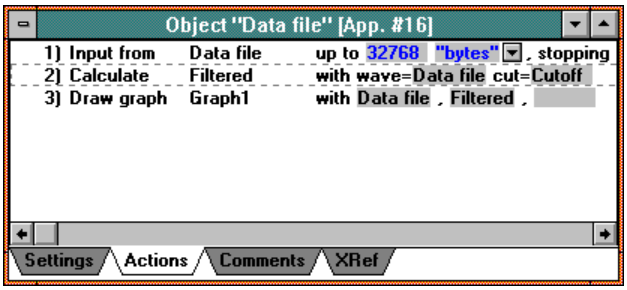


The **lowpass** filter function is one of the many math functions included in TestPoint for data analysis. The parameters give the

waveform to be filtered, the sampling frequency, and the cutoff frequency.

Modify the action list

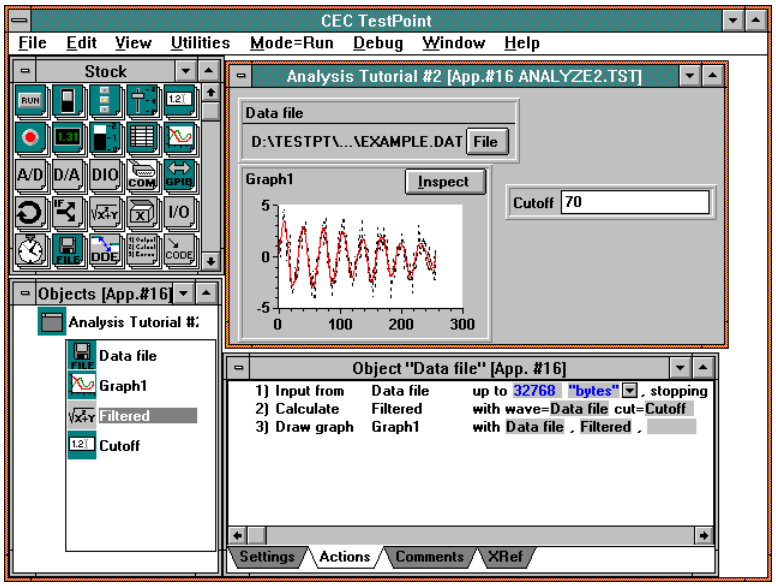
Drag the Math object to the action list, and drop it between line 1 and line 2.
Choose the "Calculate" action.
Drag the File object to the "wave=" parameter, and the "Cutoff" data-entry object to the "cut=" parameter.
Drag the Math object to the second (blank) parameter in the "Draw graph action line.



Objects can be dropped anywhere in an action list to insert lines at a desired location.

Line 3 now provides two data items to the graph, which are drawn as two traces.

Run the application, and choose the "example.dat" file again.



If you want to see just the filtered data on the graph, click on the "Data file" parameter in line 3 and use the Del key to delete it, then select the file again.

Switch back to Edit mode.

This example is also provided, already built for you, in the file **TUTORIAL\ANALYZE2.TST**

Find the frequency

The data is expected to be an exponentially decaying sine wave. To determine its frequency, we can use a Fourier Transform and find the highest point in the frequency spectrum.

Add more math

Drag in another Math object.
Name it "Freq".
Set its formula to:

```
maxindex(select(FFT(1000,wave),1)) * 1000/dim(wave)
```

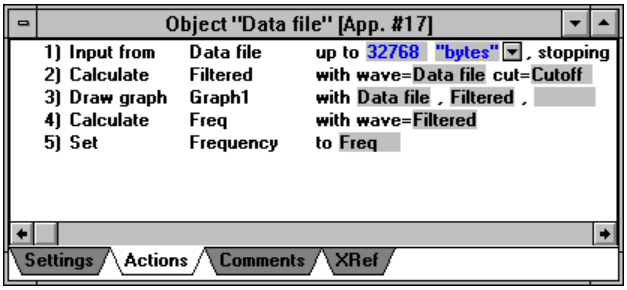
The **FFT** function does a Fast Fourier Transform, given the wave and sampling frequency, and returns a list of 3 items: the frequency components, the magnitudes, and the phases. The **select** function picks out the magnitude part, and the **maxindex** function finds the index of the largest value. Scaling this based on the 1000 Hz sampling frequency gives the waveform's base frequency.

Add a display

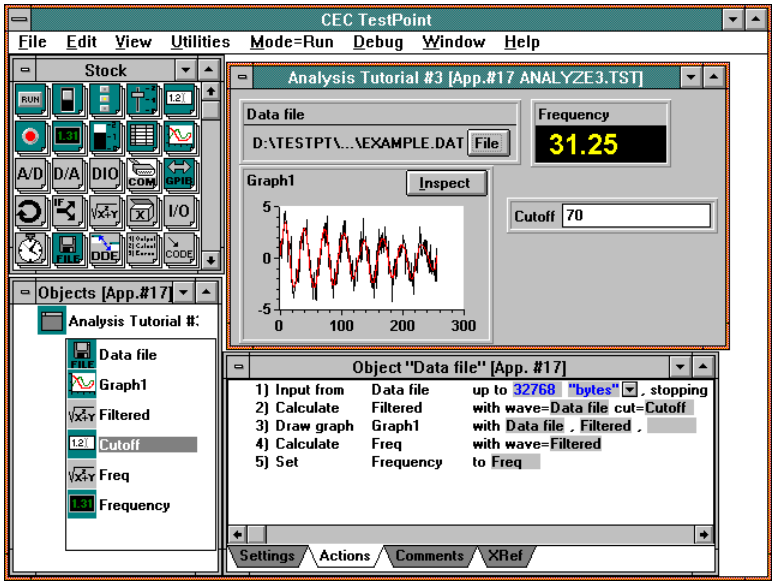
Drag a Display  **object from the stock to the panel.**
Name it "Frequency".

Drag the "Freq" math object to the action list.
Choose the "Calculate" action.
Drag the "Filtered" math object to the "wave" parameter.

Drag the "Frequency" display to the action list.
Drag the "Freq" object to this parameter.



Run the application again,
and select the "example.dat" file again.



Switch back to Edit mode.

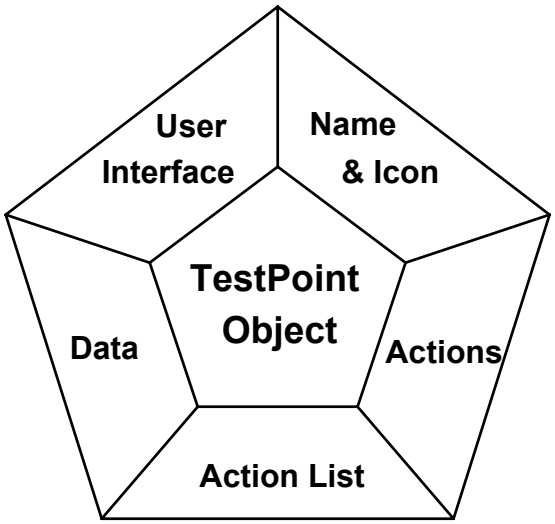
This example is also provided, already built for you, in the file **TUTORIAL\ANALYZE3.TST**

Review - TestPoint Concepts

Objects

An **object** in TestPoint represents a component of your application, such as a user interface item, an external measurement device, a disk data file, or a calculation. A single object incorporates, in one place, the actions and information (data) for such a component.

Objects have a number of characteristics:



Actions Every TestPoint object has one or more actions which it can execute. For example, display objects can do a "set to" action, which changes the value displayed. The file object has actions for writing to the file, reading the file, erasing the file, etc.

Data Every TestPoint object also has a data value. This data value may be changed as the result of an action of the object. For example, executing the "read sample" action of the analog/digital input object sets the object's data to the result of the sampling operation. Object data values may be used in math calculations and as parameters in actions for any other object.

Name & Icon

Every object has a name, which is a text label for your convenience as application author. In some cases, the name is also used as a caption for the object on the application panel. Objects also have small graphic icons which indicate the object category. For example, pushbutton objects all use this icon: 

.

User Interface

Some object categories also have a user interface. The pushbutton, switch, data entry, display, indicator, and graph objects are all examples of objects which have a user interface and appear on the panel. Some objects, like the pushbutton and switch, react to user input through the mouse or keyboard. User interface objects which accept input change their data values when the user interacts with them. For example, the data entry object takes on a new value when the user types into it and then hits Enter or Tab.

Action List

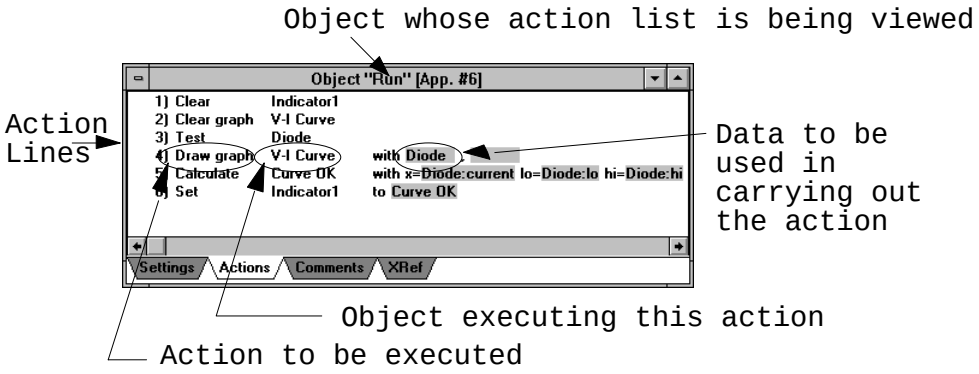
Many object categories have an action list associated with them. The action list specifies a sequence of actions to be carried out by the application's objects when the associated object gets an event. For example, the pushbutton object gets an event when the user clicks on it with the mouse or activates it from the keyboard. The file object gets an event when the user selects a new filename. Analog input objects can get events when sampled data becomes available.

"Child" objects

The panel object is special in that it can contain other objects as "children". A top-level panel always exists for each application, and is the main user interface for that application. Objects added to the application become child objects of the main panel. In addition, an application can contain panel objects, to allow multiple windows to be displayed. These panel objects can also have child objects.

Action Lists

Action lists in TestPoint are the means for specifying what the application does.



Action lists are built by choosing objects you want to interact with and dragging them to the action list window. TestPoint responds by creating a textual action line describing back to you the action you have chosen.

An action list is shown as a numbered series of actions. New actions may be added by dragging in objects. Actions may be deleted and their order may be rearranged. Action lists can also include repeating loops and conditionally executed sections (using the Loop and Conditional objects).

Data

Each TestPoint object has a data value, which can be modified as the result of its actions. These data values, in turn, can be used as parameters in another action.

For example, in this action list:

```
1) Enter from    Voltmeter    up to 256 bytes, stop on EOS=LF
2) Set           Result       to Voltmeter
```

line 2 sets the value of object "Result" to the current value of object "Voltmeter" (which was modified in the action in line 1).

Actions which modify object data generally replace the old data value with a new value. For example, in this action list:

```
1) Enter from    Voltmeter    up to 256 bytes, stop on EOS=LF
2) Enter from    Voltmeter    up to 256 bytes, stop on EOS=LF
3) Set           Result       to Voltmeter
```

"Result" gets set to the second value read from the meter. The first value is lost, because line 2 replaces it with the new value. Normally, actions to process or store the data should follow an action which results in reading new data.

Execution

TestPoint executes action lists when events occur. Events can be:

- user input (mouse and keyboard)
- timers
- interrupts from hardware interfaces such as GPIB or A/D
- dynamic data exchange messages from other Windows applications

or, as an important special case:

- the result of an action executed in another action list

For example, if a data-entry object named "Amplitude" has the action list:

1)	Calculate	Formula	with x=Amplitude
2)	Set	Result	to Formula

and a pushbutton named "Preset" has this action list:

1)	Set	Amplitude	to 2.5

then clicking on "Preset" will not only put a new value into the "Amplitude" field on the panel, but also execute "Amplitude"'s action list, displaying a new result, just as if a the new value had been typed in directly.

New events can even be defined by external code added to TestPoint (using the Code object).

Each class of object defines the event it responds to, and provides an action list that may be filled in. Some object types, being purely for data processing or output, have no event or action list.

More Examples

This section contains selected example applications which are included with your copy of TestPoint, along with detailed explanations of how they work.

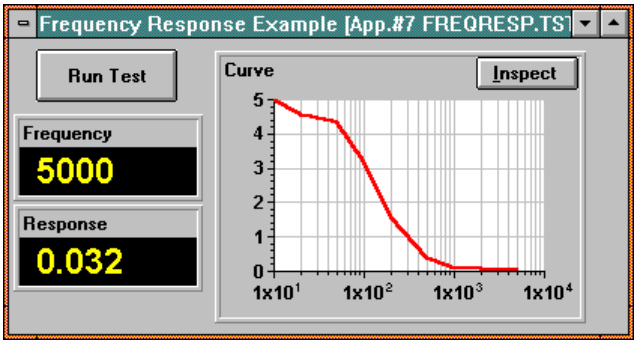
You may want to run these examples, or load them into the TestPoint editor to follow along as you read the explanation.

If you use the Debug menu to turn on the Single Step mode, you can execute the applications one action line at a time.

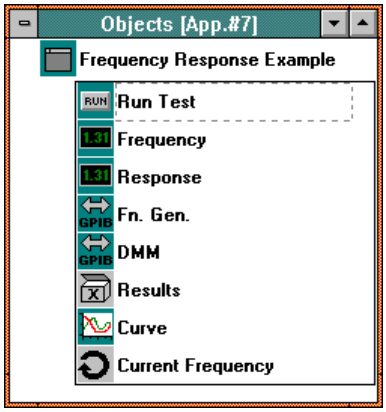
The View/Data menu command lets you see the data types and values of the objects as the application executes.

Frequency Response Example

This application can be found in the file **EXAMPLES\FREQRESP.TST** within your TestPoint directory.



This example uses a function generator to send waveforms of various frequencies to a unit under test, and a voltmeter to read the amplitude of an output. The resulting frequency response curve is graphed on a semi-log scale.



1)	Clear	Results	
2)	Clear Graph	Curve	
3)	Decade Series	Current Frequency	from 10^1 to 10^3,in 1,2,5 sequence
4)	Set	Frequency	to Current Frequency
5)	Output to	Fn. Gen.	with ":FREQ ", Current Frequency, term.=LF, send EOI?=1
6)	Enter from	DMM	up to 256 bytes, stop on EOS=LF or EOI
7)	Set	Response	to DMM
8)	Append to	Results	from Current Frequency, DMM
9)	End	Current Frequency	
10)	Draw Graph	Curve	with Results

When the "Run" button is clicked, the action list executes. Note that, in this example, the action list also executes once automatically when the application is started (placed in Mode=Run). This is because of the "Exec. actions at initialize" setting of the pushbutton.

First, lines 1 and 2 initialize the application. The object "Results" is a Container, in which the frequencies and response voltages will be accumulated.

Line 3 is the beginning of the main loop. It uses the Loop object "Current Frequency" and the "Decade Series" action, which steps through the values 10,20,50,100,200,500,1000,2000, and 5000. The body of the loop, lines 4 to 8, are executed 9 times, with the value of the "Current Frequency" object equal to each of these numbers.

Line 4 puts the current frequency value into the Display object "Frequency", so the user can monitor the progress of the application.

Line 5 sends a command to the function generator, which is an external GPIB instrument. The "Output to" action is used to send the constant string ":FREQ " followed by the current frequency value. In this example, we assume the instrument model being used accepts commands in this format.

Line 6 reads a voltage response value from the DMM (digital multimeter), which is also a GPIB instrument. The "Enter from" action reads characters from the device, and then automatically converts the result to a number. After line 6 executes, the data value of object "DMM" is the measured voltage.

Line 7 sets Display object "Response" to the measured voltage.

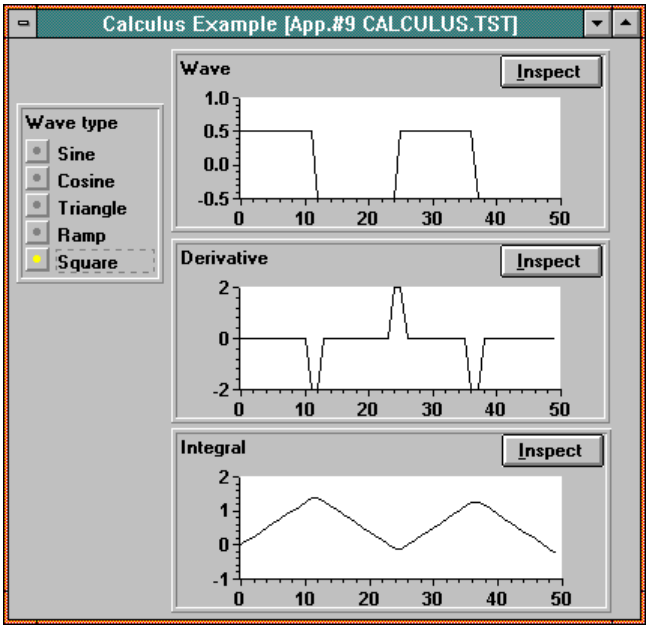
Line 8 accumulates the results of the test by appending the current frequency and response voltage to the container "Results".

After the loop is complete, line 10 draws a graph using the values in the container. The graph object settings have been configured for semi-log plotting with the desired grid and trace color options.

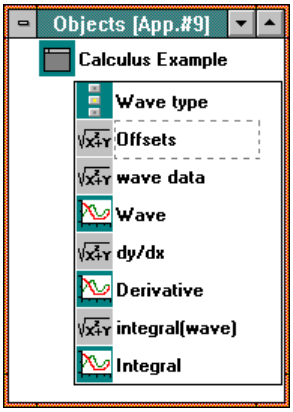
Note that this example uses the "Demo Mode" setting on the GPIB objects, so they do not attempt to access actual external devices. Instead, they read data from the file DEMO.DAT.

Calculus Example

This example may be found in the file **EXAMPLES\CALCULUS.TST** within your TestPoint directory.



This example generates a chosen type of waveform and then computes the derivative and integral of that function, and graphs all three.



Formulae for the math objects:

Offsets:

vector (0, 0, -0.499, -0.499, -0.499)

wave data:

generate (50,25,fn) + offset[fn]

dy/dx:

derivative2 (wave, pi()/12.5)

integral(wave):

integrate (wave, pi()/12.5)

Action list for Wave type:

1)	Calculate	Offsets	
2)	Calculate	wave data	with fn=Wave type offset=Offsets
3)	Draw graph	Wave	with wave data
4)	Calculate	dy/dx	with wave=wave data
5)	Draw graph	Derivative	with dy/dx
6)	Calculate	integral(wave)	with wave=wave data
7)	Draw graph	Integral	with integral(wave)

Every time a new choice is made by clicking the "Wave type" Selector object, the action list executes.

Line 2 generates the waveform, using the math function **generate(),** which takes arguments for number of points, waveform period, and waveform type. The waveform type argument **fn** comes from the value of the selector object. An offset is added to the waveform, depending on the chosen function, so that the waveform is centered around zero. Note that the offset comes from indexing a constant vector created in the "Offsets" math object in line 1.

Line 3 draws the waveform graph.

Lines 4 and 5 calculate the derivative and graph it. The math funciton **derivative2()** is used, given the waveform and the delta x value as parameters. TestPoint includes two functions: derivative() and derivative2(). Derivative2() is much faster, because it fits second-order curves to each set of three data values in the input waveform, while derivative() fits 4th-order curves to each set of five values.

Lines 6 and 7 calculate and graph the integral of the waveform.

For more detailed information

You can see a detailed description of each TestPoint object by choosing Help, Index from within the TestPoint editor. If you have the full (non-demo) version of TestPoint, the "Techniques and Reference" manual has complete information.