
A BASIC Plus Notebook

----

v2.4 /01 feb 99 / greg goebel

* This document describes the BASIC Plus extensions for HP BASIC on its different platforms. It provides a comprehensive description, including examples and illustrations.

Notes :

Ce document a été réalisé par le groupe responsable du support technique de la gamme des produits HP BASIC ou RMB (Rocky Mountain Basic) de la division MXD de Hewlett-Packard. Il est reproduit par BOURBAKY qui distribue l'ensemble des produits de cette gamme en France avec l'aimable autorisation de son auteur Greg Goebel.

Agilent Technologies
Innovating the HP Way
Channel Partner

****

```
Cie : BOURBAKY
      BP 53
      13, Rue des Alpes
      07302 TOURNON Cedex - France
Tel (Nat.) : 04 75 07 81 20    Tel (Int.) : +33 4 75 07 81 20
Fax (Nat.) : 04 75 07 29 74    Fax (Int.) : +33 4 75 07 29 74

web :    <http://www.bourbaky.com>
e-mail : <info@bourbaky.com>
```

****

Table Of Contents

[1.0] BPLUS (1): An Overview

v2.4 / 01 feb 99 / greg goebel

* This introductory chapter gives a broad description of BASIC Plus.

[1.1] INTRODUCTION

* BASIC Plus (or "BPlus") is a binary that extends the operation of the traditional HP BASIC system such as RMB Workstation, RMB-UX, or HP BASIC for Windows by adding user-interface features such as dialogs and widgets. This document provides a comprehensive introduction to BPlus. It assumes a familiarity with the concepts of HP BASIC.

This is a very large document and includes illustrations and detailed examples. For those of you who find such a big document more than you have time to wade through, don't worry. It's structured so that you don't have to read the whole thing.

This first chapter provides an overview of the subject. The next few chapters provide an outline that will allow you to use the product intelligently, and the remainder of the chapters explain the individual widgets in detail. The appendix covers the HELPX system used on RMB Workstations and RMB-UX, but not on HP BASIC for Windows.

* As with any document, it is impossible to know where your own work begins and that of your colleagues begins. Thanks to all who provided inputs to this document, and to those who allowed me to spend the time to work on it. As the author of this document, all mistakes and omissions are my own responsibility. For my readers who are not native English speakers, please forgive any use of slang terms that may be confusing.

[1.2] BASIC PLUS IN GENERAL

* BASIC Plus ("BPlus") provides a set of commands to help you create test-and-measurement user interfaces with a MOTIF windows-style appearance, using user-interface "dialogs" and "widgets".

In the RMB Workstation and RMB-UX environments, it also includes several applications, such as:

* A "Help" utility that you can invoke from HP BASIC to get information on any HP BASIC command.
* A "Help Compiler" to allow you to create your own help files for the help utility.
* A "Screen Builder" tool to help you lay out user interfaces.
* Other useful utilities such as a "Clock" and the "Notepad" editor.
* An "Applications Manager" that allows you to run these applications.

These additional features are not supported (or supported only to a very limited extent) in HP BASIC for Windows, since the MS Windows environment provides these facilities by default.

A dialog acts, from the program's point of view, as an INPUT statement with a TIMEOUT. From the user's point of view, it appears as a pop-up panel that goes away when the user makes a response. There are ten types of dialogs, from a simple dialog that displays a string to the user (who then clicks a button on the dialog panel to get rid of it) to a dialog that gives a file listing and asks for a file selection.

BPlus includes the following dialogs:

INFORMATION	Displays information, user responds with mouse button.
ERROR	Displays error message, user responds mouse button.
QUESTION	Displays question, user responds with mouse button.
WARNING	Displays warning, user responds mouse button.
NUMBER	Allows input of numeric data.
KEYPAD	Similar to NUMBER, but provides a calculator keypad.
STRING	Prompts user for text string.
COMBO	Allows entry by text input or from list.
LIST	Allows user to select from a list of items.
FILE	Allows user to select a file from a directory.

Widgets are much more flexible items that can be used to build sophisticated user interfaces. You can, for example, put together "virtual instruments" that include pushbuttons, sliders, text boxes, meters, graphics displays, and even pulldown menus, that can be controlled by a mouse.

BPlus includes the following widgets:

PANEL	Provides a "panel" to place other widgets.
SEPARATOR	Allows line to be drawn to separate regions on PANEL.
PUSHBUTTON	Provides "pushbutton" that can be clicked by mouse.
TOGGLEBUTTON	Similar to PUSHBUTTON.
RADIOBUTTON	Similar to PUSHBUTTON.
LABEL	Allows labels to be placed on user interface.
NUMBER	Allows input of numeric data.
KEYPAD	Similar to NUMBER, but provides a calculator keypad.
STRING	Allows string to be entered from user interface.
COMBO	Allows string entry or selection from a list of items.
LIST	Allows selection from a list of items.
FILE	Allows selection of disk volume, directory, or file.
BAR	Single-bar bargraph.
BARS	Multiple-bar bargraph.
METER	Meter.
LIMITS	Similar to a METER but the needle moves along a bar.
PRINTER	Allows output of multiple lines of text.
SLIDER	Allows numeric input with mouse using slider.
SCROLLBAR	Simplified version of SLIDER, used for scrolling.
XYGRAPH	Allows drawing of X-Y graphs.
STRIPCHART	Allows drawing of stripcharts.
HPGLVIEW	Allows viewing of HPGL files.
BITMAP	Allows display of .BMP or .XWD bitmap files.
PULLDOWN MENU	Used for building pulldown menu systems.
CASCADE MENU	Used for building lower-level pulldown menus.
MENU BUTTON	Provides menu selection in pulldown or cascade menus.
MENU TOGGLE	Similar to MENU BUTTON.
MENU SEPARATOR	Allows line to be drawn in menus as item separator.
SYSTEM	Used to build user interfaces & widget arrays.

* All of the dialogs and widgets possess a large number of attribute settings that allow you to customize their operation. Core attributes, for example, includes the position of the object on the screen, the background color of the object, and the size of the object.

There are also attributes specific to a certain type of object. For example, the STRIPCHART widget has over thirty additional attributes, including number of channels displayed, color assigned to each channel, and control over the number of points displayed, number of points in the display buffer, and how the display is updated.

Program Example: xchamber.rmb

[1.3] BPLUS FOR RMB WORKSTATIONS & RMB-UX

* The facilities provided by BPlus for the RMB Workstation and RMB-UX environments provide extensions not otherwise available in those environments.

The online Help utility provides a reference for all HP BASIC keywords. The HP BASIC Condensed Reference Guide information for HP BASIC keywords is at your fingertips. You can access information directly by keyword name or find the correct keyword using the keyword dictionary. In addition, you can copy simple examples from online help directly into the HP BASIC editor.

You can create custom help files for the Help utility by writing them with the Notepad utility -- as text files, using a simple HP text-formatting language called HELPX -- and then compiling them with the Help Compiler. You can also link a help file to a widget, allowing help to be displayed when you press the secondary mouse button on the widget.

The Screen Builder allows you to interactively build simple BPlus user interfaces.

Finally, the Applications Manager provides a Windows Program Manager-like user interface for using the BPlus tools. It allows you to run multiple BPlus applications, minimize them, or delete them.

* For RMB Workstation and RMB-UX, BPlus is provided as a separate product. You must have at least 3 MB of RAM and HP BASIC 6.2 or above. BPlus is loaded as a binary and can be stored as a part of your system. Once you've loaded and stored BPlus, you don't need to worry about including a special set of subprograms in each of your programs.

BPlus is supported on all RMB Workstation and RMB-UX platforms (including the PC Measurement Coprocessor) that have a bitmapped display.

BPlus for RMB is best used with a color display. It also works well with grayscale monochrome displays, though items that have distinctive colors on a color display may in some cases be mapped to indistinguishable shades on a grayscale display. It will also work on single-plane monochrome displays, but the appearance is not particularly impressive and that configuration is not recommended.

[1.4] BPLUS ON HP BASIC FOR WINDOWS

* BASIC Plus is included by default with the 6.3 or later release of HP BASIC for Windows. It includes all dialogs and widgets, but of the apps and application environment provided under RMB Workstation and RMB-UX the only one supported on HBW is the Screen Builder. The other applications are redundant in the Windows environment.

[2.0] BPLUS (2): Quick Tutor -- Getting Started, Apps, Dialogs

v2.4 / 01 feb 99 / greg goebel

* Now that you've read the overview, you know that BPlus extends the traditional HP BASIC language with three new groups of facilities: applications, dialogs, and widgets. This chapter and the three following will provide you with enough additional information to start working with BPlus.

[2.1] GETTING STARTED

* How hard a task is it to work with BPlus? There's two answers to that question.

The first answer is: don't be intimidated by BPlus! You can start working immediately with a small amount of training. BPlus dialogs and widgets are created and manipulated by simple rules, and with a little direction any reasonably competent HP BASIC programmer can be using them in a short time. It's like playing with a set of LEGO blocks; there are a few kinds of different blocks, and fitting them together is not very hard.

The second answer is, however: obtaining a mastery of the dialogs and widgets themselves isn't trivial, as there are many details to consider. The better you understand these details, the faster and better you will be able to implement your goals. Just like the LEGO blocks: the blocks may be simple, but you can put together some extremely complicated toys with them, and it helps a lot to have a plan and methodology for doing it. You need to have strategies to write programs with BPlus that are easy to construct, debug, and use.

Of course, there are no fixed rules for such strategies. This document uses techniques I have developed, but they are not the last word, users may find other techniques more to their liking. But they're a good starting point, and having some form of workable rules provides much more productivity than simply starting from nothing.

[2.2] INSTALLING & USING BASIC PLUS

* Installing and using BPlus under RMB is a slightly complicated subject. BPlus is an RMB binary, but it is very different from the RMB binaries of the past. Consider an RMB binary like the HFS or the similar DFS binary. If you perform the statement:

```
   LOAD BIN "DFS"
```

-- the DFS binary will be loaded and become part of the RMB system file. You can then store the system file, and when you boot RMB again, the DFS binary will be part of the BASIC system file.

This is true for BPlus as well. You can LOAD BIN "BPLUS", store the system, and then when you boot, the binary is part of the systm. However, by itself, all the BPlus binary does is take up memory.

This is because the BPlus binary itself is actually a "framework" into which various widgets, dialogs, and applications (like the Help application) can be "plugged". Each widget or dialog is stored on disk as a single file, or "binlet". You can mass-load the binlets, or simply load each binlet by default when you create a particular widget, dialog, or applications. The binlets are not saved when you store the system.

Binlet load is controlled by a configuration file. On boot, LOAD BIN "BPLUS", or SCRATCH A, RMD will search for a subdirectory called PLUS (or certain variations on that path) and check it for a file named CONFIG. This is a program listing file SAVEd as an HP-UX (or the equivalent MCP DOS) text file, containing configuration information written as program remarks. You can edit this file to specify:

* The disk volume where BPlus files are found.
* RAM usage by BPlus.
* The default BPlus system font.
* Mouse, SLIDER widget, and context-sensitive help handling.
* What fonts, widgets, and apps to preload.
* What applications are loaded into Application Manager.
* The name of a "tiling" bitmap file for the Applications environment.
* The PEN settings for the PENS used by BPlus itself for its widgets.
* What PENs are available to BPlus on different displays.
* The PEN mappings for the user under BPlus.

BPlus comes with a default CONFIG file, so you don't need to worry about the internals of that file when you're just getting started.

* Under HP BASIC for Windows, you also load BPLUS with:

```
   LOAD BIN "BPLUS"
```

However, in this case, you must do this every time you boot HBW. Under Windows, the widgets and dialogs are actually implemented as Windows dynamic link libraries (DLLs) and never become part of the BASIC system proper.

HBW also supports a CONFIG file, but of course some of the features uses in RMB are missing, such as those relating to the Applications Manager.

[2.3] GETTING STARTED WITH BASIC PLUS APPLICATIONS

* Under RMB, the core of the BPlus applications environment is the Applications Manager; this doesn't exist under HP BASIC for Windows, though HBW does share online Help and the Screen Builder app with RMB.

To run the Applications Manager under RMB, simply enter APP at the RMB input line, and the display will change color and the Applications Manager panel will pop up on the display:

```
+---+-----------------------------------------------+
| = |             Application Manager               |
+---+-----------------------------------------------+
| File  Help                                        |
+---------------------------------------------------+
|  +------+  +------+  +------+  +------+  +------+  |
|  |      |  |      |  |      |  |      |  |      |  |
|  |      |  |      |  |      |  |      |  |      |  |
|  +------+  +------+  +------+  +------+  +------+  |
|   Screen    Notepad   Clock      Help      Help    |
|   Builder                                Compiler  |
+---------------------------------------------------+
```

You can run the applications by double-clicking on them with your mouse. For example, double-click on "Notepad" and you get the Notepad editor:

```
+---+-------------------------------------------------+---+---+
| = |                     Notepad                     | x | X |
+---+-------------------------------------------------+---+---+
| File  Edit  Search                                          |
+-------------------------------------------------------------+
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
|                                                             |
+-------------------------------------------------------------+
```

Notepad provides a simple text editor, very similar to the Microsoft Windows Notepad editor. Similarly, clicking on "Clock" gives you:

```
+---+--------------------------+---+---+
| = |           Clock          | x | X |
+---+--------------------------+---+---+
| Display  Timer  Alarm  Options       |
+--------------------------------------+
|                                      |
|                  *                   |
|                                      |
|           *      |      *            |
|                  |                   |
|         *        +------  *          |
|                                      |
|           *             *            |
|                                      |
|                  *                   |
|                                      |
|              12:15:03 PM             |
|                                      |
+--------------------------------------+
```

Clock allows you to perform alarm and timer operations, change clock formats, and so on.

* The Help utility proivides a panel that lists a menu of HP BASIC and BPlus commands. You can either run it from Application Manager or just enter HELP at the RMB command line. Either way, you get the panel:

```
+---+---------------------------------------------------+---+
| = |                HP BASIC: Contents                 | X |
+---+---------------------------------------------------+---+
| File  SeeAlso  Programs                                   |
+----------+--------+------+----------+------+--------------+
| Contents | Search | Back | Copy Code | Quit |              |
+----------+--------+------+----------+------+------------+-+
| BASIC Keywords                                        |^|
|                                                       | |
|   Widget Dictionary                                   | |
|   Keywords By Category                                | |
|                                                       | |
| BASIC Plus Information                                | |
|                                                       | |
|   BASIC Plus Widget Reference                         | |
|   BASIC Plus Keyword Dictionary                       | |
|   BASIC Plus Examples                                 | |
|                                                       | |
|                                       line 11 of 16 |v|
+-------------------------------------------------------+-+
```

Note that this particular form for the Help utility is used on RMB. Under HBW, the default Windows Help utility is used instead. The two are similar -- not surprisingly, since the RMB Help utility was designed using the Windows Help Utility as a model.

You can use a mouse to click through hierarchies of Help topics and even copy code fragments into an RMB program. A few simple example programs are also available through Help.

You can specify a particular topic to display when you invoke help. For example, invoking:

```
   HELP "OUTPUT"
```

-- will bring up a panel that contains the Help entry for the OUTPUT statement. You can also specify a custom Help file:

```
   HELP "Saturn","PLANETS.HLP"
```

* The Help File Compiler allows you to create your own Help files for RMB. It won't work under Windows, howwever, and you have to use whatever Windows Help compiler you can get your hands on.

The Help File Compiler's user interface is simple:

```
+---+---------------------------------------------------+---+---+
| = |                 Help File Compiler                | x | X |
+---+---------------------------------------------------+---+---+
| File                                                          |
+---------------------------------------------------------------+
| Source File:                                                  |
| +-----------------------------------+ +-----------+ +-----------+ |
| |                                   | |   Help    | |  Compile  | |
| +-----------------------------------+ +-----------+ +-----------+ |
| Destination File:                     | System... | |   Stop    | |
| +-----------------------------------+ +-----------+ +-----------+ |
| |                                   | |   Try...  | |   Quit    | |
| +-----------------------------------+ +-----------+ +-----------+ |
| Compile Status                                                |
| +-----------------------------------------------------------+ |
| |                                                           | |
| |                                                           | |
| |                                                           | |
| |                                                           | |
| |                                                           | |
| |                                                           | |
| |                                                           | |
| |                                                           | |
| +-----------------------------------------------------------+ |
+---------------------------------------------------------------+
```

There's not much to it. You simply give it the name of the HP HELPX Help language-formatted source text file, the name of the output Help file, and it compiles the source into the output Help file.

The HELPX language is a simple derivative of a text formatting language called HP TAG, which HP uses in publishing manuals. HELPX is very simple to use and provides the ability to create Help files with hyperlinks and examples that can be embedded into the RMB editor from the Help utility. HELPX is described in more detail in a later chapter.

* If you run the Screen Builder, you get the user interface:

```
   +---+----------------------+---+---+ +-------------------------------+---+
   | = |   HP Screen Builder   | x | X | |            Panel0             | X |
   +---+----------------------+---+---+ +-------------------------------+---+
   | File  Widget  Options            | |                                   |
   +----------------------------------+ |                                   |
   | -- Widget Types ---------------- | |                                   |
   |                                  | |                                   |
   | [  Panel  ][Pulldown ][ Cascade ] | |                                   |
   |                                  | |                                   |
   | [ MButton ][ MToggle ][ MSepar  ] | |                                   |
   |                                  | |                                   |
   | [ PSepar  ][ Button  ][  Radio  ] | |                                   |
   |                                  | |                                   |
   | [ Toggle  ][Scrollbar][  Label  ] | |                                   |
   |                                  | |                                   |
   | [ String  ][ Numeric ][  List   ] | |                                   |
   |                                  | |                                   |
   | [ Combo   ][   Bar   ][  Bars   ] | |                                   |
   |                                  | |                                   |
   | [  File   ][ Limits  ][  Meter  ] | |                                   |
   |                                  | |                                   |
   | [ Printer ][ Slider  ][ StripC  ] | |                                   |
   |                                  | |                                   |
   | [ XYGraph ][ Bitmap  ][  HPGL   ] | |                                   |
   |                                  | |                                   |
   | [  Clock  ][ Keypad  ]            | |                                   |
   |                                  | |                                   |
   | -------------------------------- | |                                   |
   +----------------------------------+ +-----------------------------------+
```

You can click on the various icons at left to place them on the PANEL at right, and then move them around and edit them as you like. The resulting user interface is stored in a "descriptor file" that a program can use to build the defined user interface. The Screen Builder does not generate BASIC code.

* Finally, you can click on the second button from right on these applications to hide them and list them as entries in the "Minimized Windows" app, which will appear in the bottom-left corner of the display

when it is needed:

```
   +------------------------+
   |    Minimized Windows   |
   +------------------------+
   |[635   : Notepad       ]|
   |                        |
   |                        |
   |                        |
   +------------------------+
```

* Note that you cannot invoke RMB programs from Application Manager. You can only invoke applications created specifically for that environment. There are no tools available to customers for this purpose.

The applications are described in more detail in a later chapter.

[2.4] DIALOG & WIDGET SYNTAX

* Before we jump into dialogs and widgets I need to explain the basic nomenclature and rules of grammar, so we can have a common language to describe the details that follow. It is hard to extract a discussion of syntax from a description of features, however, so this section also provides a micro-introduction to dialogs and widgets that will be developed in following sections.

* A dialog is from the programming point of view very similar to an HP BASIC INPUT statement. It is easy to use an INPUT statement to prompt the user:

```
   INPUT "Do you want to quit (y/n)?",Reply$
```

This gives the prompt string ("Do you want to quit (y/n)?") and a return variable (Reply$). You can do the same thing, more or less, with a dialog:

```
   DIALOG "QUESTION","Do you want to quit?",Button
```

This gives the type of dialog ("QUESTION", must be upper case, by the way), the prompt string ("Do you want to quit?"), and a return variable ("Button").

In the case of the INPUT statement, you get the following string on the DISP line:

```
   Do you want to quit?
```

You then type in some text in reply and hit the enter key. With the DIALOG, you get a panel of the form:

```
   +------------------------------+---+
   |             QUESTION         | X |
   +------------------------------+---+
   |                                  |
   |  ?    Do you want to quit?       |
   |                                  |
   +----------------------------------+
   |      +--------+    +--------+    |
   |      |  Yes   |    |   No   |    |
   |      +--------+    +--------+    |
   +----------------------------------+
```

You can then use a mouse to click on the button you want, and the dialog will vanish. You can find out which button was clicked from the value of the return variable, "Button", which will contain 0 for "Yes", and 1 for "No".

In both cases, the calling program comes to a complete halt until you reply. The dialog, however, is a little smarter than the INPUT statement, in that you can give the dialog a timeout so it will go away on its own after a while:

```
   DIALOG "QUESTION","Do you want to quit?",Button;TIMEOUT 5
```

Now the dialog will go away after 5 seconds. Note that if the TIMEOUT occurs, "Button" will contain a "-1".

You can modify the dialog in many other ways -- subject to the nasty and (if you think about it) obvious constraint that all the modifications have to fit on the same line (since the program stops when you execute the DIALOG statement) -- but I need to develop a few more ideas before I can explain how.

* A widget, in contrast, does not look like a INPUT statement. It's much more like a "virtual" input or output device, a make-believe piece of hardware you create, and then destroy, with software. This illusion is easy to maintain since the widgets actually look like hardware devices: click buttons, audio sliders, displays, and the like.

In HP BASIC, you make a connection to an I/O device with an ASSIGN statement. When using BPlus, you use ASSIGN to make the connection to a widget, as well as create the widget:

```
   ASSIGN @Click TO WIDGET "PUSHBUTTON"
```

This creates a widget of type PUSHBUTTON (it immediately pops up on the display) and gives it an I/O path (or in BPlus terminology, a "widget handle") named "@Click". You disconnect from, and wipe out, this particular widget with the statement:

```
   ASSIGN @Click TO *
```

Following the model of a widget as an I/O interface further, you can interact with this widget, or virtual device, using CONTROL and STATUS statements.

However, on a real hardware interface, a CONTROL or STATUS statement will deal with sets of registers, normally consisting of various fields or bits that can be tricky to use to perform a desired action. BPlus takes a more direct approach: you control or interrogate the widget's features using "attribute-value" pairs, and the CONTROL-SET and STATUS-RETURN keywords.

Consider, as an example, the background color of the PUSHBUTTON we created above. You can set the background color to red with the CONTROL statement:

```
   CONTROL @Click;SET ("BACKGROUND":2)
```

-- where 2 is the HP BASIC pen code for the color red. You can similarly read back the pen value using a STATUS statement:

```
   STATUS @Click;RETURN ("BACKGROUND":Pencolor)
```

-- where Pencolor is a numeric variable that will store the value returned. (Many attributes can be both written and read.)

There are a wide number of attributes associated with each widget. Some are common to most widget types, for example:

* BACKGROUND: Specifies HP BASIC PEN color for widget background.
* FONT: Size and style of font to be used for text.
* VISIBLE: If 1, widget is visible; if 0, widget is hidden.
* X: X location of widget in pixels.
* Y: Y location of widget in pixels.
* WIDTH: Width of widget in pixels.
* HEIGHT: Height of widget in pixels.
* MAXIMIZABLE: If 1, widget can be popped up to full screen.
* MINIMIZABLE: If 1, widget can be hidden in Minimized Windows app.
* MOVABLE: If 1, widget can be moved around on display with a mouse.
* RESIZABLE: If 1, widget can be stretched with a mouse.
* PEN: Color of text.

These and other common attributes are discussed in detail in a later chapter. Other attributes are unique to sets of widget types or even a single widget type, and will be discussed in the chapters for the individual widgets.

Many widgets are also capable of generating HP BASIC interrupts, or "events". For example, if you enable an event for the PUSHBUTTON widget we created above with:

```
   ON EVENT @Click,"ACTIVATED" GOSUB Handler
```

-- you will cause a GOSUB to a routine named "Handler" whenever you click on the PUSHBUTTON. These widget events follow the same rules as other events in HP BASIC; you can ENABLE and DISABLE them, and you can turn them off with:

```
   OFF EVENT @Click,"ACTIVATED"
```

Note that we have to specify the event type -- "ACTIVATED" in this case -- in both the ON EVENT and OFF EVENT statements, since some widgets have multiple types of events associated with them that can be activated at the same time.

Since simply waiting in a loop waiting for something to happen:

```
   ON EVENT @Button,"ACTIVATED"
   LOOP
   END LOOP
```

-- is a waste of time in multiprocessing systems, BPlus defines a statement, WAIT FOR EVENT, that allows HP BASIC simply to go idle and wait for something to happen:

```
   ON EVENT @Button,"ACTIVATED"
   LOOP
     WAIT FOR EVENT
   END LOOP
```

* One relevant note on HP BASIC syntax before we proceed. Consider typing in the following ON EVENT statement:

```
   ON EVENT @Button,"ACTIVATED" CALL Handler(X1,X2)
```

This looks okay, but you'll get a SYNTAX ERROR. As it turns out, this isn't really a bug, it's a poorly-documented HP BASIC "feature": if you do a CALL statement as an argument to some other statement, such as IF-THEN or ON EVENT, you cannot specify any parameters for the CALL, and the "(X1,X2)" makes HP BASIC choke.

So the only alternative is to perform the call indirectly via a GOSUB:

```
   ON EVENT @Button,"ACTIVATED" GOSUB Call_handler
   ....
   Call_handler: !
   CALL Handler(X1,X2)
   RETURN
```

* There comes a point where the virtual interface concept of a widget breaks down. Widgets have a number of restrictions from a I/O-programming point of view:

   * You cannot perform ENTERs or OUTPUTs to them. This is a particular nuisance when you want to display formatted text, for instance; you have to do the formatted OUTPUT to a string variable and then use a CONTROL statement to put the string in the variable.

   * Similarly, you cannot use a RESET statement with a widget. You can use an ABORTIO statement, but it doesn't do anything.

   * Despite the fact that there is a PRINTER widget and a type of plotter widget (the HPGL VIEW widget), you can't do a PRINTER IS or PLOTTER IS to any widget.

* Okay, that's the fundamental rules for a widget. Now for elaborations.

You can perform CONTROL;SET or STATUS;RETURN statements on multiple attributes at once. For one useful example, consider:

```
   CONTROL @Click;SET ("X":Btnx,"Y":Btny,"WIDTH":Btnw,"HEIGHT":Btnh)
```

You can also perform the SET (or RETURN) statements as part of the ASSIGN statement:

```
   ASSIGN @Btn TO WIDGET "PUSHBUTTON";SET ("X":X,"Y":Y),RETURN ("LABEL":F$)
```

This gets clumsy, however, and can make your program hard to read and edit.

Another trick is to store all the attribute names in a string array and all the matching values in another array of the same size, and then invoke a SET statement using the matched arrays to set them all at once, as shown below:

```
   Attributes$(1)="X"
   Attributes$(2)="Y"
   Attributes$(3)="WIDTH"
   Attributes$(4)="HEIGHT"
   Values(1)=Btnx
   Values(2)=Btny
   Values(3)=Btnw
   Values(4)=Btny
   CONTROL @Click;SET (Attributes(*):Values(*))
```

Similarly, you can RETURN values from a widget using the same trick:

```
   STATUS @Click;RETURN (Attributes(*):Values(*))
```

One thing to remember, though, is that the value array is going to be either all numeric or all strings, so you may need two sets of arrays. This trick might seem to be more trouble than it's worth, and for widgets it often is. Where it comes in handy is for dialogs.

Remember a few pages back, where I said that you could make many adjustments on dialogs, but you were restricted to doing it on one line?

As with widget ASSIGN statements, you can tack SET or RETURN clauses onto a DIALOG statement. The only way to do this in a line of reasonable length is to use arrays:

```
   T$="STRING"
   P$="What is your name?"
   DATA "X","Y","WIDTH","HEIGHT","BACKGROUND"
   READ A$(*)
   V(1)=Dx
   V(2)=Dy
   V(3)=Dw
   V(4)=Dh
   V(5)=Pencolor
   DIALOG T$,P$,Button;TIMEOUT 5,SET (A(*):V(*),RETURN ("VALUE":Name$)
```

This statement creates a STRING dialog with a 5-second timeout, a given width and X,Y coordinate, and a given background color. Since a STRING dialog also has a field where you can enter a string, the DIALOG statement includes a RETURN attribute, "Value", that puts the string into a variable, "Name$".

* At this point, you should be familiar with the syntax for creating, modifying, and deleting widgets and dialogs. We can now describe their operation in more detail.

[2.5] GETTING STARTED WITH DIALOGS

* As you know by now, a dialog is programmatically similar to an INPUT statement: it asks the user a question and brings the program to a halt until the user responds. The DIALOG statement:

```
   DIALOG "INFORMATION","Time to go home!"
```

-- gives the INFORMATION dialog:

```
   +--------------------------------+---+
   |            INFORMATION         | X |
   +--------------------------------+---+
   |                                    |
   |  I        Time to go home!         |
   |                                    |
   +------------------------------------+
   |             +--------+             |
   |             |   OK   |             |
   |             +--------+             |
   +------------------------------------+
```

I didn't bother to include a variable to catch the value of the button. In this case, it's always going to be 0, so there's no reason to worry about it.

In the case of a dialog that has more than one button, such as a QUESTION dialog:

```
   DIALOG "QUESTION","Are you for real?",Btn
```

```
   +------------------------------------+
   |              QUESTION              |
   +------------------------------------+
   |                                    |
   |  ?       Are you for real?         |
   |                                    |
   +------------------------------------+
   |      +--------+    +--------+      |
   |      |  Yes   |    |   No   |      |
   |      +--------+    +--------+      |
   +------------------------------------+
```

-- then a return button index is returned in the "Btn" variable -- a "0" for "Yes", a "1" for "No" -- which is a little counterintuitive since HP BASIC usually thinks of a "0" as FALSE and a "1" as TRUE, but it's just returning a number that reflects the ordering of the buttons from left to right, starting at 0.

This is simple enough. There are additional features to consider. For example, dialogs have an attribute called "DIALOG BUTTONS" that allow you to change the number and labels of the buttons on the dialog panel: you define a string array with the labels and give it to the "DIALOG BUTTONS" attribute as a parameter:

```
   10     INTEGER Btn
   20     DIM P$[100],S$(1:3)[32]
   30     DATA "Fire Lasers","Fire Missile","Stand Down"
   40     READ S$(*)
   50     P$="Weapon Systems Command?"
   60     DIALOG "QUESTION",P$,Btn;SET("DIALOG BUTTONS":S$(*))
   70     SELECT Btn
   80     CASE 0
   90       BEEP
   100      DIALOG "INFORMATION","Lasers fired!"
   110    CASE 1
   120      BEEP
   130      DIALOG "INFORMATION","Missile fired!"
   140    CASE 2
   150      DIALOG "INFORMATION","Weapons system standing down!"
   160    END SELECT
   170    END
```

This pops up a dialog:

```
   +----------------------------------------------------------------+---+
   |                             QUESTION                           | X |
   +----------------------------------------------------------------+---+
   |                                                                    |
   |     ?                   Weapon Systems Command?                    |
   |                                                                    |
   +--------------------------------------------------------------------+
   |  +-------------------+  +------------------+  +------------------+  |
   |  |    Fire Lasers    |  |   Fire Missile   |  |   Stand Down     |  |
   |  +-------------------+  +------------------+  +------------------+  |
   +--------------------------------------------------------------------+
```

The "Fire Lasers" button will return a value of 0; the "Fire Missile" button will return a value of 1; the "Stand Down" button will return a value of 2. By default, if you press the RETURN key on the keyboard in response to the dialog, you will get a value of 0. You can change this using the "DEFAULT BUTTON" attribute: you just give it the number of the button (counting from 0 and from the left) you want to be the default.

You can make any number of buttons in the dialog by giving DIALOG BUTTONS a string array of the appropriate size. You're not restricted to the dialog's default number of buttons, though above about three or four buttons the dialog becomes too wide to be practical.

Custom buttons are a very handy technique in building practical applications, since they allow you to implement low-level decisions without cluttering up your user interface with hardwired widgets. The dialogues are there when needed and go away when they are not needed.

* So far you have seen the QUESTION and INFORMATION dialogs. They are

similar, except for slightly different cosmetics and a different number of default buttons. Two other dialogs -- ERROR and WARNING -- are almost identical to the INFORMATION dialog, except for cosmetics.

These widgets generate output only (or -- in the case of the QUESTION dialog -- very limited input). The rest of the dialogs provide a wider range of inputs.

* The NUMBER dialog allows you to input a number:

```
   DIALOG "NUMBER","Please enter a number:",Btn;RETURN ("VALUE":Numval)
```

This pops up the dialog:

```
   +------------------------+---+
   |           NUMBER       | X |
   +------------------------+---+
   |                            |
   |    Please enter a number:  |
   |                            |
   |  +----------------------+  |
   |  |                      |  |
   |  +----------------------+  |
   |                            |
   +----------------------------+
   |   +--------+   +--------+  |
   |   |   OK   |   | Cancel |  |
   |   +--------+   +--------+  |
   +----------------------------+
```

Note that you can specify different numeric input formats such as hexadecimal, binary, integer, and so on. The NUMBER dialog will not allow you to enter an format different from the one specified.

* The KEYPAD dialog:

```
   DIALOG "KEYPAD","Please enter a number:",Btn;RETURN ("VALUE":Numval)
```

-- performs exactly the same function, but uses a calculator-keypad format, as shown below:

```
   +-------------------------+---+
   |          KEYPAD         | X |
   +-------------------------+---+
   |    Please enter a number:   |
   |  +-----------------------+  |
   |  |                       |  |
   |  +-----+-----+-----+-----+  |
   |  |  A  |  B  |  C  |  D  |  |
   |  +-----+-----+-----+-----+  |
   |  |  7  |  8  |  9  |  E  |  |
   |  +-----+-----+-----+-----+  |
   |  |  4  |  5  |  6  |  F  |  |
   |  +-----+-----+-----+-----+  |
   |  |  1  |  2  |  3  | <-- |  |
   |  +-----+-----+-----+-----+  |
   |  |  -  |  0  |  .  | --> |  |
   |  +-----+-----+-----+-----+  |
   |  | CLR | DEL | INS | ENT |  |
   |  +-----+-----+-----+-----+  |
   +-----------------------------+
   |   +--------+   +--------+   |
   |   |   OK   |   | Cancel |   |
   |   +--------+   +--------+   |
   +-----------------------------+
```

* The STRING dialog (which we've already seen as an example in the previous section) improves on these dialogs by providing a field into which a user can type a string of text:

```
   DIALOG "STRING","Enter your password:",Btn;RETURN ("VALUE":P$)
```

-- creates:

```
   +--------------------------------+---+
   |              STRING            | X |
   +--------------------------------+---+
   |                                    |
   |         Enter your password:       |
   |                                    |
   |     +-------------------------+    |
   |     |                         |    |
   |     +-------------------------+    |
   +------------------------------------+
   |      +--------+    +--------+      |
   |      |   OK   |    | Cancel |      |
   |      +--------+    +--------+      |
   +------------------------------------+
```

The text typed in to the STRING dialog is returned through the "VALUE" attribute and its "Pass$" value.

* The COMBO dialog combines a string-input field like that of the STRING dialog with a list (defined by a string array) that you can pull down with a mouse. For example:

```
   DATA "German","French","English","Japanese","Spanish"
   READ L$(*)
   Prompt$="Select your language:"
   DIALOG "LIST",Prompt$,Button;SET ("ITEMS":L$(*)),RETURN ("TEXT":S$)
```

This gives the dialog:

```
   +--------------------------------+---+
   |               COMBO            | X |
   +--------------------------------+---+
   |                                    |
   |        Select your language:       |
   |                                    |
   |    +------------------------+-+    |
   |    |                        | |    |
   |    +------------------------+-+    |
   |                                    |
   |                                    |
   +------------------------------------+
   |      +--------+    +--------+      |
   |      |   OK   |    | Cancel |      |
   |      +--------+    +--------+      |
   +------------------------------------+
```

You can enter text in the text field or click on the bar at its right to get the list:

```
   +--------------------------------+---+
   |               COMBO            | X |
   +--------------------------------+---+
   |                                    |
   |        Select your language:       |
   |                                    |
   |    +------------------------+-+    |
   |    |                        | |    |
   |    +------------------------+-+    |
   |    |German                  |^|    |
   +----|French                  | |----+
   |    |English                 | |    |
   |    |Japanese                | |    |
   |    |Spanish                 |v|    |
   |    +------------------------+-+    |
   +------------------------------------+
```

Note how the COMBO dialog returns the input text or list input through a string accessed by the TEXT attribute. You could also access list inputs through the SELECTION attribute, as with the LIST dialog, but then you will miss any inputs typed into the text field.

* A LIST dialog is like a COMBO widget that only has a list. It allows you to select from a list of items defined by a string array:

```
DATA "German","French","English","Japanese","Spanish"
READ L$(*)
Prompt$="Select your language:"
DIALOG "LIST",Prompt$,Button;SET ("ITEMS":L$(*)),RETURN ("SELECTION":Index)
```

This yields:

```
   +---------------------------------+---+
   |                LIST             | X |
   +---------------------------------+---+
   |                                     |
   |       Select your language:         |
   |                                     |
   |  +---------------------------+-+    |
   |  | German                    |^|    |
   |  | French                    | |    |
   |  | English                   | |    |
   |  | Japanese                  | |    |
   |  | Spanish                   |v|    |
   |  +---------------------------+-+    |
   |                                     |
   +-------------------------------------+
   |      +--------+    +--------+       |
   |      |   OK   |    | Cancel |       |
   |      +--------+    +--------+       |
   +-------------------------------------+
```

The value clicked is returned from the dialog through the SELECTION attribute as an index to the string array. Note (very importantly) that in all cases where a BPlus dialog or widget returns the value of an index, the first index is always assumed to be 0, even if another base array index is defined in the BASIC program.

That means that if you click on the entry for "Japanese" in the example above, you get a SELECTION value of 3 -- not a value of 4. (You'll get a "-1" for an index if no item was selected.)

The LIST dialog can also be set to a MULTISELECT mode where you can click multiple entries in the list. SELECTION is then given a numeric array as a value. Entries that are selected return a "1" in their corresponding numeric array elements. Entries that are not selected return a "0".

* Finally, the FILE dialog allows you to select a file or a directory:

```
   DIALOG "FILE",Prompt$,Btn;RETURN ("SELECTION":S$),TIMEOUT 5
```

This yields:

```
   +-------------------------------------+---+
   |                  FILE               | X |
   +-------------------------------------+---+
   |                                         |
   |           Please select a file:         |
   |                                         |
   | +-------------------------------------+ |
   | |                                     | |
   | | Current directory:      Directories: | |
   | | +---------------------+  +--------+-+ | |
   | | |                     |  |        | | | |
   | | +---------------------+  +--------+-+ | |
   | |                                     | |
   | | File wildcard:          Files:      | |
   | | +---------------------+  +--------+-+ | |
   | | |                     |  |        | | | |
   | | +---------------------+  +--------+-+ | |
   | |                                     | |
   | | File selection:                     | |
   | | +---------------------------------+ | |
   | | |                                 | | |
   | | +---------------------------------+ | |
   | +-------------------------------------+ |
   +-----------------------------------------+
   |         +--------+     +--------+       |
   |         |   OK   |     | Cancel |       |
   |         +--------+     +--------+       |
   +-----------------------------------------+
```

The file selected is returned through the SELECTION attribute. You can also obtain a directory name through the DIRECTORY attribute.

More details about dialogs are available in a later chapter.

---

[3.0] BPLUS (3): Quick Tutor -- Widgets & TEMPLATE

v2.4 / 01 feb 99 / greg goebel

* Programming with BPlus widgets has much in common with programming with BPlus dialogs, but there are differences as well. Widgets are useful for building the "fixed" parts of a program's user interface -- the buttons, meters, and barcharts you will see on the display.

Dialogs are useful for dynamic conditions -- for example, to warn the user when he or she is taking a step that may have severe consequences ("ARE YOU SURE YOU WANT TO BEGIN THE SELF-DESTRUCT SEQUENCE?!"), or to indicate that some error condition has occurred ("CORE MELTDOWN IMMINENT, START RUNNING NOW!"), or to request detail information from the user ("WHAT COLOR IS YOUR UNDERWEAR?").

---

---

[3.1] GETTING STARTED WITH WIDGETS

* Let's learn widgets by walking through building a program with a BPlus user interface, starting with the simplest of the widgets, the PANEL.

The PANEL is just that: a PANEL, nothing more than a rectangle on a screen. Its primary function is to provide a "template" for other widgets. Creating one is easy. All you need is the statement:

```
   ASSIGN @Main TO WIDGET "PANEL"
```

The widget handle @Main is an arbitrary name; it can be almost any word. The "PANEL" designation gives the widget type, and remember that it must be in capitals.

When this is executed, the PANEL will pop up on the display. Let's write a simple example program to demonstrate. This program will clear the display, save itself as a convenience, build the PANEL, wait for 30 seconds, and then die.

```
   10    ! A few preliminaries -- save the file, clean off the display:
   20    !
   30    CLEAR SCREEN
   40    DISP "SAVING FILE!"
   50    RE-SAVE "TEST"
   60    !
   70    ! Now get down to business:
   80    !
   90    DISP "BUILDING PANEL"
   100   ASSIGN @Main TO WIDGET "PANEL"
   110   DISP
   120   !
   130   ! Wait 30 seconds, then exit.
   140   !
   150   WAIT 30
   160   !
   170   ! Exit here, restore everything.
   180   !
   190 Finis: !
   200   ASSIGN @Main TO *         ! Wipe out PANEL.
   210   CLEAR SCREEN
   220   DISP "DONE"
   230   END
```

The PANEL appears on the display:

```
   +--------------------------------------------------------------------+
   |                                                                    |
   |  +------------------------------+---+                              |
   |  |              PANEL           | X |                              |
   |  +------------------------------+---+                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   |  |                                  |                              |
   +--------------------------------------------------------------------+
```

* This PANEL isn't very conveniently situated, but you can move and size it to wherever you like on the display by setting an origin (using the attributes "X" and "Y") and its height and width (using the attributes "HEIGHT" and "WIDTH"). These attributes are specified in pixels (display dots) with the origin (0,0) coordinate at the upper-left corner of the display.

For example, a standard VGA-class display has a resolution of 640 by 480 pixels. You could define the PANEL as half the width and height of the display and centered, using the following CONTROL statement:

```
   CONTROL @Main;SET ("X":160,"Y":120,"WIDTH":320,"HEIGHT":240)
```

Add this statement after the ASSIGN statement:

```
   ...

   90    DISP "BUILDING PANEL"
   110   ASSIGN @Main TO WIDGET "PANEL"
   120   CONTROL @Main;SET ("X":160,"Y":120,"WIDTH":320,"HEIGHT":240)
   ...
```

-- and you get the display:

```
   +---------------------------------------------------------------------+
   |                                                                     |
   |                                                                     |
   |                                                                     |
   |                                                                     |
   |                +-------------------------------+---+                |
   |                |              PANEL            | X |                |
   |                +-------------------------------+---+                |
   |                |                                   |                |
   |                |                                   |                |
   |                |                                   |                |
   |                |                                   |                |
   |                |                                   |                |
   |                |                                   |                |
   |                +-----------------------------------+                |
   |                                                                     |
   |                                                                     |
   |                                                                     |
   |                                                                     |
   |                                                                     |
   +---------------------------------------------------------------------+
```

Now, at the risk of making things too complicated early on, I want to point out that I have just done something that you should under no circumstances do when building a BPlus user interface: I actually specified actual numbers when I executed the CONTROL statement -- that is, I "hard-coded" the values.

This might seem harmless enough (and certainly in this example, it is) but consider:

* When you start to fit more widgets together on the display to build your user interface, hard-coding the values is going to get nasty. Every time you change one value, you'll have to scroll around in the program and change another.

  A better idea is to define the values of widget coordinates and dimensions as a list of variables before you create the widgets.

* Hard-coding the values is okay as long as you're stuck on one display system, but suppose you're writing an HP BASIC program that you will run on different displays? (That may happen to you even if you don't intend it. Suppose you replace your old obsolete system and find out that your new spiffy one has a different display resolution?)

  In some cases, you may want your application to scale itself to the display size, by defining its own dimensions as some fraction of the display dimensions, as returned by the GESCAPE CRT,3 statement. In that case, hard-coding dimensions and coordinates would be out of the question.

For the sake of simplicity, we won't attempt to design an application that scales to the display (it isn't always a good idea), but we will use variables as a matter of simple prudence. Let's assume a VGA-class display. This gives:

```
...

80    !
90    INTEGER Dw,Dh,Pw,Ph,Px,Py
100   Dw=640                      ! Display width in pixels.
110   Dh=480                      ! Display height in pixels.
120   Pw=Dw/2                     ! PANEL width is half display width.
130   Ph=Dh/2                     ! Ditto for PANEL height.
140   Px=(Dw-Pw)/2                ! Center panel.
150   Py=(Dh-Ph)/2
160   !
170   DISP "BUILDING PANEL"
180   ASSIGN @Main TO WIDGET "PANEL"
190   CONTROL @Main;SET ("X":Px,"Y":Py,"WIDTH":Pw,"HEIGHT":Ph)
200   DISP
210   !
...
```

Now you can build dialogs and widgets. The next problem is to figure out what to do with them.

[3.2] A TEMPLATE PROGRAM -- INTRODUCTION

* Figuring out how to make use of dialogs and widgets is tricky. BPlus is complicated and it's hard to know where to start to build a program with it, particularly if you don't have a specific project in mind as an end goal.

Well, having no specific project in mind does suggest to a good Zen-minded programmer a nice thing to get started with: a general-purpose BPlus program that can be easily modified to do a wide variety of tasks.

Such a program is very useful. When you are programming in HP BASIC itself, you can easily PRINT output for the user, INPUT data from the user, and branch to routines using ON KEY statements. Put more simply, HP BASIC provides a ready-made user interface for you. With BPlus, in contrast, you have to build a user interface from widgets and dialogs. There's a certain amount of overhead you have to invest just to get to square one.

Once you have invested this overhead, however, you can do far more with BPLus using much less effort, and much of the overhead will be precisely the same no matter what your goal is. If you write a standard program that takes care of this overhead in order to provide facilities useful for simple tasks, that standard program can be then used as a basis for more complicated ones.

We will build such a program in the rest of this chapter, a process that should give you a comfortable feeling for how BPlus works. However, getting that comfortable understanding means that a large number of new concepts must be introduced. Since they can't all be introduced at the same time, you may have to read through this material twice, once to get an overview of these fundamental concepts and a second time to link them together.

But please don't expect full comprehension from this chapter, either, since some of the concepts are elaborate and demand a more detailed description. Such descriptions are provided in later chapters.

[3.3] A TEMPLATE PROGRAM -- ORGANIZATION

* What simple facilities would be useful in our template program? A good starting point here is the familiar: HP BASIC. As noted above, an HP BASIC program normally relies on three statements to provide user-interface services:

   * PRINT (or DISP) for display output.
   * INPUT for user input.
   * ON KEY to tell the program to perform various tasks.

Our simple program can provide all these facilities with widgets and dialogs:

* A PRINTER widget, which provides a scrolling text display, not unlike the text output on an RMB workstation's display.

* A STRING dialog (shown in the previous chapter) to get input from the user.

* A pulldown menu system for telling the program to perform various tasks. Such a system can be built using two widgets: the PULLDOWN MENU widget, which creates a menu bar across the top of a PANEL and creates an empty pulldown menu, plus one or more MENU BUTTONs, which can be used to populate the pulldown menu and let the user select what task to perform.

* So, our template program, which we'll cleverly call TEMPLATE, requires the following BPlus elements:

* A PANEL to provide a framework for the other widgets.
* A PULLDOWN MENU widget to build a pulldown menu on the PANEL.
* One or more MENU BUTTONs to populate the pulldown menu and execute tasks.
* A PRINTER widget to provide scrolling text output.
* A STRING dialog to get input from the user.

TEMPLATE has, at the highest level, the following organization:

* Define variables and perform other initializations.
* Build the PANEL.
* Put a PULLDOWN MENU on the PANEL.
* Put some MENU BUTTONS in the PULLDOWN MENU.
* Put the PRINTER widget in the PANEL.
* Set up a linkage between the MENU BUTTONS and the tasks they execute.
* Loop forever and wait to be told to quit or execute a function.

* Code to perform housekeeping on program exit (called by MENU BUTTON).
* A routine to use STRING dialog to get input (available to any routine).
* A routine to generate output to PRINTER widget (available to any routine).

As noted in this list, there has to be some way for the user to exit the program, so our PULLDOWN MENU has to have at least one MENU BUTTON, to allow it to quit the program. And for testing purposes, we also ought to have a way to test the STRING dialog and the PRINTER widget, so we need to have a MENU BUTTON to call the STRING dialog input routine, and a MENU BUTTON to print something; the time of day will do fine.

* Before we proceed, remember that TEMPLATE is just that, a template. It does no useful task itself, but it can be easily modified to do so. All you have to do is plug in new routines to perform the tasks you want. These routines can be executed by calling them with added MENU BUTTONs, and can get input by calling the STRING dialog routine and can generate output by calling the PRINTER widget output routine.

TEMPLATE also reflects the organization of a cleanly-designed BPlus program:
it builds a user interface, sets up a pulldown menu system, and loops, waiting for user input. The actual work in the program is done by a set of modular routines that are called by the user from the pulldown menu system. The program will also normally have a set of modular routines to provide services to the rest of the program. This architecture is easy to build, easy to maintain, and easy to extend and debug.

* Now back to our discussion. The first thing to do is make up a list of widgets, and give them names and descriptions, sort of like laying out all our parts where we can see them:

@Main	Main PANEL widget.
@Menu	PULLDOWN MENU widget.
@Get_text	MENU BUTTON widget to call STRING dialog test routine.
@Print_time	MENU BUTTON widget to call PRINTER widget test routine.
@S	MENU SEPARATOR widget (see below).
@Quit	MENU BUTTON widget to quit program.
@Prn	PRINTER widget for text output.

What's this MENU SEPARATOR business? No big deal, all it does is put a separating line in the menu so the MENU BUTTONs that call the working routines are visually distinct from the MENU BUTTON used to quit the program.

Note that the STRING dialog used in TEMPLATE isn't in this list. That's because as a dialog, it has no permanent existence, it is created to perform a task and is destroyed once it has performed that task, and so we don't have to give it a handle or otherwise account for it in detail. (This is actually an advantage of using dialogs and a good reason for making extensive use of them.)

* This done, we then sketch out the user interface, giving its appearance and labeling where the widgets lie:

```
+--------------------------------------------------------+---+
|                            @Main                       | X |
+--------------------------------------------------------+---+
| @Menu                                                      |
+---------------+--------------------------------------------+
|  @Get_text    | --------------------------------------+ |
|  @Print_time  |             @Prn                      | |
+- @S ----------+                                       | |
|  @Quit        |                                       | |
+---------------+                                       | |
| |                                                     | |
| |                                                     | |
| |                                                     | |
| +-----------------------------------------------------+ |
+------------------------------------------------------------+
```

Note that the PRINTER widget, @Prn, actually takes up all the interior area of the @Main PANEL, but it's shown as indented for clarity here.

In a more complicated user interface, you could then use this "widget map" to lay out sets of variables on the map to define the widget locations, but in one this simple that's hardly necessary. So we're now ready to build a first pass version of TEMPLATE.

[3.4] A TEMPLATE PROGRAM -- WRITING TEMPLATE

* A scratch first-pass program to implement TEMPLATE follows. Details will be discussed after the listing.

```
   10    ! *****************************************************************
   20    !
   30    ! TEMPLATE / v1.0
   40    !
   50    ! *****************************************************************
   60    !
   70    ! Variables to get button and string values from dialog.
   80    !
   90    INTEGER Btn
   100   DIM S$[256]
   110   !
   120   ! Variables to store widget & display coordinates and dimensions:
   130   !
   140   INTEGER Px,Py,Pw,Ph        ! Main PANEL variables.
   150   INTEGER Dw,Dh,Iw,Ih        ! Display & inside PANEL dimensions.
   160   !
   170   Dw=640                     ! Display dimensions.
   180   Dh=480
   190   Pw=Dw*.75                  ! Center PANEL in display.
   200   Ph=Dh*.75
   210   Px=(Dw-Pw)/2
   220   Py=(Dh-Ph)/2
   230   !
   240   ! *****************************************************************
   250   !
   260   ! Create the PANEL widget.
   270   !
   280   CLEAR SCREEN
   290   ASSIGN @Main TO WIDGET "PANEL";SET("VISIBLE":0)
   300   CONTROL @Main;SET ("X":Px,"Y":Py,"WIDTH":Pw,"HEIGHT":Ph)
   310   !
   320   ! Set up menu.
   330   !
   340   ASSIGN @Menu TO WIDGET "PULLDOWN MENU";PARENT @Main
   350   CONTROL @Menu;SET ("LABEL":"Menu")
```

```
 360   !
 370   ASSIGN @Get_text TO WIDGET "MENU BUTTON";PARENT @Menu
 380   CONTROL @Get_text;SET ("LABEL":"Get_text")
 390   !
 400   ASSIGN @Print_time TO WIDGET "MENU BUTTON";PARENT @Menu
 410   CONTROL @Print_time;SET ("LABEL":"Print Time")
 420   !
 430   ASSIGN @S TO WIDGET "MENU SEPARATOR";PARENT @Menu
 440   !
 450   ASSIGN @Quit TO WIDGET "MENU BUTTON";PARENT @Menu
 460   CONTROL @Quit;SET ("LABEL":"Quit Program")
 470   !
 480   ! Build PRINTER widget.
 490   !
 500   COM @Prn
 510   ASSIGN @Prn TO WIDGET "PRINTER";PARENT @Main
 520   STATUS @Main;RETURN ("INSIDE WIDTH":Iw,"INSIDE HEIGHT":Ih)
 530   CONTROL @Prn;SET ("X":0,"Y":0,"WIDTH":Iw,"HEIGHT":Ih)
 540   !
 550   ! Set up events for menu entries.
 560   !
 570   ON EVENT @Get_text,"ACTIVATED" GOSUB Get_text
 580   ON EVENT @Print_time,"ACTIVATED" GOSUB Print_time
 590   ON EVENT @Quit,"ACTIVATED" GOTO Finis
 600   !
 610   CONTROL @Main;SET ("VISIBLE":1)   ! Make PANEL visible.
 620   CALL Printit("Waiting ...")
 630   !
 640   LOOP
 650   END LOOP
 660   STOP
 670   !
 675   ! ********************** GO HERE WHEN DONE **********************
 680   !
 690  Finis: !
 700   ASSIGN @Main TO *                      ! Kill off main panel.
 710   STOP
 720   !
 730   ! ********************** END OF MAIN PROGRAM ********************
 740   !
 750   ! Routine to get a string input from user via a DIALOG.  If the
 760   ! user doesn't cancel the input, the routine sends it to the
         PRINTER
 770   ! widget.
 780   !
 790  Get_text: !
 800   DIALOG "STRING","Please input a string:",Btn;RETURN("VALUE":S$)
 810   IF Btn=0 THEN CALL Printit(S$)
 820   RETURN
 830   !
 840   ! Routine to print time.
 850   !
```

```
860  Print_time: !
870   CALL Printit(TIME$(TIMEDATE))
880   RETURN
890   !
900   END
900   !
910   ! ************************ SUBS FOLLOW HERE *********************
920   !
930   ! Sub to print text to PRINTER widget.
940   !
950   SUB Printit(S$)
960     COM @Prn
970     CONTROL @Prn;SET("APPEND TEXT":S$)
980   SUBEND
990   !
1000  ! *********************** That's All, Folks! ********************
```

Run this program and you get:

```
+-------------------------------------------------------+---+
|                          PANEL                         | X |
+-------------------------------------------------------+---+
| Menu                                                      |
+-----------------------------------------------------------+
| +-------------------------------------------------------+ |
| | Waiting ...                                            | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| |                                                        | |
| +-------------------------------------------------------+ |
+-----------------------------------------------------------+
```

If you use the mouse to click on the "Menu", you get the following menu:

```
| Menu
+------------+------------
| Get text   |
| Print time |
+------------+
| Quit       |
+------------+
|
```

Click on "Get text" and a STRING dialog pops up:

```
   +--------------------------------+---+
   |               STRING           | X |
   +--------------------------------+---+
   |                                    |
   |        Please input a string:      |
   |                                    |
   |    +---------------------------+   |
   |    |                           |   |
   |    +---------------------------+   |
   +------------------------------------+
   |      +--------+    +--------+      |
   |      |   OK   |    | Cancel |      |
   |      +--------+    +--------+      |
   +------------------------------------+
```

Enter a string, click on "OK"; the dialog disappears, your string is printed in the PRINTER widget. Click on "Cancel", and the dialog disappears.

Click on the "Print time" menu entry and the time is printed to the PRINTER widget. Click on "Quit"; the PANEL vanishes and you're back into HP BASIC.

* Let's see how this magic is done. The first statements in the program simply define some useful variables for handling a dialog:

```
   INTEGER Btn     ! Gets button value from dialog.
   DIM S$[256]     ! Gets string value from dialog.
```

Next, the program defines the size (3/4 of the display) and position (center of the display) of the main PANEL:

```
   Dw=640
   Dh=480
   Pw=Dw*.75
   Ph=Dh*.75
   Px=(Dw-Pw)/2
   Py=(Dh-Ph)/2
```

A 640x480 display resolution is assumed, but you can adjust for a different resolution, by changing the "Dw" and "Dh" variables appropriately.

* The program then creates the PANEL and positions it appropriately:

```
   ASSIGN @Main TO WIDGET "PANEL";SET ("VISIBLE":0)
   CONTROL @Main;SET ("X":Px,"Y":Py,"WIDTH":Pw,"HEIGHT":Ph)
```

Note how a SET statement can be attached to the ASSIGN statement. In this case, we set the PANEL's VISIBLE attribute to 0, to make the PANEL invisible. Why this is done will be explained momentarily.

Once the PANEL is available, we can add the pulldown menu system:

```
   ASSIGN @Menu TO WIDGET "PULLDOWN MENU";PARENT @Main
   ASSIGN @Get_text TO WIDGET "MENU BUTTON";PARENT @Menu
   ASSIGN @Print_time TO WIDGET "MENU BUTTON";PARENT @Menu
   ASSIGN @S TO WIDGET "MENU SEPARATOR";PARENT @Menu
   ASSIGN @Quit TO WIDGET "MENU BUTTON";PARENT @Menu
```

The CONTROL statements associated with these widgets are not listed here, as they merely specify the LABEL string that is shown for the widget. Leaving them out shows the process for building a pulldown menu system more clearly.

The first of these ASSIGN statements creates the PULLDOWN MENU widget. This sets up a menu bar across the top of the PANEL underneath the title bar. But what's this PARENT business? This requires some discussion.

You can generally create widgets without using the PARENT keyword. For example, if you invoke some (arbitrary) ASSIGN statements as follows:

```
   ASSIGN @Main TO WIDGET "PANEL"
   ASSIGN @Label TO WIDGET "LABEL"
   ASSIGN @Toggle TO WIDGET "TOGGLEBUTTON"
```

-- you get three different and totally independent widgets. However, that's not generally what you want when you build a user interface; you want to "glue" other widgets onto another widget, so these "child" widgets will move along with their "parent" widget. You set up this relationship by specifying the PARENT widget every time you ASSIGN one of the child widgets:

```
   ASSIGN @Main TO WIDGET "PANEL"
   ASSIGN @Label TO WIDGET "LABEL";PARENT @Main
   ASSIGN @Toggle TO WIDGET "TOGGLEBUTTON";PARENT @Main
```

You could also make another PANEL a child of @Main and make widgets children of this lower-order PANEL in turn, which leads to the concept of a "level-0" widget.

A level-0 widget is the "top-level" widget. It has no parent, though it may or may not have children. Level-0 widgets have certain privileges that other widgets lack, such as a title bar. In this case, the PANEL is the level-0 widget and all other widgets will be "glued" to it. We glue on a PULLDOWN MENU by specifying the PANEL as the parent.

* But notice that the MENU BUTTONs and MENU SEPARATOR that make up the menu have the PULLDOWN MENU as the parent, not the PANEL. Naturally, since that parent-child relationship is what "glues" the MENU BUTTONs and MENU SEPARATOR to the PULLDOWN MENU, just as the PULLDOWN MENU is "glued" to the PANEL. Since the PULLDOWN MENU is a child of the PANEL, however, the MENU BUTTONs and MENU SEPARATOR are still indirectly children of the PANEL.

Anyway, the ASSIGN statements for the MENU BUTTONs and MENU SEPARATOR populate the pulldown menu system. Note that the order in which the menu elements appear in the pulldown menu is determined by the order in which they are created.

* The pulldown menu system complete (if still inactivated), we now build the PRINTER widget:

```
   COM @Prn
   ASSIGN @Prn TO WIDGET "PRINTER";PARENT @Main
   STATUS @Main;RETURN ("INSIDE WIDTH":Iw,"INSIDE HEIGHT":Ih)
   CONTROL @Prn;SET ("X":0,"Y":0,"WIDTH":Iw,"HEIGHT":Ih)
```

We declare the handle for the printer widget, @Prn, as a common variable so it can be accessed by the subprogram used to print text to it. The widget handle does not have to exist before it is declared as common.

There are, of course, different approaches to accessing the PRINTER widget than calling it via a subprogram that identifies the widget through a common variable. You could, for example, GOSUB to a "local" subroutine to do the print, but that would mean assigning the string to be printed to a variable before doing the GOSUB, which is clumsy. You could also pass the widget handle to the subprogram as a parameter instead of declaring it common, but that would only be useful if there were multiple PRINTER widgets in your user interface, so declaring it as common is more sensible.

The PRINTER widget is created with ASSIGN, using @Main as the parent; no surprises there. The next statement, though, does demand some explanation.

* When we created the PANEL, we first set the display size so we could position the PANEL in the center of the display. However, for the child widgets, the PANEL is their "display": the 0,0 coordinate is in the upper-left corner of the PANEL below the PULLDOWN MENU, and they have to fit in the space left over from the PANEL's title and menu bars.

But how big is this inside space? We know the outside width and height of the PANEL because we set it, but how much space is left? The title and menu bars may have different heights on different displays; they're of no fixed size.

The answer is simple. You use the attributes INSIDE WIDTH and INSIDE HEIGHT, which return the dimensions we want:

```
   STATUS @Main;RETURN ("INSIDE WIDTH":Iw,"INSIDE HEIGHT":Ih)
```

The PRINTER widget can then be sized to these dimensions to fit neatly into the PANEL.

* In general, for most programs you will need to interrogate the main PANEL in this way so that you can place child widgets inside of it. Note that if you are trying to create a child widget and it absolutely refuses to show up, you probably didn't initialize a variable someplace, or you're using the wrong variable name: the widget actually is there, it just has zero width or height. Our feeble human minds cannot see such objects, and so they appear invisible.

* This essentially completes construction of the user interface, so now we're ready to put it into operation. First, we activate the MENU BUTTONs in the pulldown menu system:

```
   ON EVENT @Get_text,"ACTIVATED" GOSUB Get_text
   ON EVENT @Print_time,"ACTIVATED" GOSUB Print_time
   ON EVENT @Quit,"ACTIVATED" GOTO Finis
```

This sets up events to call routines to execute the program actions (or, in the case of @Quit, jump to code to exit the program). The BPlus user interface is essentially inert until you arm an event so it can do something. The program can generate output to various widgets, but cannot detect user inputs. In fact, you don't even get a mouse cursor until you arm an event. (Note that similarly you can't get rid of the cursor except by shutting off all the events!)

* Finally, we make the PANEL visible, call the PRINTER output routine to put a prompt on the display, and then loop, waiting for a MENU BUTTON event:

```
   CONTROL @Main;SET ("VISIBLE":1)
   CALL Printit("Waiting ...")
   LOOP
   END LOOP
```

Recall that when we began to build the user interface, we created the PANEL and set it to invisible in the same command; the very last thing we did in building the user interface was to set the PANEL to VISIBLE again.

The reason for this is that if you build a user interface with various widgets and don't set the parent PANEL to invisible, the different widgets magically appear and dance around as their attributes are changed. This is fun to watch for a little while, but it soon gets annoying.

But if you make the parent PANEL invisible, all the child widgets remain invisible, too, so all you have to do is make the parent PANEL invisible when you create it and then make it visible only after you've loaded it up with widgets, and the user interface neatly pops up on the display.

* The rest of the program, whose various elements are called by the MENU BUTTONs (directly or indirectly), is something of an anticlimax.

Putting the last first ... if the user selects the @Quit MENU BUTTON, the program jumps to the exit code, which wipes out the BPlus user interface and ends the program.

```
   Finis: !
     ASSIGN @Main TO *
     STOP
```

Closing @Main not only kills off the PANEL, it also kills off any child widgets on the PANEL.

The @Get_text and @Print_time MENU BUTTONs call two simple routines:

```
 Get_text: !
   DIALOG "STRING","Please input a string:",Btn;RETURN ("VALUE":S$)
   IF Btn=0 THEN CALL Printit(S$)
   RETURN

 Print_time: !
   CALL Printit(TIME$(TIMEDATE))
   RETURN
```

Notice how the "Get_text" routine checks the "Btn" value to see if the user has clicked on the dialog's "OK" or "Cancel" button, and only prints the value if the user has clicked on "OK".

Both these routines call the "Printit" subroutine, which is also very simple:

```
   SUB Printit(S$)
     COM @Prn
     CONTROL @Prn;SET ("APPEND TEXT":S$)
   SUBEND
```

The APPEND TEXT attribute simply prints the string to the PRINTER widget.

* This completes a version of TEMPLATE that provides the basic required functionality. More can easily be done with it, however.

[4.0] BPLUS (4): Quick Tutor -- Advanced TEMPLATE

v2.4 / 01 feb 99 / greg goebel

* Our first-pass version of TEMPLATE does the job, but only in the most minimal way possible. We can easily add refinements.

[4.1] DETAILS & REFINEMENTS

* The first is virtually a requirement in practice. In our previous program we assigned the display dimensions as follows:

```
   Dw=640
   Dh=480
```

While it is simple to adapt the display resolution by changing these constants, it is also inconvenient; it is better to design the program to be smart enough to detect the display resolution on its own.

HP BASIC includes a statement called GESCAPE that provides various types of information about and control over the host's display system; an operation code parameter determines the type of information obtained. Operation code 3 gives the bounds of the display in pixels, and it can be used to obtain the display resolution as follows:

```
   INTEGER D(1:4)
   ...
   GESCAPE CRT,3;D(*)
   Dw=D(3)-D(1)
   Dh=D(4)-D(2)
```

This sets the "Dw" and "Dh" variables to the full width and height of the display in pixels. (Actually, to be perfectly precise, you need to add 1 to "Dw" and "Dh" ... you can if you want.) The TEMPLATE program uses the full display, including the lines normally reserved for the softkeys, the text input line, and the runlight, so before you actually set up the main PANEL you probably want to turn these things off:

```
   KEY LABELS OFF          ! Turn off key labels.
   RUNLIGHT OFF            ! Turn off "run light".
   STATUS CRT,10;Cursor    ! Save cursor code value.
   CONTROL CRT,10;0        ! Turn off cursor.
```

Note that there can be different cursor styles under RMB/UX, which is why you save the value before turning off the cursor. When you're done with the program, you can turn these elements back on again with:

```
   KEY LABELS ON
   RUNLIGHT ON
   CONTROL CRT,10;Cursor
```

One of the problems with this approach is that if your program bombs, you're left without a cursor, which can be a nuisance; you have to reset HP BASIC
to get it back.

* There is an alternative approach towards display handling which is simpler than having BPlus take over the whole display: you simply reserve the BPlus user interface to the PRINT section of the display and leave the softkeys and input line untouched. This approach is more flexible in that it leaves the option of using both conventional HP BASIC user-input techniques and BPlus widgets in the same program -- but at the expense of a certain amount of clutter.

You can determine the number of text lines the display supports by reading CRT status register 13 as follows:

```
   STATUS CRT,13;Nlines
```

-- where "Nlines" is an arbitrary variable name. Since the softkeys and so on take up the bottom 7 lines of the display, you can determine the height of the remainder with:

```
   Dh=(D(4)-D(2))*((Nlines-7)/Nlines)
```

In the current case, we'll let TEMPLATE use the whole display, but it's nice to have alternatives, and you'll see this trick in later chapters.

* We might as well give the PANEL an intelligent name. The TITLE on a widget normally defaults to the widget name ("PANEL" in this case) but we can set it to what we like:

```
   CONTROL @Main;SET ("TITLE":"This Space For Rent")
```

-- which yields:

```
+--------------------------------------------------------+---+
|                    This Space For Rent                 | X |
+--------------------------------------------------------+---+
| Menu                                                       |
```

* The PANEL contains a button in the upper-right corner; if you click on that with the mouse, the PANEL will be expanded (MAXIMIZEd) over the entire display. If you click on it again, it will shrink back to its original size.

Similarly, the PANEL has a border around it (impossible to show in the illustrations here); if you grab a side or corner with the mouse, you can stretch (RESIZE) the PANEL to a different shape. It is also possible to use the mouse to move the PANEL around on the display by grabbing the PANEL's title bar with the mouse cursor.

Moving the PANEL around on the display is harmless in this example and there's no reason to prevent it. However, we're populating this PANEL with another widget (the PRINTER widget), and child widgets won't, by default, change size when you MAXIMIZE or RESIZE the PANEL (though the PULLDOWN MENU is an exception to that rule, as it is closely integrated into the PANEL).

This can lead to a very muddled user interface. The brute-force way of dealing with this is to make it impossible for the user to change the PANEL's size and shape, by setting the MAXIMIZABLE and RESIZABLE attributes to 0:

```
   CONTROL @Main;SET ("MAXIMIZABLE":0,"RESIZABLE":0)
```

Note that when you clear these attributes the MAXIMIZABLE button and the border disappear; they're not useful any more. There is also a MOVABLE attribute that can be set to 0 to lock the user interface in place, but there is absolutely no reason to require that here.

* Note also that another attribute of this type was added in BPlus 2.0: MINIMIZABLE. If you enable this attribute -- it's off by default -- another box appears on the title bar, immediately to the left of the MAXIMIZE button; click on this MINIMIZE button, and the PANEL will disappear and be listed in the Minimized Windows application.

This is an extremely useful feature if you want to make a BPlus application that has multiple user interfaces; it allows the user to conveniently select any of the set that he or she wants, and just as conveniently hide them again when they are not needed. However, TEMPLATE has only one user interface, so we won't worry about this more here.

* It's really taking the easy way out to set MAXIMIZABLE and RESIZABLE to 0; TEMPLATE is supposed to be a general-purpose program, and it would be nice to allow the user to change the shape or size of the user interface. This was difficult in the first version of BPlus, but the second release

offered a new feature to help do the job: the SIZE CONTROL attribute.

If you set SIZE CONTROL to the value "RESIZE CHILDREN":

```
   CONTROL @Main;SET ("SIZE CONTROL":"RESIZE CHILDREN")
```

-- then all child widgets will be automatically resized when you change the size of the PANEL. (You can also set it to SCROLLABLE, which allows you to display elements bigger than your main PANEL and scroll around in them, but, again, we won't worry about that here.)

* There's a few other minor things we can do to improve our program. For example, you could set a color scheme of your preference on the PRINTER widget:

```
   CONTROL @Prn;SET ("BACKGROUND":Blue,"PEN":White)
```

This sets the PRINTER widget text to white text on a blue background, a scheme that at least I find pleasing. Of course, to invoke the statement above, you need to have defined "constants" giving the PEN numbers for the default colors:

```
   INTEGER Black,White,Red,Yellow,Green,Cyan,Blue,Magenta
   DATA 0,1,2,3,4,5,6,7
   INTEGER Black,White,Red,Yellow,Green,Cyan,Blue,Magenta
```

This is so useful that you should have this code in any BPlus program.

You could also specify a larger font for the PRINTER widget to make the text more visible, if you like:

```
   CONTROL @Prn;SET ("FONT":"18 BY 30")
```

This defines a specific font provided with BPlus. I see no particular need to do that in our current TEMPLATE program, but you might want to use a bigger font if you want to make your text more visible.

* The initial TEMPLATE had a pull-down menu system, which is almost a requirement for any program that has a number of command inputs; but if you have a very simple program that only needs, say, a "Quit" button, a full menu system is somewhat cumbersome.

The second version of BPlus incorporated a neat improvement that allows you to set up a simple menu system with a minimum of fuss on any level-0 widget: the SYSTEM MENU.

The SYSTEM MENU looks suspiciously like a widget, but it's not -- it's an attribute of a level-0 widget, much like a title bar. If you set the SYSTEM

MENU attribute for, say, a PANEL, to a string value like "Quit":

```
   CONTROL @Main; SET("SYSTEM MENU":"Quit")
```

-- then when the PANEL appears on the display, it has a "toaster box" in its upper left corner:

```
   +---+---------------------------------------------------+---+
   | = |                This Space For Rent                | X |
   +---+---------------------------------------------------+---+
   | Menu                                                      |
```

This is known as a "toaster box" because it is said that it looks like a toaster, as seen from the top ... anyway, click on the toaster box with a mouse and you get a "toaster menu":

```
   +---+------
   | = |
   +---+--+---
   | Quit |
   +------+
   |
```

You can set an event on the parent widget to trap the activation of this toaster menu; the event, not surprisingly, is the SYSTEM MENU event:

```
   ON EVENT @Main,"SYSTEM MENU" GOTO Finis
```

You can set up multiple menu entries by setting the SYSTEM MENU attribute with a string array, instead of merely a string, and when the SYSTEM MENU event occurs, you can figure out which entry in the menu caused the event by reading the string-array index from the parent widget with the SYSTEM MENU EVENT attribute. (Note that the index returned always starts at an index of 0, even if the string array has a base index of 1.) For example:

```
   DIM M$(0:2)[50]
   DATA "Menu Entry 1","Menu Entry 2","Menu Entry 3"
   READ M$(*)
   CONTROL @Main;SET ("SYSTEM MENU":M$(*))
   ON EVENT @Main,"SYSTEM MENU" GOSUB Dispmenu
   ...
 Dispmenu: !
   STATUS @Main;RETURN ("SYSTEM MENU EVENT":N)
   DISP "Menu entry: "&M$(N)
   RETURN
```

However, trying to do complicated things with a SYSTEM MENU is pointless, since if you are trying to anything but build a simple single list of items, a conventional BPlus menu system works a lot better. (A SYSTEM MENU does have the additional virtue that it can be used with any level-0 widget; a regular menu system only works with a PANEL. This allows you in some cases to set up a surprisingly sophisticated user interface with only one widget.)

But a small SYSTEM MENU is very handy -- you'll see them in a lot of the demos -- and so one is included in the improved TEMPLATE as an alternative to the conventional menu system. All it does is duplicate the "Quit" MENU BUTTON, but the overhead is so low (two lines) that it's no nuisance to have it.

* You could also customize TEMPLATE with your own items, program resources that you find useful. For example, I often write test programs that use random numbers to generate dummy data, so I need to initialize the random-number generator from the time of day in a way that varies greatly between program runs:

```
   RANDOMIZE INT(FRACT(TIMEDATE)*10^7)
```

A random number generator produces pseudo-random numbers from a seed value; if you don't change the seed value, you get the same random numbers. Using the time of day as a seed allows you to get unpredictable sequences, but you get the most variation if you use widely-varying seeds of large value. In the statement above, HP BASIC takes the fractional-second part of the TIMEDATE count and multiplies it by a large value, fulfilling these conditions.

There are also certain variables that I often use in programs; "Btn" and "S$" are examples of such handy variables I included in the first version of TEMPLATE. I could add other variables to get status from control registers; store a interface select code or device address; store an error code or timeout value; or just store some arbitrary integer or real value:

```
   INTEGER Btn,Sts,Isc,Addr,Err,Timeoutval,N
   REAL R
   DIM S$[256]
```

Re-using the same variable names in different programs makes writing new programs faster and simpler.

Putting all these things together gives our improved version of TEMPLATE.

Program Example: xtemple.rmb

[4.2] YET MORE TEMPLATE FEATURES

* There are of course any number of other improvements you can add to TEMPLATE to suit your needs.

For example, when you click on "Quit", the program ends and there's nothing you can do about it. In some cases, that is not desirable; you want the user to be able to reconsider exiting the program before actually doing it. That can be easily done by changing the exit code from:

```
   ON EVENT @Quit,"ACTIVATED" GOTO Finis
   ...
 Finis: !
   ASSIGN @Main TO *
   STOP
```

-- to:

```
   ON EVENT @Quit,"ACTIVATED" GOSUB Finis
   ...
 Finis: !
   DIALOG "QUESTION","Exit program?",Btn
   IF Btn=1 THEN RETURN
   ASSIGN @Main TO *
   STOP
```

This pops up a dialog to ask the user to confirm program exit; if the user clicks the "No" button, the program continues, if the user clicks the "Yes" button, the program dies.

* Another very useful thing is a custom error handler. When an error occurs in a program, the program crashes; the user may want the options of retrying the statement that caused the error, continuing past that statement, or exiting the program gracefully (so the program can, for example, restore the display to some intelligible state instead of leaving it cluttered with program debris).

This can be done by setting an event to an error handler:

```
   ON ERROR GOSUB Errtrap
```

The error handler can bring up a dialog with customized buttons to offer the user the appropriate options:

```
   +---------------------------------------------+---+
   |                     ERROR                   | X |
   +---------------------------------------------+---+
   |                                                 |
   |  STOP     <print error message string here>    |
   |                                                 |
   +-------------------------------------------------+
   |  +----------+    +----------+    +----------+  |
   |  |  Retry   |    | Continue |    |   Quit   |  |
   |  +----------+    +----------+    +----------+  |
   +-------------------------------------------------+
```

This is a perfect example of how dialogs can be used to implement low-level decision making. The handler routine would look something like this:

```
 Errtrap: !
   B$(1)="Retry"            ! Set up custom buttons.
   B$(2)="Continue"
   B$(3)="Quit"
   DIALOG "ERROR",ERRM$,Btn;SET ("DIALOG BUTTONS":B$(*))
   SELECT Btn
   CASE 0                   ! Retry.
     RETURN
   CASE 1                   ! Skip offending statement, continue.
     ERROR RETURN
   CASE 2                   ! Quit program.
     ASSIGN @Main to *
   STOP
   END SELECT
   RETURN
```

Note that the ERRM$ returns the error number, the line number where the error occurred, and the error description. Note also that you probably want to set up the custom buttons in the main program -- I do it in the subroutine here just to show what needs to be done.

* My early attempts at such an error handler led to a nasty trap that you may run into if you are not careful. The first one only allowed a RETURN to the error condition ... and when an error occurred, the event called the error handler and popped up the dialog. When I clicked on the dialog button, the error handler returned to the error condition ... and called the error handler, which popped up the dialog again ... and so on and so on -- and I had to reset HP BASIC to escape!

[4.3] TEMPLATE & BEYOND

* TEMPLATE in its completion is not merely a good example program, but is useful for building almost any BPlus program you want. And its design follows useful principles for building BPlus user interfaces:

  * Design your program as a "shell" that presents an interface to the user, with a set of command inputs such as PUSHBUTTONS, MENU BUTTONS, and so on. Each of these command inputs generates an event to call a handler routine. This "event-driven" architecture is simple to grasp and extend.

  * Only use a single level-0 widget, a main PANEL, in a user interface. Multiple level-0 widgets are difficult to control.

  * Don't use any more widgets than you have to. The more widgets you have, the more complicated (hard to use) your user interface, and the

harder your program will be to design and debug.

* Never use a PUSHBUTTON or LABEL or other widget when you can use pulldown menu widgets to do the same job. Pulldown menu widgets are easy to implement, intuitive to use, and flexible -- you can, for example, have the MENU BUTTON for a particular setting display the setting itself -- say, "Oven = ON" -- and dynamically change when you select the MENU BUTTON -- say, to "Oven = OFF".

  You can simlarly integrate numeric values into the MENU BUTTON to provide feedback on the current setting.

* If you have a very simple program that doesn't have much in the way of menu requirements, just use a SYSTEM MENU and save even more trouble.

* Use dialogs as much as you can. Use dialogs to implement low-level decisions, display data, and perform other functions that do not need to be permanently available on the user interface; dialogs use less overhead than widgets, and as they are transitory eliminate a lot of clutter. Use of dialogs with customized buttons is highly recommended.

There are some features that TEMPLATE does not demonstrate, however. Most importantly, excluding the pulldown menu system, it only had one distinct widget inside the PANEL, the PRINTER widget. In many cases there will be more widgets inside the PANEL.

If you're not careful, setting up these widgets can become a time-consuming job. Besides minimizing the number of widgets, there are two things you can do to make the child widgets easier to handle:

* Use standardized widget sizes and lay the widgets out on as regular a grid as possible. This minimizes the number of variables you need to define the layout and makes the system easier to grasp (with the drawback that if you completely redesign the interface, you generally have to redefine your variables).

  If this is not possible or desirable, at least give the variables a consistent and clear naming convention.

* Whenever possible, try to define these variables in such a way that adjusting one automatically adjusts all the others; otherwise you may end up tweaking widgets back and forth until hell freezes over. This is a subtle trick, but it will be illustrated in later chapters.

One last rule, to borrow from George Orwell: break any of the above rules rather than do something stupid. But first make sure you know the reason why.

[4.4] FINE POINTS: KEYBOARD INTERACTION, TAB GROUPS, WIDGET HELP

* For lack of a better place to discuss these topics ... when you set up a BPlus user interface, you normally interact with it using a mouse, clicking on buttons and moving sliders and the like. But suppose you don't have a mouse or don't like using one? How can you get the keyboard to do what you want?

Let's look at a typical BPlus user interface:

```
+-----------------------------+
| Typical BPlus User Interface |
+-----------------------------+
| Menu                        |
+-----------------------------+
|  +----------+  +----------+  |
|  | BUTTON 1 |  | BUTTON 2 |  |
|  +----------+  +----------+  |
|  +----------+  +----------+  |
|  | BUTTON 3 |  | BUTTON 4 |  |
|  +----------+  +----------+  |
|  +-----------------------+  |
|  | <                   > |  |
|  +-----------------------+  |
+-----------------------------+
```

When the user interface comes up, you can press Alt-Tab (on a PC keyboard; Extend Character-Tab on a workstation keyboard); the arrow will then pop up to the Menu and mark it with a square cursor.

Once this is done, you can use the Tab key to cycle through all the input widgets in the user interface: Tab, and the cursor goes to BUTTON 1; Tab again, and you go to BUTTON 2; then BUTTON 3; BUTTON 4; then the the SCROLLBAR at the bottom; Tab again, you go back to the TOGGLEBUTTON. You can press Shift-Tab to move in the reverse direction ... by the way, the order in which you Tab through the widgets is not determined by their physical layout, but by the sequence in which they were created in the program.

When you are on the PUSHBUTTONs, you can activate them by banging on the space bar. When you are on the SCROLLBAR, you can move though the MINOR INCREMENT using the arrow keys, and through the MAJOR INCREMENT by using the Page Up and Page Down keys (on a PC keyboard; Prev and Next on a workstation keyboard).

If you want to get to the menu, just press Alt-Tab again; keep Alt held down and you can navigate through the menu with the arrow keys, and use Enter to select the menu entry.

If you have multiple independent widgets on your display (not a great idea), you can cycle through them by pressing Alt-Tab.

* Now that you know about keyboard interaction with BPlus, it's much easier to explain the concept of a tab group.

Most input widgets have an attribute called TAB STOP, which is by default set to 1. When you cycle through your user interface with Tab or Shift-Tab, the box cursor will stop at an input widget if its TAB STOP attribute is 1. Which begs the next questions: what happens if TAB STOP is 0, and why care?

The best way to answer this is with an example; let's consider a hypothetical user interface that consists of 4 rows of 3 PUSHBUTTONs. Ignoring messy details like coordinates, size, and labels, suppose we write code like this:

```
   ASSIGN @Br1c1 TO WIDGET "PUSHBUTTON";PARENT @Main
   ASSIGN @Br1c2 TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("TAB STOP":0)
   ASSIGN @Br1c3 TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("TAB STOP":0)
   !
   ASSIGN @Br2c1 TO WIDGET "PUSHBUTTON";PARENT @Main
   ASSIGN @Br2c2 TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("TAB STOP":0)
   ASSIGN @Br2c3 TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("TAB STOP":0)
   !
   ASSIGN @Br3c1 TO WIDGET "PUSHBUTTON";PARENT @Main
   ASSIGN @Br3c2 TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("TAB STOP":0)
   ASSIGN @Br3c3 TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("TAB STOP":0)
   !
   ASSIGN @Br4c1 TO WIDGET "PUSHBUTTON";PARENT @Main
   ASSIGN @Br4c2 TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("TAB STOP":0)
   ASSIGN @Br4c3 TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("TAB STOP":0)
```

This gives a user interface, that looks, say, like this:

```
   +---------------------------------+
   |          Tab Group Demo         |
   +---------------------------------+
   | +-------+ +-------+ +--------+ |
   | | R1_C1 | | R1_C2 | | R1_C3  | |
   | +-------+ +-------+ +--------+ |
   | +-------+ +-------+ +--------+ |
   | | R2_C1 | | R2_C2 | | R2_C3  | |
   | +-------+ +-------+ +--------+ |
   | +-------+ +-------+ +--------+ |
   | | R3_C1 | | R3_C2 | | R3_C3  | |
   | +-------+ +-------+ +--------+ |
   | +-------+ +-------+ +--------+ |
   | | R4_C1 | | R4_C2 | | R4_C3  | |
   | +-------+ +-------+ +--------+ |
   +---------------------------------+
```

-- where the PUSHBUTTON label corresponds to its widget handle. Note that all the PUSHBUTTONs on the left side do not set TAB STOP to 0 and so are left with TAB STOP defaulted to 1; the middle and right columns have TAB STOP set to 0.

If you use the keyboard to cycle through this user interface, you will notice that as you press Tab or Shift-Tab, you only cycle through the PUSHBUTTONs on the left. If you want to get to a particular PUSHBUTTON in the middle or right columns, you Tab to the appropriate PUSHBUTTON in the left column in the same row, and then use the cursor keys to cycle through the row.

Each row in this case constitutes a tab group. Physical layout in the tab group has nothing to do with the order of the tab group; what defines the order of the tab group is the sequence in which the input widgets are created. If you ASSIGN one widget with a TAB STOP of 1, then any following widgets with a TAB STOP of 0 are part of that tab group, until you ASSIGN another widget with a TAB STOP of 1.

* Why bother with this? In this example, it's not obvious what good this is; but suppose you have a complicated user interface with lots of controls? You could group the controls into separate clusters, each set up as a tab group. You could step from cluster to cluster with Tab and then maneuver within them using the cursor keys.

And when you learn about RADIOBUTTON widgets, you'll find out that tab groups have another interesting use.

* BPlus 2.0 offered an interesting refinement over the first release of BPlus, in that it allowed all widgets (except for things like PANEL SEPARATORs) to be hooked up to the online Help utility through the attributes HELP FILE and HELP TOPIC. If you press the f1 softkey or the secondary mouse button when on top of a widget, the Help utility pops up set to the designated HELP FILE and HELP TOPIC. (By default, all you'll get is a help topic on the widget itself.)

This Help feature can be controlled by options in the Bplus CONFIG file. These are set in the "@ context help" section and perform the following functions:

* f1_is_help: Set or clear f1 key for help access.
* rht_ms_btn_is_help: Set or clear right mouse button for help access.
* provide_defaults: Set or clear providing default widget HELP TOPIC.

More will be said about this in the appendix on the HELPX language.

[5.0] BPLUS (5): Quick Tutor -- Widget & Attribute Summary

v2.4 / 01 feb 99 / greg goebel

* Now that we know what widgets are all about, it's time for a tour of the entire set. This section provides short summaries of later chapters; if you want to learn more about a widget, please consult the appropriate chapter. (Some of these widgets were adequately introduced earlier, but for thoroughness I will review the complete set.)

The first section of this chapter reviews the widgets; the second section outlines attributes which are commonly used with all or nearly all dialogs and widgets. Attributes specific to the individual kinds of widgets are discussed in the chapters that describe the widgets.

[5.1] A WIDGET SUMMARY

* The widgets supported by BPlus are listed below.

* PANEL & PANEL SEPARATOR:

The PANEL is just a simple rectangle that acts as a template to hang other widgets on. ASSIGNing a PANEL SEPARATOR to a PANEL allows you to draw a line across a PANEL, and you can set the ORIENTATION of this line to HORIZONTAL or VERTICAL.

Advanced features added in BPlus 2.0 allow the PANEL to automatically resize its child widgets when the PANEL is modified; to allow scrolling of a user interface that is too big to fit into the PANEL; and to allow "tiling" (like MS-Windows) of a background bitmap pattern.

* LABEL:

The LABEL allows you to display a VALUE of some sort -- either a number, a complex number, or a string, it's not picky about its inputs:

```
   ASSIGN @Lbl TO WIDGET "LABEL";PARENT @Main
   CONTROL @Lbl;SET ("VALUE":someval)
```

You can also set it to display a given number of ROWs and COLUMNs of text:

```
   CONTROL @Lbl;SET ("ROWS":3,"COLUMNS":15)
```

* PUSHBUTTON, TOGGLEBUTTON, RADIOBUTTON:

The PUSHBUTTON is a simple input device, which allows a mouse click to cause an "ACTIVATED" event to control program flow.

```
   ASSIGN @Btn TO WIDGET "PUSHBUTTON";PARENT @Main,SET ("LABEL":"Push Me")
   ON EVENT @Btn,"ACTIVATED" GOSUB Handler
```

BPlus 2.0 added a useful feature to the PUSHBUTTON: you can give it a string array as a LABEL value, and then the user will cycle through the LABELs as it is clicked with the mouse:

```
   DIM L$(0:2)[16]
   DATA "Code Red","Code Yellow","Code Green"
   READ L$(*)
   CONTROL @Btn;SET ("LABELS":L$(*),"STATES":2)
```

The TOGGLEBUTTON is similar to the PUSHBUTTON, except that it has a VALUE attribute associated with it; each time you click on the TOGGLEBUTTON, the VALUE toggles between 0 and 1 (0 is the default). The TOGGLEBUTTON also has an indicator on it, a little box, that switches from clear when the VALUE is 0 to black when the VALUE is 1. Like the PUSHBUTTON widget, the TOGGLEBUTTON has a single event, but it is named CHANGED, not ACTIVATED:

```
   ASSIGN @Toggle TO WIDGET "TOGGLEBUTTON";PARENT @Main
   CONTROL @Toggle;SET ("LABEL":"Click Me")
   ON EVENT @Toggle,"CHANGED" GOSUB Handler
   ...
 Handler: STATUS @Toggle;RETURN ("VALUE":Tval)
   DISP Tval
   RETURN
```

The TOGGLEBUTTON is useful as an on-off switch.

The RADIOBUTTON functions identically to the TOGGLEBUTTON, with two exceptions: the indicator is a diamond, not a little box; and if you set up a tab group of such widgets:

```
   ASSIGN @Radio1 TO WIDGET "RADIOBUTTON";PARENT @Main
   ASSIGN @Radio2 TO WIDGET "RADIOBUTTON";PARENT @Main,SET ("TAB STOP":0)
   ASSIGN @Radio3 TO WIDGET "RADIOBUTTON";PARENT @Main,SET ("TAB STOP":0)
   ASSIGN @Radio4 TO WIDGET "RADIOBUTTON";PARENT @Main,SET ("TAB STOP":0)
```

-- then you will only be able to set the value of one of these RADIOBUTTONs in this tab group to 1 at a time; setting one clears any other that is set. (You will get a CHANGED event for both the RADIOBUTTON that is set and the RADIOBUTTON that is cleared.) The RADIOBUTTON is useful for selecting from a list of mutually-exclusive choices.

* NUMBER, KEYPAD:

The NUMBER widget is very similar to a STRING widget, except that it only allows the user to input a numeric value. You can specify:

* The numeric FORMAT that it will accept (REAL, SHORT INTEGER, LONG INTEGER, BINARY, OCTAL, or HEX).
* The REAL NOTATION for REAL inputs (FIXED, SCIENTIFIC, or ENGINEERING).
* The REAL RESOLUTION of floating-point inputs.
* The MINIMUM or MAXIMUM allowed inputs.
* The round-off value of the input (INCREMENT).

Like the STRING widget, you can trap DONE, KEYSTROKE, and RETURN events; you can also trap an INVALID NUMBER event for a bogus user input.

The KEYPAD widget has the same functionality as the NUMBER widget, but displays a numeric keypad. For example, executing:

```
   ASSIGN @Key TO WIDGET "KEYPAD"
```

-- yields the widget:

```
   +-------------------+---+
   |        KEYPAD     | X |
   +-------------------+---+
   |                       |
   +-----+-----+-----+-----+
   |  A  |  B  |  C  |  D  |
   +-----+-----+-----+-----+
   |  7  |  8  |  9  |  E  |
   +-----+-----+-----+-----+
   |  4  |  5  |  6  |  F  |
   +-----+-----+-----+-----+
   |  1  |  2  |  3  | <-- |
   +-----+-----+-----+-----+
   |  -  |  0  |  .  | --> |
   +-----+-----+-----+-----+
   | CLR | DEL | INS | ENT |
   +-----+-----+-----+-----+
```

* STRING:

The STRING input widget puts up a field into which the user can type a text string; you can set an event on each key (KEYSTROKE), when the user presses the Enter key (RETURN), or when the user moves focus away from the STRING widget (DONE).

The STRING widget was greatly enhanced in the second version of BPlus to provide advanced attributes that allow the STRING widget to be used to build simple text-editing tools, with features such as cutting and pasting, text searching, and file-I/O. These features are too complicated to discuss in this short space, however, and are detailed in a later chapter.

* COMBO:

The COMBO widget is similar to the COMBO dialog; it combines the features of the STRING widget with the ability to enter from a predefined list:

```
   ASSIGN @Combo TO WIDGET "COMBO":SET ("ITEMS":L$(*))
```

You can use a SELECTION event to trap a selection from the list and use the SELECTION attribute to get the list index. You can also use a RETURN event to trap an entry in the string-input part of the widget, and then use the TEXT attribute to read the data.

```
   ON EVENT @Combo,"SELECTION" GOSUB Handler
   ON EVENT @Combo,"RETURN" GOSUB Handler
   ...
 Handler: STATUS @Combo;RETURN ("TEXT":T$)
   DISP T$
   RETURN
```

* LIST:

The LIST widget acts similarly to the LIST dialog: it allows you to display a list of ITEMS (derived from a string array) and then click on an entry in the list to get back the list item. As with the LIST dialog, you can also set a MULTISELECT attribute, in which case the LIST returns values of 0 or 1 to a numeric array that matches the string array, with the 1s matching the items that are set.

For example, if you execute:

```
   DIM L$(0:4)[32]
   ASSIGN @List TO WIDGET "LIST";SET ("COLUMNS":8,"ROWS":3)
   DATA "ITEM 1","ITEM 2","ITEM 3","ITEM 4","ITEM 5"
   READ L$(*)
   CONTROL @List;SET ("ITEMS":L$(*))
   ON EVENT @List,"SELECTION" GOSUB Handler
```

-- you get the widget:

```
   +----+---+
   |  LI| X |
   +----+---+
   | ITEM 1 |
   | ITEM 2 |
   | ITEM 3 |
   | ITEM 4 |
   | ITEM 5 |
   +--------+
```

If your "Handler" routine has the statements:

```
   STATUS @List;RETURN ("SELECTION":Nval)
   DISP Nval
```

-- then if you click on "ITEM 2", you would get a "1". (Clicking on "ITEM 1" gives a "0".) If you had set:

```
   CONTROL @List;SET ("MULTISELECT":1)
```

-- then you would change these "Handler" statements to:

```
   STATUS @List;RETURN ("SELECTION":A(*))
   DISP A(*)
```

Then clicking on "ITEM 2" would give:

```
   0       1       0      0      0
```

* FILE:

The FILE widget is very similar to the FILE dialog; if you execute:

```
   ASSIGN @F TO WIDGET "FILE"
```

-- you get the widget:

```
+---+-------------------------------+---+
| = |       File Widget Demo        | X |
+---+-------------------------------+---+
|                                       |
| Current directory:      Directories:  |
| +---------------------+  +--------+-+ |
| |                     |  |        | | |
| +---------------------+  +--------+-+ |
|                                       |
| File wildcard:          Files:        |
| +---------------------+  +--------+-+ |
| |                     |  |        | | |
| +---------------------+  +--------+-+ |
|                                       |
| File selection:                       |
| +---------------------------------+ |
| |                                 | |
| +---------------------------------+ |
+---------------------------------------+
```

As with the FILE dialog, there are SELECTION, PATTERN, and DIRECTORY attributes to allow you to set the file, wildcard pattern, and directory. There is also a SELECTION event that occurs when a file has been selected.

* PRINTER:

The PRINTER widget simply provides a printing output device; you can list a line of text on the PRINTER widget with the APPEND TEXT attribute:

```
   CONTROL @Prn;SET ("APPEND TEXT":Str$)
```

You can also dump an entire array of strings at one time using APPEND TEXT.

You can print a blank line by using APPEND TEXT with a null string. You can clear the PRINTER widget and start displaying new text by using the TEXT attribute instead of the APPEND TEXT attribute.

Other attributes were added in BPlus 2.0 to allow allow selection of any line in the PRINTER display (CURRENT LINE); to highlight the current line and allow it to be set with a mouse (HIGHLIGHT CURRENT); to read the current line and lines following into a string or string array (CURRENT TEXT); and to INSERT LINES or DELETE LINES relative to the current line.

As noted earlier, you cannot perform a PRINTER IS to a PRINTER widget.

* HPGL VIEW, BITMAP:

The HPGL VIEW widget accepts a text file full of HPGL plotter commands and displays the plot. You give it the file name using the HPGL FILE attribute:

```
   CONTROL @Graf;SET ("HPGL FILE":"Picture")
```

You can set a RETAIN RASTER attribute to keep the HPGL VIEW widget from completely redrawing its graphics every time you disturb it; when RETAIN RASTER is set, BPlus takes a "snapshot" of the graphics and retains it for repainting later.

Note that only the most fundamental HPGL commands are implemented -- for example, area fills don't work. Also note that you cannot use an HPGL VIEW widget as the PLOTTER IS device.

The BITMAP widget similarly allows you to read and display either an X11 .XWD or MS-Windows .BMP bitmapped-graphics file:

```
   ASSIGN @Bitz TO WIDGET "BITMAP";SET ("AUTO SIZE":1)
   CONTROL @Bitz;SET ("BITMAP FILE":"CANDY.BMP")
```

The AUTO SIZE attribute specifies that the widget resize itself to the bitmap. Note that you can trap a MOUSE CLICKED event from the BITMAP widget; this event is caused by clicking on a pixel in the bitmap. You can then determine the coordinates of the pixel with the MOUSE CLICK attribute.

The BITMAP widget can also be used to grab a BPlus user interface image, or a portion of one, and store it in a bitmap graphics file.

* SLIDER, SCROLLBAR:

The SLIDER is a numeric-input widget; create a SLIDER:

```
   ASSIGN @Slider TO WIDGET "SLIDER";PARENT @Main
   ON EVENT @Slider,"DONE" GOSUB Handler
   ...
 Handler: STATUS @Slider;RETURN ("VALUE":Slideval)
   DISP Slideval
   RETURN
```

-- and you get a device that looks like the slider on a piece of audio equipment:

```
   +-----+---+
   |  SLI| X |
   +-----+---+
   | [ 100 ] |
   |    ^    |
   |    |    |
   |    |    |
   | [ 35  ] |
   |    |    |
   |    v    |
   | [  1  ] |
   +---------+
```

If you click on the arrows at the ends, the slidebar moves in that direction by an increment of 1; if you click on the trough between the slidebar and an arrow, the slidebar will be moved in that direction by an increment of 10.
These define the MINOR INCREMENT and MAJOR INCREMENT attributes, and you can change them if you like; you can also set an attribute called DIRECT MOVE that disables MAJOR INCREMENT, and has the effect that if you click on the trough, the slidebar will move immediately to that point.

You can also change the MINIMUM and MAXIMUM limits, set the SLIDER to a LOGARITHMIC-scaled operation, and change the ORIENTATION from the default VERTICAL to HORIZONTAL -- as well as make many other cosmetic changes.

The SCROLLBAR is a stripped-down version of the SLIDER; if you execute:

```
   ASSIGN @Scroll TO WIDGET "SCROLLBAR";PARENT @Main
   CONTROL @Scroll;SET ("ORIENTATION":"HORIZONTAL")
```

-- gives a device of the form:

```
   +-----------------------+
   |<   [ ]              >|
   +-----------------------+
```

This works exactly the same as the SLIDER, except it lacks limits and VALUE boxes, and cannot be set to LOGARITHMIC mode. (When vertical, the MAXIMUM limit is at the bottom, not the top.)

* CLOCK:

The CLOCK widget displays a time-of-day clock whose TYPE can be set to ANALOG or DIGITAL or MIXED (both). You can set an ALARM TIME and then trap an ALARM event when the CLOCK reaches that time.

You can also set the CLOCK TYPE to a TIMER mode, and operate it in an UP or DOWN counting mode (as set by the TIMER DIRECTION attribute), and in a cyclical mode (as set by the TIMER REPEAT); you can also trap a TIMER event in this mode. (Note that in TIMER mode you only get a digital display.)

* BAR:

The BAR widget defines a bar indicator. You can set the MAXIMUM and MININUM of the bar, and set upper and lower thresholds; as you write values to the BAR, it will move up and down. These statements construct a BAR widget that travels from 1 to 100:

```
   ASSIGN @Bar TO WIDGET "BAR";SET ("ORIENTATION":"HORIZONTAL")
   FOR N=1 TO 100
```

```
     CONTROL @Bar;SET ("VALUE":N)
   NEXT N
```

This gives:

```
   +-----------------------------+---+
   |               BAR           | X |
   +-----------------------------+---+
   |                                 |
   +-------------+                   |
   |             |                   |
   +-------------+                   |
   |                                 |
   +---------------------------------+
```

Note that the BAR only moves when you SET a VALUE to it; it will not operate on its own.

The BAR can be divided into three ranges -- LOW, MIDDLE, and HIGH -- by specifying LOW and HIGH LIMITs:

```
   CONTROL @Bar;SET ("LOW LIMIT":10,"HIGH LIMIT":90)
```

Then you can set an alarm on one or more of these ALARM RANGES:

```
   CONTROL @Bar;SET ("ALARM RANGES":"LOW,HIGH")
```

By default, you'll get a BEEP every time the VALUE falls below the LOW LIMIT or above the HIGH LIMIT. You can change the BEEP to an event trap with:

```
   CONTROL @Bar;SET ("ALARM TYPE":"EVENT")
   ON EVENT @Bar,"ALARM" GOSUB Handler
```

Other BAR attributes allow you to change the MINIMUM and MAXIMUM limits, and set BAR background color, and the color of the LOW, MIDDLE, and HIGH ranges. You can also set the ORIENTATION to VERTICAL or HORIZONTAL, as shown above.

* METER:

The METER widget defines a moving-needle indicator; it it is programmed identically to the BAR widget, though it has a very different appearance and several additional features. If you create a METER widget:

```
   ASSIGN @M TO WIDGET "METER"
   FOR N=1 TO 100
```

```
    CONTROL @M;SET ("VALUE":N)
  NEXT N
```

-- you get the widget:

```
   +-------------+---+
   |      METER  | X |
   +--------+----+---+
   |   1    |   100  |
   +--------+--------+
   |        .        |
   |   .    |    .   |
   |        |        |
   | .      |      . |
   +-----------------+
   |       50        |
   +-----------------+
```

Note that the METER widget, unlike the BAR widget, displays its MINIMUM and MAXIMUM limits, and well as its current VALUE.

All attributes relevant to the BAR widget operate on the METER widget, and event handling is the same. However, the METER widget has other attributes to control its appearance, such as the SWEEP ANGLE (you can set it up to 360 degrees), the thickness of the needle and the arc, the background colors and font colors of the limits and VALUE boxes, and so on.

* LIMITS:

The LIMITS widget is similar to the BAR and METER widgets; it's something like a METER that's been bent flat. While a BAR widget indicates a changing value by changing the length of a bar, a LIMITS widget indicates a changing value by moving a "marker" through a fixed range. For example:

```
   ASSIGN @Lim TO WIDGET "LIMITS"
   CONTROL @Lim;SET ("MINIMUM":-50,"MAXIMUM":50)
   CONTROL @Lim;SET ("LOW LIMIT":25,"HIGH LIMIT":25)
```

-- gives a LIMITS widget of the form:

```
   +----------------------------+---+
   |             LIMITS         | X |
   +----------------------------+---+
   |        [-25 ]     [ 25 ]       |
   |                                |
   |   +-------+----------+------+  |
   |   |       |     I    |      |  |
   |   +-------+----------+------+  |
   |                                |
   | [-50 ]                 [ 50 ] |
   +--------------------------------+
```

The LIMITS widget uses similar attributes to the BAR and METER widgets.

* BARS:

The BARS widget is an extension of the BAR widget; if you simply create a BARS widget:

```
   ASSIGN @Bars TO WIDGET "BARS";SET ("VALUE":50)
```

-- you get a single bar indicator, but with limits, value, and label boxes attached, similar to the METER widget:

```
   +-------------------+---+
   |          BARS     | X |
   +-------------------+---+
   | [100] +-------------+ |
   |       |             | |
   |       |             | |
   |       |    +---+    | |
   |       |    |   |    | |
   |       |    |   |    | |
   |       |    |   |    | |
   | [ 1 ] +----+---+----+ |
   |       [     50      ] |
   |       [    BAR 1    ] |
   +-----------------------+
```

However, you can add more bars to the widget simply by setting the BAR COUNT attribute to the desired number of bars:

```
   ASSIGN @Bars TO WIDGET "BARS";SET ("BAR COUNT":3,"VALUES":Barval(*))
```

-- which gives:

```
   +-----------------------------+---+
   |               BARS          | X |
   +-----------------------------+---+
   | [100] +-----------------------+ |
   |       |                       | |
   |       |                       | |
   |       |         +---+         | |
   |       |  +---+  |   |         | |
   |       |  |   |  |   |  +---+  | |
   |       |  |   |  |   |  |   |  | |
   | [ 1 ] +--+---+--+---+--+---+--+ |
   |         [ 50  ][ 70  ][ 35  ]   |
   |         [BAR 1][BAR 2][BAR 3]   |
   +---------------------------------+
```

Now all 3 bars can be updated at once using the VALUES (not VALUE!) attribute with an array.

Some attributes of the BARS widget affect the entire widget, such as MAXIMUM and MINIMUM; a few can be set for an individual bar. You select an individual bar by using the CURRENT BAR attribute; you can then set parameters like the HIGH and LOW alarm ranges and the colors of the bar in the specified ranges.

You can set an ALARM event for the entire widget; then any bar value that falls into the specified range will trip an event.

* STRIPCHART, XYGRAPH:

The STRIPCHART widget provides a scrolling graphics display that can display up to 100 separate traces. This is one of the most complicated BPlus widgets, and summarizing its operation is difficult.

The STRIPCHART widget resembles a plotter display; it has a area in which to draw traces, plus numbering and tick-mark boxes along the X and Y axes. The STRIPCHART has three levels of attributes: those that let you to configure the overall operation of the widget; those that let you to configure the appearance of the X and Y axes boxes of the STRIPCHART, and those that let you to change colors (and other features) of each trace of the STRIPCHART.

The overall ("global") attributes allow you to set the ORIENTATION (VERTICAL or HORIZONTAL) of the STRIPCHART, the background color of the trace display, the number of traces (TRACE COUNT), and some other features.

You can set the global attribute CURRENT AXIS to select the X or Y axis and then use other attributes to set up an ORIGIN, RANGE, AXIS LABEL, and the numeric formats for that axis. If you don't want to mess with this stuff, you can set the axis to AUTOSCALE and not worry about the scaling, or use other attributes to turn off the numbering and tick mark boxes entirely.

You can then set the global attribute CURRENT TRACE to pick one of your traces and use other attributes to assign that trace a TRACE PEN color and a TRACE LABEL (the TRACE LABELs then appear at the bottom of the widget in their appropriate TRACE PEN color).

Once you have configured the appearance of the widget, you can (assuming that you have set the SHARED X attribute to 1) then update all the traces in parallel by writing an X value (using the global POINT LOCATION attribute) and then an array of Y values (using the global VALUES attribute); as you increment the POINT LOCATION, the STRIPCHART trace display scrolls left.

The XYGRAPH widget allows you to display the graphs of up to 100 different values simultaneously. Configuring the XYGRAPH is almost identical to configuring the STRIPCHART and the two have an almost identical appearance; however, loading data into the XYGRAPH is entirely different from loading it into the STRIPCHART.

In the XYGRAPH, you display a data set by setting a CURRENT TRACE, set the X DATA attribute with an array that contains the x data for the trace, and then set the Y DATA attribute with an array that contains the matching y data for the trace:

```
   CONTROL @Graf;SET ("CURRENT TRACE":3,"X DATA":X3(*),"Y DATA":Y3(*))
```

The trace is then drawn. You can also set a SHARED X attribute to allow a single x-data array to be used for all y-data arrays.

BPlus version 2 added two features to the STRIPCHART and XYGRAPH: it allows you to set a background graticule for the plotting area (a minor feature but one whose lack led to complaints in the first version); and the ability to place "markers" on traces that the user could then move with a mouse to determine trace values or differences between trace values.

* PULLDOWN MENU:

There are five widgets in BPlus used to create pulldown menu systems: PULLDOWN MENU, CASCADE MENU, MENU BUTTON, MENU TOGGLE, and MENU SEPARATOR.

When you create a PULLDOWN MENU widget, using a level-0 PANEL as a parent, you get a pulldown menu bar across the PANEL below the PANEL's title bar; the LABEL for the PULLDOWN MENU widget will appear in the bar.

The PULLDOWN MENU provides an "attachment point" for child widgets of the other MENU widget types; as you create instances of the other MENU widget types with a specific PULLDOWN MENU as a parent, the other MENU widget types build up a pulldown menu panel.

As you create other PULLDOWN MENU widgets, using the PANEL as a parent, their labels appear across the menu bar; they provide "attachment points" to other pulldown menus.

* CASCADE MENU:

This widget is very similar to the PULLDOWN MENU widget, except that it is created as a child of a PULLDOWN MENU widget to provide a second-level menu.
Like the PULLDOWN MENU widget, it provides an "attachment point" for other menu widgets.

You can also create CASCADE MENU widgets as children of other CASCADE MENU widgets, allow you to build a hierarchical tree of menus.

* MENU BUTTON:

This widget is created as the child of PULLDOWN or CASCADE MENU widgets, and creates an entry in a pulldown menu. It acts like a PUSHBUTTON in that you can set an ACTIVATED event on it to execute the desired menu action.

* MENU TOGGLE:

This widget is identical to the MENU BUTTON widget, except that it has a VALUE attribute that toggles between 0 and 1 every time it is selected. (A "*" also marks the menu entry when the MENU TOGGLE is set to 1.) The event in this case is named CHANGED.

* MENU SEPARATOR:

All this widget does is put a horizontal line in a pulldown menu to separate the different menu entries.

For an example of building a pulldown menu system, the following statements;

```
   ASSIGN @Pm TO WIDGET "PULLDOWN MENU";PARENT @Main,SET ("LABEL":"Menu")
   ASSIGN @Mb1 TO WIDGET "MENU BUTTON";PARENT @Pm,SET ("LABEL":"Button 1")
   ASSIGN @Tb1 TO WIDGET "MENU TOGGLE";PARENT @Pm,SET ("LABEL":"Toggle 1")
   ASSIGN @Cm TO WIDGET "CASCADE MENU";PARENT @Pm,SET ("LABEL":"Cascade")
   ASSIGN @Mb2 TO WIDGET "MENU BUTTON";PARENT @Cm,SET ("LABEL":"Button 2")
   ASSIGN @Tb2 TO WIDGET "MENU TOGGLE";PARENT @Cm,SET ("LABEL":"Toggle 2")
   !
   ON EVENT @Mb1,"ACTIVATED" GOSUB Handle_btn_1
   ON EVENT @Tb1,"CHANGED" GOSUB Handle_togl_1
   ON EVENT @Mb2,"ACTIVATED" GOSUB Handle_btn_2
   ON EVENT @Tb2,"CHANGED" GOSUB Handle_togl_2
```

-- give the menu system:

```
   +------------------------------
   | Menu
   +------------+-----------------
   | Button 1   |
   | Toggle 1   |
   | ---------- +----------+
   | Cascade  > | Button 2 |
   +------------+ Toggle 2 |
   |            +----------+
```

Note that the organization of the pulldown menu system arises from the sequence in which the widgets are created in the program.

* SYSTEM WIDGET:

One of the interesting additions BPlus in its second edition was the so-called SYSTEM widget. This was developed for use with the Screen Builder application. The Screen Builder doesn't generate code; it instead generates a file that describes the panel you have generated with it. The SYSTEM widget allows you to connect this file to a program to access its widgets:

```
   ASSIGN @Sys TO WIDGET "SYSTEM";SET ("*LOAD":Filename$)
```

Once the Screen Builder file has been loaded, then any widget defined in the file can be controlled via the SYSTEM widget by selecting it with the *NAME attribute. For example, if a main PANEL is defined in the file with the name "Main", it can be accessed with:

```
   CONTROL @Sys;SET ("*NAME":"Main","WIDTH":10,"HEIGHT":20)
```

Similarly, if "Main" has a child widget named "Meter2", that child can be accessed by using a "pathname" that specifies both the parent and child:

```
   CONTROL @Sys;SET ("*NAME":"Main/Meter2","X":10,"Y":20)
```

Events can be trapped through the SYSTEM widget as well, though all you can do is specify the event type to be trapped -- not the specific event source:

```
   ON EVENT @Sys,"CLICKED" GOTO Handler
   ON EVENT @Sys,"DONE" GOTO Handler
```

You can set events on a SYSTEM menu for the widgets it contains, though you can't specify which widget the event belongs to. When the event trap occurs, you can determine the source by using the *QUEUED EVENT attribute:

```
   Ev$(1:2)[50]
   STATUS @Sys;RETURN ("*QUEUED EVENT":Ev$(*)
   PRINT "Widget name: ";Ev$(1);"  Widget event: ";Ev$(2)
```

SYSTEM widget event handling, however, is a complicated subject and will be discussed in more detail in a later chapter.

The SYSTEM widget also has an attribute named *CREATE that allows you to create widgets without using the Screen Builder. For example:

```
   FOR N=1 TO 10  ! Create ten PUSHBUTTONs.
     CONTROL @Sys;SET ("*NAME":"B"&VAL$(N),"*CREATE":"PUSHBUTTON")
   NEXT N
```

You could then use *NAME to access any of the ten PUSHBUTTONs in the SYSTEM widget. You can use a SYSTEM widget as a child of a PANEL defined in the conventional fashion to set up a button or label array for the PANEL.

[5.2] GENERIC WIDGET & DIALOG ATTRIBUTE REFERENCE

* There is a set of common attributes, as described by the following list. All dimensions and locations are described in pixels:

BACKGROUND	Type:	numeric
	Values:	legal PEN values (0-255)
WIDGETS & DIALOGS	Default:	depends on system defaults

Specifies PEN color for widget or dialog background.

BORDER	Type:	numeric
	Values:	0 or 1
WIDGETS	Default:	1

If 1, a border is drawn around the widget.

DEFAULT BUTTON	Type:	numeric
	Values:	valid index into DIALOG BUTTONS
DIALOGS	Default:	index to "OK" button

This is the button that is activated when you press Enter.

DIALOG BUTTONS	Type:	string array
	Values:	valid strings
DIALOGS	Default:	varies

Allows you to read or change dialog buttons.

FONT	Type:	string
	Values:	font names
most WIDGETS, DIALOGS	Default:	depends on system defaults

Specifies text style.

HEIGHT	Type:	numeric
	Values:	widget or dialog height in pixels
WIDGETS & DIALOGS	Default:	varies

Specifies height in pixels of widget or dialog. May depend on ROWS attribute on some widgets or dialogs.

```
INSIDE WIDTH            Type:    numeric
                        Values:  widget interior width in pixels
WIDGETS                 Default: varies
```

Read-only attribute, gives interior width of widget.

```
INSIDE HEIGHT           Type:    numeric
                        Values:  widget interior height in pixels
WIDGETS                 Default: varies
```

Read-only attribute, gives interior height of widget.

```
JUSTIFICATION           Type:    string
                        Values:  "LEFT" or "CENTER"
some WIDGETS, DIALOGS   Default: "CENTER"
```

Specifies justification of text in dialog. (Also used in some widgets along with a "RIGHT" justification.)

```
MAXIMIZABLE             Type:    numeric
                        Values:  0 or 1
WIDGETS (0) & DIALOGS   Default: 1 (YES)
```

If 1 and mouse events are enabled, a MAXIMIZABLE box will appear in the title bar, allowing user to pop panel to a full screen display (and back).

```
MINIMIZABLE             Type:    numeric
                        Values:  0 or 1
WIDGETS (0) & DIALOGS   Default: 0 (NO)
```

If 1 and mouse events are enabled, a MINIMIZABLE button will appear in the title bar, allowing user to pop panel to hide the widget in the Minimized Windows application. (BPlus 2.0 & later only.)

```
MOVABLE                 Type:    numeric
                        Values:  0 or 1
WIDGETS (0) & DIALOGS   Default: 1 (YES)
```

If 1, the widget (level-0 only) or dialog can be moved around on the display by the user with a mouse. If 0, the object is "locked down".

```
PEN                     Type:    numeric
                        Values:  PEN numbers (0-255)
most WIDGETS, DIALOGS   Default: depends on system defaults
```

Specifies text color.

RESIZABLE	Type:	numeric
	Values:	0 or 1
WIDGETS (0) & DIALOGS	Default:	1 (YES)

If 1, a RESIZE border is drawn around the widget (level-0 only) or dialog, which a user can "click and drag" to change the size of the object.

RESTORE SCREEN	Type:	numeric
	Values:	0 or 1
WIDGETS & DIALOGS	Default:	0 (don't restore screen)

Specifies whether the display underneath the widget or dialog will be preserved when the object is removed. If RESTORE SCREEN is set (1), a "snapshot" of the display underneath is saved when the object is created, which means that the object should not be moved or resized (since that original "snapshot" would then no longer be valid).

STACKING ORDER	Type:	numeric
	Values:	0 to the number of siblings
WIDGETS (0)	Default:	0

Given a set of level-0 widgets that overlay each other, defines which one (STACKING ORDER = 0) will be on top.

SYSTEM MENU	Type:	string, string array (level-0 only)
WIDGETS (0)	Values:	legal string values

Allows you to define a "toaster menu" in a level-0 widget by giving it a menu entry as a string or a set of menu entries as a string array. (BPlus 2.0 & later only.)

SYSTEM MENU COUNT	Type:	numeric (level-0 only, read only)
WIDGETS (0)	Values:	positive integer

Returns the number of items in the SYSTEM MENU. (BPlus 2.0 & later only.)

SYSTEM MENU EVENT	Type:	numeric (level-0 only, read only)
	Values:	0 to items -1
WIDGETS (0)	Default:	0

Returns the string array index of the SYSTEM MENU item that generated the last SYSTEM MENU event; the array is always considered to start with an index of 0. (BPlus 2.0 & later only.)

```
TITLE                  Type:    string
                       Values:  any string
WIDGETS (0) & DIALOGS  Default: widget name
```

String that will appear in the widget (level-0 only) or dialog header.

```
USER DATA              Type:    string
                       Values:  any string
WIDGETS & DIALOGS      Default: "" (null string)
```

A user-defined string that can be associated with a widget or dialog.

```
VERSION                Type:    string
WIDGETS & DIALOGS      Values:  any string
```

Read-only; returns the version level of BPlus (2.0 & later only).

```
VISIBLE                Type:    numeric
                       Values:  0 or 1
WIDGETS                Default: 1 (visible)
```

If 1, widget is visible; if 0, widget is hidden.

```
WIDTH                  Type:    numeric
                       Values:  object width in pixels
WIDGETS & DIALOGS      Default: depends on system defaults
```

Specifies width in pixels of widget or dialog.

```
X                      Type:    numeric
                       Values:  object x-location in pixels
WIDGETS & DIALOGS      Default: autoplacement
```

Specifies the location along the horizontal axis of the upper left corner of the object, with respect to the upper-left-corner of the display (for a standalone widget or all dialogs) or the upper-left-corner of the parent panel (for a widget that is part of a PANEL).

```
Y                      Type:    numeric
                       Values:  object y-location in pixels
WIDGETS & DIALOGS      Default: autoplacement
```

Specifies the location along the vertical axis of the upper left corner of the object, with respect to the upper-left-corner of the display (for a standalone widget or all dialogs) or the upper-left-corner of the parent panel (for a widget that is part of a panel).

All level-0 widgets (as of Bplus 2.0) can generate the following events:

CLOSE	Occurs when the widget is destroyed.
SYSTEM MENU	Occurs when the SYSTEM MENU is activated.

[6.0] BPLUS (6): BASIC Plus Applications

v2.4 / 01 feb 99 / greg goebel

* The BPlus product includes an applications environment that runs in the HP BASIC environment. It features an "Application Manager" that allows use of several built-in applications. This chapter describes the applications environment and the standard applications in detail.

[6.1] THE APPLICATIONS ENVIRONMENT & APPLICATIONS MANAGER

* The BASIC Plus applications environment provides access to the following standard applications:

   * Notepad editor.
   * Clock.
   * Help utility.
   * Help compiler.
   * Screen Builder.

There was no applications environment in BPlus before the 2.0 revision, and the only application that could be used was an earlier and different version of the Help utility. The applications environment also does not apply, as mentioned in earlier chapters, to HBW; the only application supported is the Screen Builder (HBW online Help uses the standard MS-Win Help utility).

* If you have BPlus installed, and you enter at the RMB prompt:

```
   APP
```

-- then the standard RMB display will go away to be replaced with a gray background, with the following panel floating in the upper-left corner:

```
+---+-----------------------------------------------+
| = |             Application Manager               |
+---+-----------------------------------------------+
| File  Help                                        |
+---------------------------------------------------+
|  +------+  +------+  +------+  +------+  +------+  |
|  | icon |  | icon |  | icon |  | icon |  | icon |  |
|  |      |  |      |  |      |  |      |  |      |  |
|  +------+  +------+  +------+  +------+  +------+  |
|   Screen    Notepad   Clock      Help      Help    |
|   Builder                                Compiler  |
+---------------------------------------------------+
```

This is the Application Manager; it is somewhat similar in concept to (but simpler than) MS-Windows Program Manager in that it can be used to run programs identified at sets of icons.

There isn't much to using it; you can either double-click on an icon to run the application, or run it from the "File" menu:

```
| File  Help
+-------+-----
| Run   |
| Exit  |
+-------+
|
```

-- as well as bail out of the applications environment and return to the standard RMB environment. The other menu, "Help":

```
    Help
 -+-----------+-
  | Contents  |
  | Version   |
  +-----------+
```

-- can also be used to bring up the Help utility, as well as identify the rev level of the Application Manager.

* Anyway, you can use the application manager to run the BPlus apps, and you can run multiple copies of them:

```
   +---+-----------------------------------------------+
   | = |             Application Manager               |
   +---+-----------------------------------------------+
   |                                                   |
   |                      +---+--------------------------+---+---+
   |                      | = |           Clock          | x | X |
   |                      +---+--------------------------+---+---+
   |                      | Display  Timer  Alarm  Options       | --+---+
   |      +---+--------------------------------------+---+---+   |   | X |
   |      | = |               Notepad                | x | X |   | --+---+
   |      +---+--------------------------------------+---+---+   |       |
   +----- | File  Edit  Search                               |   |       |
          +--------------------------------------------------+   |       |
          |                                                  |   |       |
          |                                                  |   |       |
          |                                                  |   |       |
          |                                                  |   |       |
          |                                                  |   |       |
          |                                                  | --+       |
          |                                                  |           |
          |                                                  |           |
          |                                                  |           |
          |                                                  | ----------+
          +--------------------------------------------------+
```

You can "maximize" an individual app by clicking on the first button from the right; this expands the app to full screen. You can reduce it back to "normal" size by clicking it again.

You can "minimize" indivdual apps by clicking on the second button from the right; this reduces the app to an entry in a list box, the Minimized Windows Manager (which is actually part of the Application Manager and not, strictly speaking, an app itself, though it's convenient to think of it as one). For example, if we iconize the Notpad editor in the illustration above, we get:

```
   +------------------------+
   |    Minimized Windows   |
   +------------------------+
   |[635   : Notepad       ]|
   |                        |
   |                        |
   |                        |
   +------------------------+
```

Note that apps also have a "toaster box" in the upper-left corner; usually this just gives a short menu to allow you to conveniently close the app.

* Now for some fine details. In the example above, we ran the applications

environment and invoked apps from it; but you can actually bypass the environment and run them directly if you like, by invoking "APP" along with the application name:

```
   APP "APCLOCK"    Run Clock application.
   APP "APHCOMP"    Run Help Compiler.
   APP "APMGR"      Run Application Manager.
   APP "APNOTEP"    Run NotePad.
   APP "APSCRBLD"   Run Screen Builder.
```

Note that you can run Help by simply invoking HELP from the RMB prompt. All these commands can be executed programmatically, allowing you to integrate them into RMB applications ... though about the only app that is really useful in that context is the NotePad editor.

Applications have life of their own, separate from RMB programs -- running, pausing, stopping, editing, scratching, loading, or getting a program has no effect on apps. When apps are active, BPlus "owns" system resources -- the keyboard, display, and so on. An application continues to live until the user closes it, or until the RMB environment undergoes a Shift-reset or a SCRATCH A.

The background color or a tiled bitmap for the application environment can be specified in the CONFIG.

Note that you cannot execute an RMB program from the applications environment; that is basically attempting to run such an RMB program out of a CSUB, and RMB won't let you do that. Apps are written in C code, using MS-Windows-like calls; simple Windows programs can be in theory ported to the RMB application environment, but unfortunately HP does not offer tools to allow users to build their own apps.

[6.2] NOTEPAD

* The Notepad application is a simple text editor with mouse-selectable cut, copy, paste, and replace functions; you can use the mouse to select text for cutting, copying, and pasting. It has a simple user interface:

```
   +---+---------------------------------------------------------+---+---+
   | = |                          Notepad                        | x | X |
   +---+---------------------------------------------------------+---+---+
   | File  Edit  Search                                                  |
   +---------------------------------------------------------------------+
   |                                                                     |
   |                                                                     |
   |                                                                     |
   |                                                                     |
   +---------------------------------------------------------------------+
```

The "File" menu provides various file-I/O functions (as well as allows you to quit Notepad):

```
   | File  Edit  Searc
   +----------------+-
   | New            |      Wipe current contents of Notepad.
   | Open...        |      Load file into Notepad.
   | Save           |      Save file to current filename.
   | Save As...     |      Save file to new filename.
   | Read...        |      Merge file into current text.
   | Quit           |      Quit Notepad.
   +----------------+
   |
```

The "Edit" menu is used to perform various operations on text within the editor that has been selected with mouse click-and-drag operations:

```
     Edit  Search
  -+-----------+-
   | Cut       |        Cut text out of editor, put in paste buffer.
   | Copy      |        Copy text from editor and put in paste buffer.
   | Paste     |        Insert paste buffer contents into editor.
   | Replace   |        Replace selected text with paste buffer contents.
   | Clear     |        Cut text out of editor, don't put in paste buffer.
   +-----------+
```

Finally, the "Search" menu provides the functions:

```
    Search
  -+-------------------+-
   | Line Number...    |  Move cursor to designated line number.
   | String...         |  Search for a string.
   | Next occurrence   |  Repeat search.
   +-------------------+
```

Cutting and copying place the selected text in a hidden "clipboard" accessible by other Notepads -- you can run more than one -- or by the Bplus 2.0 STRING widget.

The Notepad is not intended as a replacement for the RMB editing environment! You can edit RMB programs as text files, of course, but it provides no syntax checking and you can't execute a program from it.

[6.3] CLOCK

* The Clock app is simple in concept:

```
   +---+----------------------------+---+---+
   | = |            Clock           | x | X |
   +---+----------------------------+---+---+
   | Display  Timer  Alarm  Options         |
   +----------------------------------------+
   |                                        |
   |                   *                    |
   |            *             *             |
   |                   |                    |
   |                   |                    |
   |         *         +-------- *          |
   |                                        |
   |                                        |
   |            *             *             |
   |                   *                    |
   |                                        |
   |              12:15:03 PM               |
   |                                        |
   +----------------------------------------+
```

The "Display" menu has the following entries:

```
   | Display  Ti
   +----------+-
   |  Analog  |               Use analog (HMS-hands) display.
   |  Digital |               Use digital (numeric) display.
   | *Mixed   |               Use combined analog and digital display.
   |  Timer   |               Set digital timer/stopwatch mode.
   +----------+
   |
```

The "Timer" menu sets timer operation modes:

```
     Timer  Alarm  Options
  -+-----------------------+-
   |  Set Value...         |    Set timer count value.
   |  Set Update Rate...   |    Set the timer count rate.
   | *Timer counts down    |    Set timer count direction.
   |  Timer repeats        |    Set whether timer counts cyclically.
   |  Notification         |    Set timer notification mode.
   |  Start                |    Start timer.
   |  Reset                |    Clear timer.
   |  Pause                |    Pause timer.
   +-----------------------+
```

The "Alarm" menu sets alarm operations:

```
     Alarm  Options
  -+-----------------+-
   | Set Time...     |         Set alarm time.
   | Notification    |         Set alarm notification mode.
   +-----------------+
```

The "Options" menu sets a few minor functions:

```
     Options
  -+-----------------+-
   | *Show seconds   |         Enable or disable seconds hand/digits.
   |  Military time  |         Enable or disable 24-hour count.
   +-----------------+
```

[6.4] HELP

* The most useful app in the BPlus set is the Help utility (remember, once more, that HBW uses the MS-Win Help utility). The panel selected by HELP looks like this:

```
   +---+-------------------------------------------------+---+---+
   | = |                HP BASIC: Contents               | x | X |
   +---+-------------------------------------------------+---+---+
   | File  SeeAlso  Programs                                     |
   +----------+--------+------+----------+------+----------------+
   | Contents | Search | Back | Copy Code | Quit |               |
   +----------+--------+------+----------+------+--------------+-+
   |                                                          |^|
   | Keyword Keywords                                         | |
   |                                                          | |
   |   Widget Dictionary                                      | |
   |   Keywords By Category                                   | |
   |                                                          | |
   | BASIC Plus Information                                   | |
   |                                                          | |
   |   BASIC Plus Widget Reference                            | |
   |   BASIC Plus Keyword Dictionary                          | |
   |   BASIC Plus Examples                                    | |
   |                                                          | |
   |                                             line 11 of 16 |v|
   +--------------------------------------------------------------+-+
```

The "File" menu allows you to open a Help file (you can make your own, as the next section shows), print the current help topic to the current RMB PRINTER IS device, or exit the utility:

```
   | File  SeeAlso  Programs
   +-----------+--------------
   | Open...   |
   | Print     |
   | Exit      |
   +-----------+
   |
```

The "SeeAlso" and "Programs" menus are defined by the current Help file; they allow you to cross-reference the current Help topic to other help topics, and to get a list of example programs for the Help topic.

* Using the Help Utility is straightforward. When you first bring up the utility using the standard RMB help file, you get the master menu:

```
   +---+---------------------------------------------------+---+---+
   | = |                HP BASIC: Contents                 | x | X |
   +---+---------------------------------------------------+---+---+
   | File  SeeAlso  Programs                                       |
   +----------+--------+------+----------+------+----------------+
   | Contents | Search | Back | Copy Code | Quit |                 |
   +----------+--------+------+----------+------+--------------+-+
   |                                                           |^|
   | BASIC Keywords                                            | |
   |                                                           | |
   |   Keyword Dictionary                                      | |
   |   Keywords By Category                                    | |
   |                                                           | |
   | BASIC Plus Information                                    | |
   |                                                           | |
   |   BASIC Plus Widget Reference                             | |
   |   BASIC Plus Keyword Dictionary                           | |
   |   BASIC Plus Examples                                     | |
   |                                                           | |
   |                                              line 11 of 16 |v|
   +-----------------------------------------------------------+-+
```

The help topics appear in blue on a color display; anything in blue is an index to another topic, all you have to do is click on it and you get that subject. (The non-blue text won't do anything if you click on it.) Note that all the topics are not displayed, the line on the bottom indicates that there are five more lines scrolled off the bottom of the screen; you can use the scrollbar at the right of the panel to see the rest:

```
   | General Information                                       | |
   |                                                           | |
   |   Programmer's Reference                                  | |
   |   Latest Information                                      | |
   |   Using Online Help                                       | |
   |                                              line 16 of 16 |v|
   +-----------------------------------------------------------+-+
```

You can scroll one line at a time by clicking on one of the arrows on the scrollbar, or you can grab the marker with your mouse and move text up and down.

In any case, you can then click on the subject you want to investigate; for example, you could go to the "Keyword Dictionary" entry:

```
+---+--------------------------------------------------+---+---+
| = |             HP BASIC: Keyword Dictionary         | x | X |
+---+--------------------------------------------------+---+---+
| File  SeeAlso  Programs                                      |
+----------+--------+------+----------+------+----------------+
| Contents | Search | Back | Copy Code | Quit |                |
+----------+--------+------+----------+------+--------------+-+
|                                                            |^|
| - A -                                                      | |
| ABORT                   ABORTIO                            | |
| ABS                     ACS                                | |
| ACSH                    ALLOCATE                           | |
| ALPHA                   ALPHA HEIGHT                       | |
| ALPHA PEN               AND                                | |
| APPEND                  AREA                               | |
| ARG                     ASCII                              | |
| ASN                     ASNH                               | |
| ASSIGN                  ATN                                | |
| ATNH                    AXES                               | |
|                                         line 13 of 270     |v|
+------------------------------------------------------------+-+
```

Click on the "ABORT" to get that entry:

```
+---+--------------------------------------------------+---+---+
| = |                 HP BASIC: ABORT                  | x | X |
+---+--------------------------------------------------+---+---+
| File  SeeAlso  Programs                                      |
+----------+--------+------+----------+------+----------------+
| Contents | Search | Back | Copy Code | Quit |                |
+----------+--------+------+----------+------+--------------+-+
| Description                                                |^|
|                                                            | |
|   ABORT ceases HP-IB activity.  When system controller     | |
|   but not active controller, ABORT causes the computer     | |
|   to assume active control.                                | |
|                                                            | |
|   Supported on:  UX WS DOS IN                              | |
|   Required binary:  IO                                     | |
|                                                            | |
| Examples:                                                  | |
|                                                            | |
| e>ABORT 7                                                  | |
|                                          line 11 of 16     |v|
+------------------------------------------------------------+-+
```

Note the example statement at the bottom -- it's marked with an "e>" and is in black. If you click on it, that statement will be copied into the RMB editor.

You can get a list of related topics by selecting the "SeeAlso" menu:

```
   | File  SeeAlso  Programs
   +------+-----------------+----
   |      |[Instrument I/O] |
   +----- | ABORTIO         | ---
   |      | BREAK           |
          | CLEAR           |
          | RESET           |
          +-----------------+
```

Click on one of these entries and you will get the Help entry for it.

* That should give you a general idea of how the Help utility works. Now for a few other details.

The "Programs" menu allow you to jump to program examples in the Help utility related to the current entry, much like "SeeAlso" -- though relatively few entries have example programs linked to them.

The buttons below the menu:

```
   +----------+--------+------+-----------+------+
   | Contents | Search | Back | Copy Code | Quit |
   +----------+--------+------+-----------+------+
```

-- give fast access to a number of functions. When you click on the "Contents" button, you're popped back to the table of contents (as seen at the beginning of this section); the "Back" button allows you to retrace through previous screens; if you have accessed an entry that consists of a program, the "Copy Code" button dumps the code into the RMB editor.

I skipped the "Search" entry because it's a little more complicated than the others ... click on it and you get a dialog:

```
+---+-----------------------------------------------------------+---+---+
| = |                      HP BASIC: SEARCH                     | x | X |
+---+-----------------------------------------------------------+---+---+
| Search for:                                                           |
| +-----------------------------------------+                           |
| | ABORT statement                         |  +- Extended Search -+ |
| +-----------------------------------------+  |    +---------+    | |
| Index Entries                                |    |  Start  |    | |
| +---------------------------------------+-+  |    +---------+    | |
| |[ABORT statement                      ]|^|  |    |  Stop   |    | |
| | ABORTIO                              | |  |    +---------+    | |
| | ABS function                         | |  +--------------------+ |
| | ACS function                         | |       +---------+       |
| | ACSH function                        |v|       |  Cancel |       |
| +---------------------------------------+-+       +---------+       |
| ==================================================================== |
| Topics Found: 1                                                       |
| +-----------------------------------------+       +---------+       |
| |[ABORT                                  ]|       |  Go To  |       |
| |                                         |       +---------+       |
| |                                         |                         |
| +-----------------------------------------+                         |
+-----------------------------------------------------------------------+
```

You can enter topics in the "Search for" field and "Start" an "Extended Search" to see if there is any such topic indexed into the Help file; all indexed entries are listed in the "Topics Found" box, and you can then "Go To" each as you like. You could also search through the "Index Entries" to see all the indexed entries available ... note that these index pointers have to be specifically defined in the Help file -- you may not be able to find some topics even if they exist in the Help file, if they haven't been indexed.

* The Help system has some other features. Suppose you wanted help on the IMAGE statement; from RMB, you could enter the statement:

```
   HELP "ABORT"
```

-- and you would get the entry for ABORT as we investigated earlier. You can
also optionally invoke Help from RMB using a different file:

```
   HELP "SaveEntry","DBASE.HLP"
```

As noted elsewhere, you can also "connect" a widget in a user interface to a
Help file and Help topic using the HELP FILE and HELP TOPIC attributes.

[6.5] HELP COMPILER

* The Help Compiler allows you to create your own help files for the BPlus Help app (such files are not compatible with the MS-Win help utility used with HBW). It is simple in appearance:

```
+---+---------------------------------------------------------+---+---+
| = |                    Help File Compiler                   | x | X |
+---+---------------------------------------------------------+---+---+
| File                                                                |
+---------------------------------------------------------------------+
| Source File:                                                        |
| +------------------------------------+ +------------+ +------------+ |
| |                                    | |    Help    | |   Compile  | |
| +------------------------------------+ +------------+ +------------+ |
| Destination File:                      | Options... | |    Stop    | |
| +------------------------------------+ +------------+ +------------+ |
| |                                    | |    Try...  | |    Quit    | |
| +------------------------------------+ +------------+ +------------+ |
| Compile Status                                                      |
| +-----------------------------------------------------------------+ |
| |                                                                 | |
| |                                                                 | |
| |                                                                 | |
| |                                                                 | |
| |                                                                 | |
| |                                                                 | |
| |                                                                 | |
| +-----------------------------------------------------------------+ |
+---------------------------------------------------------------------+
```

You simply give the Help Compiler the name of a source Help file -- which is a text file written using HELPX Help-language format codes -- and the name of a target Help file, and then click on the "Compile" button; the "Compile Status" shows the results.
The "Help" button brings up the Help utility with a Help file that provides a tutorial on the HELPX language. The "Options..." button allows you to select several compiler options:

```
+--------------------------------------------------------------------+
|                         Compiler Options                           |
+--------------------------------------------------------------------+
|                                                                    |
| [ ] Warn unrecognized tags (treat as regular text)                 |
|                                                                    |
| [ ] Strip unrecognized tags                                        |
|                                                                    |
| [ ] Ignore unreferenced topics                                     |
|                                                                    |
| [ ] Ignore duplicate <idx> entries                                 |
|                                                                    |
| [ ] Generate examples files (from <ex> and PROGRAM topics)         |
|                                                                    |
|                +----------+              +----------+              |
|                |    OK    |              |  Cancel  |              |
|                +----------+              +----------+              |
+--------------------------------------------------------------------+
```

-- that don't make too much sense unless you know about HELPX. The "Try..." button allows you to bring up the Help utility with a given Help file and topic:

```
   +--------------------------------------------------------+
   |                 Try Help File In Viewer                |
   +--------------------------------------------------------+
   | Help Topic:                                            |
   | +----------------------------------------------------+ |
   | |                                                    | |
   | +----------------------------------------------------+ |
   | Help File:                                             |
   | +----------------------------------------------------+ |
   | |                                                    | |
   | +----------------------------------------------------+ |
   |                                                        |
   |        +-----------+              +-----------+        |
   |        |    Try    |              |  Cancel   |        |
   |        +-----------+              +-----------+        |
   +--------------------------------------------------------+
```

The "Stop" button halts a compile in progress, and of course "Quit" bails out of the utility. The "File" menu on the main panel simply duplicates the "Source...", "Destination...", and "Quit" buttons.

[6.6] SCREEN BUILDER

* The Screen Builder application is an interactive tool for placing and sizing widgets within a panel; it can save the layout it defines for editing or running later, or for incorporating into a program using the SYSTEM WIDGET. As noted, unlike the other apps, it is supported under HP BASIC for Windows.

If you bring up the Screen Builder, you get the panels:

```
+---+----------------------+---+---+ +------------------------------+---+
| = |   HP Screen Builder   | x | X | |             Panel0            | X |
+---+----------------------+---+---+ +------------------------------+---+
| File  Widget  Options            | |                                  |
+----------------------------------+ |                                  |
| -- Widget Types ---------------- | |                                  |
|                                  | |                                  |
| [  Panel  ][Pulldown ][ Cascade ] | |                                  |
|                                  | |                                  |
| [ MButton ][ MToggle ][ MSepar  ] | |                                  |
|                                  | |                                  |
| [ PSepar  ][ Button  ][  Radio  ] | |                                  |
|                                  | |                                  |
| [ Toggle  ][Scrollbar][  Label  ] | |                                  |
|                                  | |                                  |
| [ String  ][ Numeric ][  List   ] | |                                  |
|                                  | |                                  |
| [ Combo   ][   Bar   ][  Bars   ] | |                                  |
|                                  | |                                  |
| [  File   ][ Limits  ][  Meter  ] | |                                  |
|                                  | |                                  |
| [ Printer ][ Slider  ][ StripC  ] | |                                  |
|                                  | |                                  |
| [ XYGraph ][ Bitmap  ][  HPGL   ] | |                                  |
|                                  | |                                  |
| [  Clock  ][ Keypad  ]           | |                                  |
|                                  | |                                  |
| -------------------------------- | |                                  |
+----------------------------------+ +----------------------------------+
```

The Screen Builder allows you to set up a user interface and store information that describes it into a file. The "File" menu allows you to perform the appropriate operations on such files:

```
| File  Widget
+-------------+-
| Open...     |      Open an existing file.
| New         |      Start on new interface.
| Save        |      Save file under current name.
| Save As...  |      Save file under new name.
| Exit        |      Exit the screen builder.
+-------------+
|
```

The "Widget" menu allows you to create and perform other operations on widgets:

```
     Widget  Optio
  -+------------+-
   | Create...  |       Create a new widget.
   | Edit...    |       Edit an existing widget.
   | Delete...  |       Delete an existing widget.
   | Re-name... |       Re-name an existing widget.
   | Copy...    |       Copy an existing widget to a new one.
   +------------+
```

The "Options" menu allows you to select two options:

```
     Options
  -+------------+-
   | Grid...    |       Define "snap grid" for placing widgets.
   |*Status     |       Optionally set up a widget status panel.
   +------------+
```

* Using the Screen Builder is simple. You are, by default, given an initial PANEL to lay out widgets; you can then create a new widget by selecting one from the Screen Builder panel (or, if you feel contrary, from the "Create" menu entry). Say you select a BITMAP widget from the Screen Builder panel; you first get a dialog to ask you what to use for a parent, if anything:

```
   +--------------------------------------------------------+
   |                        Create                          |
   +--------------------------------------------------------+
   |                                                        |
   | Select parent of new widget, if any:                   |
   |                                                        |
   |   +---------------------------------------------+-+    |
   |   |[No Parent                                  ]|^|    |
   |   | Panel0                                      | |    |
   |   |                                             | |    |
   |   |                                             | |    |
   |   |                                             | |    |
   |   |                                             | |    |
   |   +---------------------------------------------+-+    |
   |                                                        |
   |        +----------+              +----------+          |
   |        |    OK    |              |  Cancel  |          |
   |        +----------+              +----------+          |
   +--------------------------------------------------------+
```

Note how the intial PANEL is identified as a string, "Panel0"; this is analogous to a widget handle and is used in the target program to identify a widget.

Once you select a parent -- of course we select "Panel0" in this case --

you get a dialog to define the identifier string of the new BITMAP widget:

```
   +--------------------------------+
   |             Create             |
   +--------------------------------+
   |                                |
   | Enter name of new widget:      |
   |                                |
   |   +-----------------------+    |
   |   |Bitmap0                |    |
   |   +-----------------------+    |
   |                                |
   +--------------------------------+
   |   +-----------+ +-----------+  |
   |   |    OK     | |  Cancel   |  |
   |   +-----------+ +-----------+  |
   +--------------------------------+
```

We'll leave the name at the default, "Bitmap0", for now ... anyway, click on OK and the BITMAP widget appears on the default PANEL, and you can move or size it as you please. You can continue to add widgets in this way for about as long as you like.

* Once you establish widgets, though, you will want to set the appropriate attributes on them. You do this with the "Edit" menu entry, which gives you the dialog:

```
   +--------------------------------------------------------+
   |                         Edit                           |
   +--------------------------------------------------------+
   |                                                        |
   | Select a widget:                                       |
   |                                                        |
   |   +--------------------------------------------+-+     |
   |   |[Panel0                                    ]|^|     |
   |   | Panel0/Bitmap0                             | |     |
   |   |                                            | |     |
   |   |                                            | |     |
   |   |                                            | |     |
   |   |                                            | |     |
   |   +--------------------------------------------+-+     |
   |                                                        |
   |        +-----------+             +-----------+         |
   |        |    OK     |             |  Cancel   |         |
   |        +-----------+             +-----------+         |
   +--------------------------------------------------------+
```

This is lists all the widgets in the user interface being constructed -- we only have two, the default PANEL and the BITMAP widget. Note how the name of the PANEL and the BITMAP form a "path" -- "Panel0/Bitmap0". This path defines the parent-child relationship between the two.

Anyway, select the BITMAP widget and you get an editing dialog:

```
   +-------------------------------------------------------+
   |                    Panel/Bitmap0                      |
   +-------------------------------------------------------+
   |                                                       |
   |             +-----------------------------------+-+   |
   | Attributes: |                                   | |   |
   |             +-----------------------------------+-+   |
   |                                                       |
   | +- Options ---------------------------------------+ |
   | |                                                 | |
   | |   [ ] Widget enabled.                           | |
   | |                                                 | |
   | |   [ ] Attributes auto-updated.                  | |
   | |                                                 | |
   | +-------------------------------------------------+ |
   |                                                       |
   |        +----------+             +----------+          |
   |        |    OK    |             |  Cancel  |          |
   |        +----------+             +----------+          |
   +-------------------------------------------------------+
```

You can also get this edit dialog simply by double-clicking on a widget ...
but to continue, the "Attributes" field controls a pull-down menu:

```
               +-----------------------------------+-+
   Attributes: |                                   | |
               +-----------------------------------+-+
               | AUTO SIZE                           |
               | BACKGROUND                          |
               | BITMAP FILE                         |
               | BITMAP HEIGHT                       |
               | ...                                 |
```

Select an attribute, say "BITMAP FILE", and you get a dialog to set its
value:

```
+----------------------------------------+
|              BITMAP FILE               |
+----------------------------------------+
|                                        |
|  Type:     String                      |
|           +-----------------------+    |
|  Value:   |                       |    |
|           +-----------------------+    |
|                                        |
|      +----------+   +----------+       |
|      |   Set    |   |  Close   |       |
|      +----------+   +----------+       |
+----------------------------------------+
```

Enter a file name, then click on "Set", and you'll set that attribute on the widget -- but you won't go back to the editor dialog until you click on "Close". (This operation can seem a little frustrating at first.)

* Referring back to the widget-editing dialog, there are two flags you can set -- "Widget enabled" and "Attributes auto-updated". The two work together.

Normally, you can't interact with a widget while you are manipulating it with the Screen Builder; you can't enter text in a STRING widget, or move the slider in a SLIDER widget. Setting "Widget enabled" allows you to interact with the widget, though the value won't be saved in the definition file.

That might seem pretty useless, except for the second flag, "Attributes auto-updated". When you have set "Widget enabled" so you can interact with a widget, and then set "Attributes auto-updated", then when you use the editing dialog and try to modify an attribute in a dialog, that attribute will change as you change the corresponding component on the widget.

That means that if you have the flags set, and attempt to edit the VALUE attribute on a SLIDER widget, the value in the setting dialog will change as you move the slider on the widget -- and will be saved as the appropriate value when you click on the "Save:" button.

* The "Delete", "Re-name", and "Copy" menu entries simply bring up a dialog with a list of the existing widgets; you can select a widget from the list, and following dialogs, much like those already seen, will step you through the operation.

The "Copy" operation does have the additional nice feature of allowing you to create as many copies as you like at once, and place them in a row or column. When you select a widget to copy, you get the dialog:

```
+-------------------------------------------------------+
|                         Copy                          |
+-------------------------------------------------------+
|                                                       |
|                        Copies         Spacing         |
|                       +------------+  +-----------+   |
|    X direction:       | 0          |  | 0         |   |
|                       +------------+  +-----------+   |
|                                                       |
|                       +------------+  +-----------+   |
|    X direction:       | 0          |  | 0         |   |
|                       +------------+  +-----------+   |
|                                                       |
|                       +---------------------------+   |
|    Initial name:      | T1                        |   |
|                       +---------------------------+   |
|                                                       |
|        +-----------+             +-----------+        |
|        |    OK     |             |  Cancel   |        |
|        +-----------+             +-----------+        |
+-------------------------------------------------------+
```

If you select 5 copies in the X direction with a spacing of 0, you get 5 copies of the widget in a row going to the right, nested together. Selecting copies in the Y direction gives you a column of copies of the widget that grows downward. You can adjust the spacing between the widgets in pixels by setting the appropriate spacing field.

Note that the copy operation gives names that are an incremented count from the initial widget; if the first widget is "T1" as shown above, the copies will be "T2", "T3", and so on.

* The "Grid" option brings up a dialog to allow you to set a "snap grid" for placing widgets on the user interface:

```
+--------------------------------+
|              Grid              |
+--------------------------------+
|                                |
|  Horizontal (x):  [ 1      ]   |
|                                |
|  Vertical (y):    [ 1      ]   |
|                                |
|   +-----------+ +-----------+  |
|   |    OK     | |  Cancel   |  |
|   +-----------+ +-----------+  |
+--------------------------------+
```

If you set the grid coordinates to, say, "4,4", then you can only place the widgets on pixels that are a multiple of 4.

The "Status" option allows you to bring up another Screen Builder panel that gives interactive status on the widgets in the user interface:

```
   +----------------------------------------------+
   | +- Screen ---------------------------------+ |
   | |                                          | |
   | | Cursor Postion:          315 x 101       | |
   | |                                          | |
   | | Grid Size:               1 x 1           | |
   | |                                          | |
   | | No. of widgets:          2               | |
   | |                                          | |
   | | Screen File:                             | |
   | +------------------------------------------+ |
   | +- Current Widget -------------------------+ |
   | |                                          | |
   | | Name:                    Bitmap0         | |
   | |                                          | |
   | | Parent:                  Panel0          | |
   | |                                          | |
   | | Type:                    BITMAP          | |
   | |                                          | |
   | | Relative Origin:         0 x 0           | |
   | |                                          | |
   | | Size:                    100 x 300       | |
   | |                                          | |
   | +------------------------------------------+ |
   +----------------------------------------------+
```

Note that this panel may come up over the main Screen Builder panel and hide it ... just move it out of the way with the mouse if that happens. Anyway, the Status Panel displays information on widgets as you pass the mouse over them.

* Once you have set up a user interface and then saved it, you can incorporate it into a program. However, this requires use of the SYSTEM widget, and has to be delayed to the chapter on that subject.

[7.0] BPLUS (7): Dialogs

v2.4 / 01 feb 99 / greg goebel

* This chapter provides a detailed discussion of BASIC Plus dialogues.

[7.1] OVERVIEW

* Dialogs are essentially a graphics version of the INPUT statement. BPlus includes, as noted in earlier chapters, ten predefined dialogs:

   * ERROR / INFORMATION / QUESTION / WARNING dialog
   * NUMBER and KEYPAD dialogs
   * STRING dialog
   * COMBO dialog
   * LIST dialog
   * FILE dialog

Note that this chapter is largely a rehash of the introduction to dialogs provided in the introductory chapters. While the introductory chapters were intended to say the absolute minimum about the features of BPlus, dialogs are so easy to use that the absolute minimum you need to know about them is close to the maximum you can learn about them.

Program Example: xdialogs.rmb

[7.2] ERROR / INFORMATION / QUESTION / WARNING DIALOGS

* These four dialogs act in the same way, they just possess minor cosmetic differences. Let's start by creating an ERROR dialog:

```
   10   CLEAR SCREEN
   20   DIALOG "ERROR","You're in big trouble!
   30   END
```

Run this program and you get a panel:

```
   +----------------------------+---+
   |              ERROR         | X |
   +----------------------------+---+
   |                                |
   |  STOP   You're in big trouble! |
   |                                |
   +--------------------------------+
   |           +--------+           |
   |           |   OK   |           |
   |           +--------+           |
   +--------------------------------+
```

Note that the dialog is invoked in a single DIALOG statement, followed by the dialog type ("ERROR") and your very own prompt string ("You're in big trouble!"). It stays on the display (and hangs up the program at that line) until you click on the OK key with the mouse, or bang on the keyboard space bar; and then the program can end.

Well, that's nice, but what if there's nobody there to acknowledge the error and the program ultimately has to keep on running? No problem, set a TIMEOUT:

```
   20 DIALOG "ERROR","You in big trouble!";TIMEOUT 5
```

When the dialog pops up, all you have to do is wait for 5 seconds and it will disappear again.

* Given this widget, you can then customize it to a degree. Suppose you want to have the user select from a set of different keys? You can create your own custom keys by using the DIALOG BUTTONS attribute as follows:

```
   10   CLS
   30   DIM S1$(1:2)[10]
   40   INTEGER Btn
   60   S1$(1)="Abort"
   70   S1$(2)="Continue"
   80   DIALOG "ERROR","You in big trouble!,Btn;SET ("DIALOG BUTTONS":S1$(*))
   100  DISP S1$(Btn + 1)
   110  END
```

You use the SET statement to invoke the DIALOG BUTTONS attribute along with a string array S1$, containing the names of buttons you want to use, as the value. This gives you the dialog:

```
                       +---------------------------+---+
                       |             ERROR         | X |
                       +---------------------------+---+
                       |                               |
   little STOP sign --> |  STOP   You in big trouble!    |
                       |                               |
                       +-------------------------------+
                       |  +----------+    +----------+  |
                       |  |  Abort   |    | Continue |  |
                       |  +----------+    +----------+  |
                       +-------------------------------+
```

The next question is: so how do you tell which button was clicked? Well, those of you who are keen-eyed will notice the INTEGER variable "Btn" in this program that is inserted into the DIALOG statement after the prompt string; it contains the index of the button that was clicked. Click on the ABORT button, the dialog vanishes, and the program displays:

```
   ABORT
```

Naturally, click on CONTINUE and the program displays CONTINUE. (Note that in this program I add 1 to "Btn" to get the proper index. I do it to illustrate a minor potential snag: while the array is defined with indexes from 1 to 2, the value returned is 0 or 1 (that is, the dialog always assumes a base index of 0). That's not a bug: it's supposed to work that way.)

* Now that you have two buttons, things get a little more complicated in terms of keyboard handling. You can use the Tab key to skip from one button to the other, and then press the space bar to click the button; but you can also press the Enter key and you'll get an index value back.

But which one? If you have only one button, like "OK", you'll obviously get the index for "OK", but if you have two ... well, that's where the DEFAULT BUTTON attribute comes in. You can set this attribute to the value of the button index you want as the default.

* However, with this new attribute, the line to invoke the dialog is getting too long to fit on the line.

One of the inelegancies of dealing with dialogs is that all options have to be listed on one line -- which if you have a lot of options (and there are plenty to choose from) makes for a very long line.

I dislike RMB lines wider than the screen, and more objectively such long lines are hard to edit and modify; in any case, after you start adding enough junk, you start to look for some ways to clean up the DIALOG statement a little bit.

It's easy enough; you can start putting your parameters and options into variables. For example, I can clean up the previous program as follows:

```
   10    CLEAR SCREEN
   20    DIM T$[16],P$[40],A1$[20],S1$(1:2)[10],A2$[20]
   30    INTEGER Btn
   40    DATA "ERROR","You in big trouble!"
   50    READ T$,P$
   60    DATA "DIALOG BUTTONS","ABORT","CONTINUE","DEFAULT BUTTON"
   70    READ A1$,S1$(*),A2$
   80    !
   90    DIALOG T$,P$,Btn;SET (A1$:S1$(*),A2$:0),TIMEOUT 5
   100   IF Btn=-1 THEN
   110     DISP "NO ENTRY!"
   120   ELSE
   130     DISP S1$(Btn + 1)
   140   END IF
   150   END
```

With this change, I also had enough room to include a TIMEOUT -- which also demanded a small change in the handling of the value of "Btn", since you'll get a "-1" value if the dialog times out.

In this example, I put a dialog that requires a string array as a value (DIALOG BUTTONS) in one string (A1$) and a dialog that requires a number as a value (DEFAULT BUTTON) in another string (A2$). In more general terms, you can put all the attributes that require a numeric value in one string array, and then invoke them along with a matching numeric array:

```
   DATA "ATTRIBUTE 1","ATTRIBUTE 2","ATTRIBUTE 3","ATTRIBUTE 4","ATTRIBUTE 5"
   READ An$(*)
   DATA 12,32,100,6,19
   READ Vn(*)
   ...
   DIALOG Type$,Prompt$,Btn;SET (An$(*):Vn(*))
```

You could also make another set of arrays for the attributes that required string values, say As$(*) and Vs$(*):

```
   DIALOG Type$,Prompt$,Btn;SET (An$(*):Vn(*),As$(*):Vs$(*))
```

-- but you can't mix the two types.

* Now that we've figured out how to compress the DIALOG statement a bit, it's more convenient to invoke new attributes. You can, for the current example of the ERROR dialog, define the following attributes (which are actually common to all dialogs and some are common to all widgets as well):

```
   * BACKGROUND: Specifies the background color of the dialog.
   * DIALOG BUTTONS: Allows user-defined buttons.
   * FONT: Defines the font used in the dialog text.
   * HEIGHT,WIDTH: Dimensions of widget in pixels.
   * JUSTIFICATION: Positioning of text on widget.
   * MAXIMIZABLE: If 1, dialog can be clicked to full screen.
   * MOVABLE: If 1, dialog can be moved around on display with mouse.
   * PEN: Specifies the foreground color.
   * RESIZABLE: If 1, dialog can be changed in size with mouse.
   * RESTORE SCREEN: If 1, display under dialog is not changed.
   * TITLE: Title of dialog (default is dialog name).
   * X,Y: Coordinates of dialog relative to top left display.
```

Note that if the RESTORE SCREEN attribute is set, you need to turn off RESIZABLE, MAXIMIZABLE, and MOVABLE, because the display underneath the dialog is saved only when the dialog is created; move the dialog around or change its size and you'll wipe out parts of your display graphics.

* The INFORMATION, WARNING, and QUESTION dialogs are very similar to the ERROR dialog. Executing:

```
   DIALOG "INFORMATION","This is an INFORMATION dialog!"
```

-- yields:

```
                         +--------------------------------+---+
                         |            INFORMATION         | X |
                         +--------------------------------+---+
                         |                                    |
   circle with strike -->|  0  This is an INFORMATION dialog! |
                         |                                    |
                         +------------------------------------+
                         |             +--------+             |
                         |             |   OK   |             |
                         |             +--------+             |
                         +------------------------------------+
```

* Executing:

```
   DIALOG "WARNING","This is a WARNING dialog!"
```

-- yields:

```
                          +---------------------------------+---+
                          |               WARNING           | X |
                          +---------------------------------+---+
                          |                                     |
   "!" inside triangle -->|  !    This is a WARNING dialog!     |
                          |                                     |
                          +-------------------------------------+
                          |             +--------+              |
                          |             |   OK   |              |
                          |             +--------+              |
                          +-------------------------------------+
```

* Executing:

```
   DIALOG "QUESTION","This is a QUESTION dialog!",Btn; TIMEOUT 5
```

-- yields:

```
                          +---------------------------------+---+
                          |              QUESTION           | X |
                          +---------------------------------+---+
                          |                                     |
   "?" inside person's -->|  ?    This is a QUESTION dialog!    |
     head in profile      |                                     |
                          +-------------------------------------+
                          |      +--------+    +--------+       |
                          |      |  Yes   |    |   No   |       |
                          |      +--------+    +--------+       |
                          +-------------------------------------+
```

Note that you need to specify a return variable ("Btn" in this example) to store the return value (0="Yes", 1="No", -1=TIMEOUT).

All other dialogs behave like supersets of these four: they have the same capabilities and attributes, and offer other features as well.

[7.3] NUMBER AND KEYPAD DIALOGS

* The NUMBER dialog allows entry of a number:

```
   DIALOG "NUMBER","What is your age?",Btn;RETURN ("VALUE":N)
```

This gives:

```
   +-------------------------+---+
   |            NUMBER       | X |
   +-------------------------+---+
   |                             |
   |      What is your age?      |
   |                             |
   |  +-----------------------+  |
   |  |                       |  |
   |  +-----------------------+  |
   |                             |
   +-----------------------------+
   |   +--------+   +--------+   |
   |   |   OK   |   | Cancel |   |
   |   +--------+   +--------+   |
   +-----------------------------+
```

Note that you can specify different numeric input formats such as REAL, SHORT INTEGER, BINARY, OCTAL, and HEX using the FORMAT attribute; the NUMBER dialog will not allow you to enter incorrect characters, though it may well exceed the desired numeric range. You can also specify the REAL NOTATION for REAL inputs -- it can be set to FIXED, SCIENTIFIC, or ENGINEERING format -- and the number of fractional digits for REAL inputs -- using the REAL RESOLUTION attribute.

* The KEYPAD dialog:

```
   DIALOG "KEYPAD","What is your age?",Btn;RETURN ("VALUE":N)
```

-- performs exactly the same function, but uses a calculator-keypad format:

```
   +-------------------------+---+
   |          KEYPAD         | X |
   +-------------------------+---+
   |                             |
   |      What is your age?      |
   |                             |
   |  +-----------------------+  |
   |  |                       |  |
   |  +-----+-----+-----+-----+  |
   |  |  A  |  B  |  C  |  D  |  |
   |  +-----+-----+-----+-----+  |
   |  |  7  |  8  |  9  |  E  |  |
   |  +-----+-----+-----+-----+  |
   |  |  4  |  5  |  6  |  F  |  |
   |  +-----+-----+-----+-----+  |
   |  |  1  |  2  |  3  | <-- |  |
   |  +-----+-----+-----+-----+  |
   |  |  -  |  0  |  .  | --> |  |
   |  +-----+-----+-----+-----+  |
   |  | CLR | DEL | INS | ENT |  |
   |  +-----+-----+-----+-----+  |
   |                             |
   +-----------------------------+
   |   +--------+   +--------+   |
   |   |   OK   |   | Cancel |   |
   |   +--------+   +--------+   |
   +-----------------------------+
```

The only programmatic difference between the KEYPAD and NUMBER dialogs is that you can set KEY BACKGROUND and KEY PEN colors on the KEYPAD.

[7.4] STRING DIALOG

* A STRING dialog is much like a NUMBER dialog, but allows entry of any text. The STRING dialog can be created as follows:

```
   10   CLEAR SCREEN
   30   INTEGER Btn
   40   DIM P$[40],V$[255]
   50   P$="This is a STRING dialog!"
   60   DIALOG "STRING",P$;RETURN ("VALUE":V$)
   70   DISP Btn,V$
   80   END
```

This yields:

```
   +--------------------------------+---+
   |              STRING            | X |
   +--------------------------------+---+
   |                                    |
   |      This is a STRING dialog!      |
   |                                    |
   |    +--------------------------+    |
   |    |                          |    |
   |    +--------------------------+    |
   +------------------------------------+
   |      +--------+    +--------+      |
   |      |   OK   |    | Cancel |      |
   |      +--------+    +--------+      |
   +------------------------------------+
```

If you move the mouse cursor or tab into the text field below the prompt, you can input text into the widget, which is returned through the V$ string.

[7.5] COMBO DIALOG

* The COMBO dialog combines a string-input field like that of the STRING dialog with a list (defined by a string array) that you can pull down with a mouse. For example, the following program:

```
   10    DIM L$(1:5)[25],S$[100],T$[100]
   20    DATA "Ice Cream","Cake","Pie","Cookies","Fruit"
   30    DATA "What would like for dessert?"
   40    READ L$(*),S$
   50    DIALOG "COMBO",S$,Btn;SET ("ITEMS":L$(*)),RETURN ("TEXT":T$)
   60    IF Btn=0 THEN
   70      DISP T$
   80    ELSE
   90      DISP "Cancelled!"
   100   END IF
   320   END
```

This gives the dialog:

```
   +--------------------------------+---+
   |              COMBO             | X |
   +--------------------------------+---+
   |                                    |
   |  What would you like for dessert?  |
   |                                    |
   |    +-----------------------+-+     |
   |    |                       | |     |
   |    +-----------------------+-+     |
   |    |Ice Cream              |^|     |
   +----|Cake                   | |-----+
   |    |Pie                    | |     |
   |    |Cookies                | |     |
   |    |Fruit                  |v|     |
   |    +-----------------------+-+     |
   +------------------------------------+
```

You can also get an index into the list items back from the COMBO dialog, but this is useless if the user simply types text in. It's easier to just get the input using the TEXT attribute.

[7.6] LIST DIALOG

* The LIST dialog allows you to select one or more text items from a list, using a mouse or keyboard inputs, much like the list provided by the COMBO dialog. However, the LIST dialog only returns a list index -- or, if put in MULTISELECT mode, an INTEGER array that matches the items in the list and is set to 1 for each item selected.

You define the list as a string array, and pass the string array to the dialog using the ITEMS attribute; then, when the user selects an item from the list and exits the dialog, you get the index corresponding to that list entry using the SELECTION attribute, as the following simple program illustrates:

```
   10   CLEAR SCREEN
   20   INTEGER Btn,Sel
   30   DIM L$(1:7)[20],P$[40]
   40   DATA "Sunday","Monday","Tuesday","Wednesday"
   50   DATA "Thursday","Friday","Saturday","This is a LIST dialog!"
   60   READ L$(*),P$
   80   DIALOG "LIST",P$;SET ("ITEMS":L$(*)),RETURN ("SELECTION":Sel)
   90   DISP Btn, Sel
   100  END
```

This yields:

```
+-------------------------------+---+
|              LIST             | X |
+-------------------------------+---+
|                                   |
|       This is a LIST dialog!      |
|                                   |
|          +------------+-+         |
|          | Sunday     |^|         |
|          | Monday     | |         |
|          | Tuesday    | |         |
|          | Wednesday  | |         |
|          | Thursday   |v|         |
|          +------------+-+         |
|                                   |
+-----------------------------------+
|      +--------+    +--------+     |
|      |   OK   |    | Cancel |     |
|      +--------+    +--------+     |
+-----------------------------------+
```

Click on "Sunday" and then click on one of the buttons -- it doesn't matter which one, you get an value back just the same -- and you'll get a SELECTION value of 0. (Note that as far as the return value of SELECTION is concerned, the array index returned always assumes a first array index is 0, not 1.)

* The following example program, in contrast, shows how to use the MULTISELECT attribute:

```
10    CLEAR SCREEN
20    INTEGER Btn,Sel(1:10),N,V(1:2)
30    DIM L$(1:10)[20],P$[40],B$[20],A$(1:2)[16],S$[16]
40    !
50    DATA "TIME","NEWSWEEK","SCIENCE","SPORTS ILLUSTRATED","SMITHSONIAN"
60    DATA "BUSINESS WEEK","FORTUNE","MAD","KEYBOARD","ESQUIRE"
70    READ L$(*)
80    !
90    DATA "MULTISELECT","COLUMNS",1,30,"SELECTION"
100   READ A$(*),V(*),S$
110   P$="Which magazines do you read?"
120   DIALOG "LIST",P$,Btn;SET ("ITEMS":L$(*),A$(*):V(*)),RETURN (S$:Sel(*))
130   DISP USING "2D,5X,10(D,X)";Btn,Sel(*)
140   END
```

This yields:

```
   |                                    |
   |    Which magazines do you read?    |
   |                                    |
   |  +--------------------------+-+    |
   |  | TIME                     |^|    |
   |  | NEWSWEEK                 | |    |
   |  | SCIENCE                  | |    |
   |  | SPORTS ILLUSTRATED       | |    |
   |  | SMITHSONIAN              |v|    |
   |  +--------------------------+-+    |
   |                                    |
```

If you click on TIME, SCIENCE, and (after scrolling up the list) KEYBOARD, then click on OK, the program displays:

```
  0    1 0 1 0 0 0 0 0 1 0
```

[7.7] FILE DIALOG

* The FILE dialog allows you to select a file from a list of files; you can specify the directory, file type, and filename range (via wildcards) that will appear in the alphanumerically-sorted list of files. A scroll bar will be present to allow the user to scroll through the resulting file list. Despite its seeming complexity, it's easy to make a file dialog:

```
   10  CLEAR SCREEN
   20  INTEGER Btn
   30  DIM S$[64]
   40  DIALOG "FILE","This is a FILE dialog!",Btn;RETURN ("SELECTION":S$)
   50  DISP Btn, S$
   60  END
```

This yields:

```
   +---------------------------------------+---+
   |                   FILE                | X |
   +---------------------------------------+---+
   |                                           |
   |           Please select a file:           |
   |                                           |
   | +---------------------------------------+ |
   | |                                       | |
   | | Current directory:       Directories: | |
   | | +---------------------+  +--------+-+ | |
   | | |                     |  |        | | | |
   | | +---------------------+  +--------+-+ | |
   | |                                       | |
   | | File wildcard:           Files:       | |
   | | +---------------------+  +--------+-+ | |
   | | |                     |  |        | | | |
   | | +---------------------+  +--------+-+ | |
   | |                                       | |
   | | File selection:                       | |
   | | +---------------------------------+   | |
   | | |                                 |   | |
   | | +---------------------------------+   | |
   | +---------------------------------------+ |
   +-------------------------------------------+
   |         +--------+     +--------+         |
   |         |   OK   |     | Cancel |         |
   |         +--------+     +--------+         |
   +-------------------------------------------+
```

The user selects a file from the list or types one in; it is returned through the S$ array. There are other attributes specific to the fields in the dialog -- DIRECTORY, PATTERN, SELECTION -- that return a string that allow you to set or read the fields (the user can change these fields interactively).

Note that you have to enter volume and pathname descriptions as defined under the target platform. If you are using a FILE dialog on a PC with a Measurement Coprocessor, for example, you could not successfully enter a file as:

```
   A:\FILES\MYFILE.TXT
```

You would need to enter it as:

```
   \FILES\MYFILE.TXT:DOS,A
```

However, the standard PC format would work with HBW.

[7.8] DIALOG ATTRIBUTES

* It is difficult to describe the attributes of dialogs in complete detail, since they often have many more attributes than are useful. This is because all the dialogs -- except for ERROR, INFORMATION, QUESTION, and WARNING -- are equivalent to widgets: there is a NUMBER dialog and a NUMBER widget, a STRING dialog and a STRING widget, and so on.

Not surprisingly, both the dialog and widget forms of, say, NUMBER use the same code -- and so share the same attributes. However, in many cases the widget has attributes that would be impractical or impossible to use in a dialog.

So, while technically speaking a NUMBER dialog uses the same attributes as a NUMBER widget, it's misleading to discuss all these attributes in the context of a dialog, and so some judgement must be exercised in the following descriptions of dialog attributes. If you are adventurous and want to consider other attributes, please consult the chapters on the corresponding widgets.

In any case, all dialogs support the following useful attributes:

BACKGROUND	DIALOG BUTTONS	FONT	HEIGHT	JUSTIFICATION
MAXIMIZABLE	MOVABLE	PEN	RESIZABLE	RESTORE SCREEN
TITLE	WIDTH	X	Y	

[7.9] ERROR/INFORMATION/QUESTION/WARNING DIALOG ATTRIBUTE REFERENCE

* These dialogs only use the default dialog attributes.

[7.10] NUMBER & KEYPAD DIALOG ATTRIBUTE REFERENCE (2.0 & LATER ONLY)

* These two widgets include the following attributes:

COLUMNS	Type:	numeric
	Values:	non-negative
	Default:	20

Width of widget in character cells.

```
FORMAT                 Type:    string
                       Values:  "REAL" or "SHORT INTEGER" or
                                "BINARY" or "OCTAL" or "HEX"
                       Default: "REAL"
```

Specifies numeric format.

```
FORMAT LENGTH          Type:    numeric
                       Values:  0 to 328
                       Default: 0
```

Defines the number of characters used to display the number, with zeroes padded on the left; intended for use with BINARY, OCTAL, or HEX values.

```
INCREMENT              Type:    numeric
                       Values:  non-negative; 0 = "don't round"
                       Default: 0
```

VALUE is "rounded" to the nearest specified INCREMENT.

```
KEY BACKGROUND         Type:    numeric
                       Values:  legal PEN numbers (0-255)
KEYPAD only            Default: depends on system defaults
```

Specifies color of key background.

```
KEY PEN                Type:    numeric
                       Values:  legal PEN numbers (0-255)
KEYPAD only            Default: depends on system defaults
```

Specifies color of key font.

```
MAXIMUM                Type:    numeric
                       Values:  in the range between current
                                FORMAT's maximum and minimum.
                       Default: MAXREAL
```

Maximum value widget will accept.

```
MAXIMUM LENGTH         Type:    numeric
                       Values:  0 to 328
                       Default: 328
```

Maximum number of characters operator can enter; however, when the widget reformats the operator entry, the text length may be longer.

MINIMUM	Type:	numeric
	Values:	in the range between current FORMAT's maximum and minimum.
	Default:	MAXREAL

Maximum value widget will accept.

REAL NOTATION	Type:	string
	Values:	"FIXED" or "SCIENTIFIC" or "ENGINEERING"
	Default:	"SCIENTIFIC"

Specifies real-number display format (only applicable to "FORMAT":"REAL")

REAL RESOLUTION	Type:	numeric
	Values:	0 to 16
	Default:	12

Number of fractional digits displayed (only applicable to "FORMAT":"REAL").

SHOW EDIT	Type:	numeric
	Values:	0 or 1
KEYPAD only	Default:	1 (ON)

Controls display of editing field.

TEXT LENGTH	Type:	numeric (RETURN only)
	Values:	integer

Gives length of text string the user's typed in.

VALUE	Type:	numeric or string
	Values:	as allowed by "FORMAT", "MINIMUM", and "MAXIMUM"
	Default:	0

Value returned by widget. "BINARY", "OCTAL", "HEX" are all in two's complement (no "-" sign allowed).

The default dialog attributes also apply.

[7.11] STRING DIALOG ATTRIBUTE REFERENCE

* The STRING dialog includes the following attributes:

```
AUTOSCROLL              Type:    numeric
                        Values:  0 or 1
                        Default: 0 (OFF)
```

If 1, characters will be scrolled if there are more than can be displayed.

```
COLUMNS                 Type:    numeric
                        Values:  positive integer
                        Default: 40
```

Gives size of STRING widget in terms of the number of characters that can fit into it.

```
CONVERT CASE            Type:    string
                        Values:  "NONE" or "UPSHIFT" or "DOWNSHIFT"
2.0 & later only        Default: "NONE"
```

Converts all entered text to upper or lower case.

```
MAXIMUM LENGTH          Type:    numeric
                        Values:  positive integer
                        Default: 40
```

Maximum length of text allowed in widget.

```
PASSWORD CHARACTER      Type:    string
                        Values:  "" (null string) or any character
2.0 & later only        Default: "" (null string)
```

When set to non-null, all entered characters are displayed as the specified password character.

```
SCROLLBARS              Type:    numeric
                        Values:  0 or 1
2.0 & later only        Default: 0 (OFF)
```

When 1, horizontal or vertical scrollbars appear when the widget contains more text than can be displayed.

```
VALUE                  Type:    string
                       Values:  any legal string
                       Default: "" (null string)

Returns string typed in by user; can also be given an initial value.
```

The default dialog attributes also apply.

[7.12] COMBO DIALOG ATTRIBUTE REFERENCE (2.0 & LATER ONLY)

* The COMBO dialog has the following attributes:

```
COLUMNS                Type:    numeric
                       Values:  0 to 32,767
                       Default: depends on system defaults

Width of widget in character cells.
```

```
DROPDOWN BUTTON        Type:     numeric
                       Values:   0 or 1
                       Default:  1 (ON)

Specifies the presence of a button that controls the visiblity of the
listbox; without the button, the listbox is always displayed.
```

```
EDITABLE               Type:     numeric
                       Values:   0 or 1
                       Default:  1 (ON)
Sets up combo box text field as an edit field or a display.
```

```
HEIGHT                 Type:    numeric
                       Values:  0 to 32,767
                       Default: depends on system defaults

On a SET, this attribute includes height of list box, whether visible or
not; on a RETURN, it includes list box only if "DROPDOWN BUTTON" is 0.
```

```
ITEM COUNT             Type:    numeric (read only)
                       Values:  0 to 32,767
                       Default: 0
```

Returns number of items in list box.

```
ITEMS                  Type:    string array
                       Values:  any
```

Values to be displayed in the list box.

```
LIST BACKGROUND        Type:    numeric
                       Values:  legal PEN numbers (0-255)
                       Default: depends on system defaults
```

Sets color of list box background (ignored on one-plane mono monitors).

```
LIST PEN               Type:    numeric
                       Values:  legal PEN numbers (0-255)
                       Default: depends on system defaults
```

Specifies pen color of list box text (ignored on one-plane mono monitors).

```
SCROLLBAR              Type:     numeric
                       Values:   0 or 1
                       Default:  1 (ON)
```

Specifies whether the list box has a scroll bar.

```
SELECTION              Type:    numeric
                       Values:  list index (0 to ITEM COUNT)
                                -1 if nothing selected
                       Default: -1
```

Index of selected item; the index always starts at 0.

```
SHOW LIST              Type:    numeric
                       Values:  0 or 1
                       Default: 0 (NO)
```

Specifies whether list box is shown or not.

```
TEXT                   Type:    string
                       Values:  any
                       Default: "" (null string)
```

Returns string in edit field.

The default dialog attributes also apply.

[7.13] LIST DIALOG ATTRIBUTE REFERENCE

* The LIST dialog box has the following attributes:

```
COLUMNS                 Type:    numeric
                        Values:  positive integer
                        Default: 10
```

Gives length of text in dialog.

```
ITEMS                   Type:    string array
                        Values:  any
                        Default: zero-element array
```

Stores strings to be placed into LIST dialog.

```
MULTISELECT             Type:    numeric
                        Values:  0 or 1
                        Default: 0 (single-select)
```

Allows only one (0) or several (1) list items to be selected.

```
ROWS                    Type:    numeric
                        Values:  positive integer
                        Default: 3
```

Gives number of rows of text in dialog.

```
SELECTION               Type:    numeric variable or numeric array
                        Values:  boolean flags for ITEMS array
                        Default: zeroes
```

If SELECTION is given a numeric variable, then the variable contains the index of the ITEMS list that the user has selected (it's -1 if nothing has been selected). If SELECTION is an array, then it can show the selection of all items in the dialog. Note that the array index returned always assumes a base index of 0.

```
TOP ITEM              Type:    numeric
                      Values:  legal index into ALTERNATIVES array
                      Default: index of first item

Item placed at top of list in visible window.
```

The default dialog attributes also apply (see section 1).

[7.14] FILE DIALOG ATTRIBUTE REFERENCE

* The attributes for the FILE dialog are as follows:

```
DIRECTORY             Type:    string
                      Values:  valid pathname
                      Default: current directory

Gives directory to be listed.
```

```
PATTERN               Type:    string
                      Values:  legal wildcard pattern
                      Default: * (all files)

Specifies which files will be displayed.
```

```
SELECTION             Type:    string
                      Values:  any file in the current directory
                      Default: "" (null string)

Specifies selected file.
```

The default dialog attributes also apply (see section 1).

[8.0] BPLUS (8): Fundamental Widgets -- PANEL, PUSHBUTTON, & LABEL

v2.4 / 01 feb 99 / greg goebel

* This chapter describes the PANEL, PUSHBUTTON, & LABEL widgets in detail.

[8.1] OVERVIEW

* The BPlus widget set includes many widgets, and it can be hard to figure out where to start. There are, however, three simple widgets that are not only easy to understand, but which can be used to build very sophisticated user interfaces that illustrate the essential points of design with BPlus:

   * PANEL widget
   * LABEL widget
   * PUSHBUTTON widget

The PANEL widget creates a rectangle you can put on the display as a place to hang other widgets. Such a PANEL can be defined as a parent widget, with all the widgets placed on the PANEL defined as child widgets -- so that when the PANEL is hidden, the child widgets are hidden also, or when the PANEL is moved, the child widgets are moved as well. (You can have multiple PANELs on a display, or have child PANELs within a another PANEL.)

You can think of PANELs as displays that can be created by a program to present parts of a user interface; these displays contain arrangements of component widgets that provide user input or output.

The simplest of these component widgets are the PUSHBUTTON and LABEL widgets. The PUSHBUTTON widget creates a ... well, button on the display; you click on this button with a mouse, and an event occurs that you can trap with an ON EVENT statement to perform an action. The PUSHBUTTON is a simple input device, and the LABEL widget is a correspondingly simple output device: all it does is display a string.

This chapter provides elementary examples of how these widgets are used, then elaborates on them in detail, constructs an example using them, and describes some advanced PANEL and PUSHBUTTON features added in BPlus 2.0. The chapter finishes with reference information on each of the widgets.

[8.2] PANEL, SEPARATOR, PUSHBUTTON, & LABEL WIDGETS -- FUNDAMENTALS

* You create a PANEL with:

```
   ASSIGN @P TO WIDGET "PANEL"
```

Generally a PANEL is designed to be a parent to other widgets (including other PANELs if so needed). As a level-0 parent, it can be MOVABLE, MAXIMIZABLE, MINIMIZABLE, RESIZABLE, and can be given a TITLE; just like other widgets, you can set its origin and dimensions, and make it visible or invisible. (The capability of making a PANEL visible or invisible is particularly handy, since if you make it disappear, all its child widgets disappear, too; you can make the PANEL invisible when you create it, populate it with child widgets, and then make the whole assemblage visible at one time.)

The PANEL also can generate two events. A REPAINT event occurs when something happens that requires the PANEL to be repainted -- such as a dialog popping over the top of it. A RESIZED event occurs when the PANEL is RESIZED.

* The SEPARATOR widget provides a useful accessory to a PANEL; the SEPARATOR doesn't do a great deal, it just draws a line across the display. You can set a HEIGHT on the SEPARATOR to give it a more distinctive appearance. (You can also change its default ORIENTATION from HORIZONTAL to VERTICAL.) Otherwise, this is the most unexciting of widgets.

* The PUSHBUTTON widget is simple, consisting of a box that contains a LABEL and for which an event called ACTIVATED can be set up. For example, if you execute the statements:

```
   ASSIGN @Btn TO WIDGET "PUSHBUTTON"; PARENT @Panel, SET ("LABEL":"Push Me")
   ON EVENT @Btn, "ACTIVATED" GOSUB Handler
```

-- then when you click on the "Push Me" button, the "Handler" routine will be executed.

* The LABEL widget is just a box that you can use to display a string or numeric value. For example:

```
   ASSIGN @Lbl TO WIDGET "LABEL"; PARENT @Panel, SET ("VALUE":"Hi There")
```

-- simply displays a box containing the string "Hi There". This seems pretty dull too, except that you can specify a number of different justifications of the information in the LABEL, and you can also give it a numeric variable or value, instead of a string -- it's "smart" enough to accept either.

Note a subtle distinction here: many widgets have a LABEL attribute that allows you to specify some describing text, but the LABEL widget itself uses a VALUE attribute to display its information -- and has no LABEL attribute.

* Customers often want to generate tabular text displays with BPlus; the LABEL widget can be used to do the job.

It's easy, in principle. You can create a multi-line LABEL widget; while you can only give the widget a single string of text as a VALUE attribute parameter, you can insert line feed characters (using CHR$(10)) into the string to separate it into multiple lines. The following program shows how:

```
10    CLEAR SCREEN
20    INTEGER N
30    DIM S$[256]
50    ASSIGN @L TO WIDGET "LABEL";SET ("VISIBLE":0)
60    CONTROL @L;SET ("COLUMNS":26,"ROWS":8,"TITLE":"Label Test")
80    CONTROL @L;SET ("JUSTIFICATION":"TOP,LEFT","WORD WRAP":1)
90    CONTROL @L;SET ("SYSTEM MENU":"Quit")
100   !
110   FOR N=1 TO 8
120     S$=S$ & " ITEM " &VAL$(N)& ":       VALUE " &VAL$(N)& " "
130     IF N < 8 THEN S$=S$ & CHR$(10)
140   NEXT N
150   CONTROL @L;SET ("VALUE":S$,"VISIBLE":1)
160   ON EVENT @L,"SYSTEM MENU" GOTO Finis
170   LOOP
180   END LOOP
190   !
200  Finis: !
210   ASSIGN @L TO *
220   END
```

This results in the panel:

```
+---+------------------------+---+
| = |      Label Test        | X |
+---+------------------------+---+
| ITEM 1:             VALUE 1  |
| ITEM 2:             VALUE 2  |
| ITEM 3:             VALUE 3  |
| ITEM 4:             VALUE 4  |
| ITEM 5:             VALUE 5  |
| ITEM 6:             VALUE 6  |
| ITEM 7:             VALUE 7  |
| ITEM 8:             VALUE 8  |
+--------------------------------+
```

Note that you have to specify the appropriate text justification to get this to come out right:

```
   80 CONTROL @L;SET ("JUSTIFICATION":"TOP,LEFT","WORD WRAP":1)
```

Note also that the ROWS and COLUMNS attributes will not give you exactly the number of text rows and columns you need in this case; that's because this program creates the LABEL as a stand-alone widget, and so it has a title bar and border, which takes up space. If the LABEL widget was the child of a PANEL widget, it would fit exactly that many rows and columns of text inside.

[8.3] PANEL / SEPARATOR / PUSHBUTTON / LABEL EXAMPLE

* These are very simple-minded widgets, but you can do a great deal with them, as the following example program shows. The example allows you to play the game of rock / paper / scissors with the computer, where you and the computer both guess one of three items -- rock, paper, or scissors -- with one differing guess winning over the other according to the rules:

   * Rock breaks scissors, rock wins.
   * Scissors cuts paper, scissors wins.
   * Paper covers rock, paper wins.

This is actually a pretty dumb game -- the computer's guesses are random, you can't outguess it, and over the long run you get the same number of outcomes, no matter how hard you try -- but it does show how to build an appealing user interface with these simple widgets. This user interface looks like this:

```
   +---+-----------------------------------+
   | = |   Rock / Paper / Scissors Game    |
   +---+-----------------------------------+
   | +-----------------------------------+ |
   | |           Make a guess!           | |
   | +-----------------------------------+ |
   |  ===================================  |
   | +----------+ +----------+ +----------+ |
   | |   ROCK   | | SCISSORS | |   PAPER  | |
   | +----------+ +----------+ +----------+ |
   |  ===================================  |
   | +----------+ +----------+ +----------+ |
   | |    ME    | |    YOU   | |   TIE    | |
   | +----------+ +----------+ +----------+ |
   | |    0     | |     0    | |    0     | |
   | +----------+ +----------+ +----------+ |
   +---------------------------------------+
```

Note how the widgets are set up on a regular grid, with variables designating the row and column coordinates of each widget. The program defines the coordinates and sizes of the widgets in an interlocking way, so that if one widget is adjusted, the others will compensate.

Program Example: xscissor.rmb

* Both the PANEL and PUSHBUTTON widgets had some interesting features added to them in BPlus 2.0.

As noted in an earlier chapter, the PANEL had a RESIZE CONTROL added to it that allows it to automatically resize its child widgets. The default setting is NONE, but if you set it to RESIZE CHILDREN, all the child widgets are automatically rescaled if you resize the PANEL -- this feature is used in some demo programs in this document).

If you set RESIZE CONTROL to SCROLLABLE, then if you resize the PANEL so that it is smaller than a defined "scroll area" the scroll bars will appear to allow you to scroll the user interface under the PANEL ... there is a set of related attributes that allow you to specify the scroll increments in pixels (SCROLL WIDTH UNITS and SCROLL HEIGHT UNITS), the corner of the PANEL to which the scrolling is relative (SCROLL JUSTIFICATION), current scroll origin (SCROLL X ORIGIN and SCROLL Y ORIGIN), and the size of the scroll area (SCROLL WIDTH and SCROLL HEIGHT).

The scrolling attributes have some odd interactions that are a little tricky to understand. SCROLL WIDTH UNITS and SCROLL HEIGHT UNITS specify the number of pixels to scroll the PANEL work area, which is easy enough to grasp; what is a little trickier to understand is that SCROLL WIDTH and SCROLL HEIGHT are not in pixels -- but in multiples of SCROLL WIDTH UNITS and SCROLL HEIGHT UNITS.

For example, if you set SCROLL WIDTH UNITS to 20 (pixels) and then set SCROLL WIDTH to 100, the width over which the PANEL interior scrolls is not 100 pixels -- but instead 2000 pixels (or 20 * 100). Note that if the SCROLL WIDTH or SCROLL HEIGHT evaluates to over 32,767 pixels, you'll get an error message ... there is an example program in the section on the BITMAP widget that shows how to handle scrolling.

* Another added PANEL feature that isn't useful in the strict sense but is kind of fun is the BACKGROUND BITMAP attribute; this allows you to specify a bitmap-graphics file to "tile" the PANEL; either an MS-Windows .BMP file or X11 .XWD bitmap file can be specified.

Program Example: xbkbits.rmb

* The PUSHBUTTON widget only had one major feature added in Bplus 2.0: the ability to support multiple text labels, using the LABELS attribute. If you specify a string array as a parameter to LABELS, and specify the number of elements in the array with the STATES attribute, the PUSHBUTTON will cycle

through the different labels. You can query the current PUSHBUTTON label setting with the STATE attribute, as the following program shows:

```
   10    ! ************************************************************
   20    !
   30    ! Label States Example Program
   40    !
   50    ! This little program allows you examine the state of a
   60    ! PUSHBUTTON widget.
   70    !
   80    ! ************************************************************
   90    !
   100   DIM L$(1:3)[50]
   110   INTEGER D(1:4),Dw,Dh,Bh,Bw,Bx,By,N
   130   DATA "Label ONE","Label TWO","Label THREE"
   140   READ L$(*)
   150   !
   160   GESCAPE CRT,3;D(*)
   170   Dw=D(3)-D(1)
   180   Dh=(D(4)-D(2))
   190   Bw=128
   200   Bh=Bw/2
   210   Bx=(Dw-Bw)/2
   220   By=(Dh-Bh)/2
   230   !
   240   ASSIGN @Btn TO WIDGET "PUSHBUTTON";SET ("SYSTEM MENU":"Quit")
   250   CONTROL @Btn;SET ("X":Bx,"Y":By,"WIDTH":Bw,"HEIGHT":Bh)
   260   CONTROL @Btn;SET ("LABELS":L$(*),"STATES":3)
   270   !
   280   ON EVENT @Btn,"ACTIVATED" GOSUB Handler
   290   ON EVENT @Btn,"SYSTEM MENU" GOTO Finis
   300   LOOP
   310   END LOOP
   315   STOP
   320   !
   330  Handler: !
   340   STATUS @Btn;RETURN ("STATE":N)
   350   DISP "State = "&VAL$(N)
   360   RETURN
   370   !
   380  Finis: !
   390   ASSIGN @Btn TO *
   400   END
   410   !
   420   ! ******************** That's All, Folks! ******************
```

Program Example: xstate.rmb

[8.4] PANEL WIDGET ATTRIBUTE REFERENCE

* The PANEL widget supports the following attributes:

```
BACKGROUND BITMAP      Type:    string
2.0 & later only       Values:  valid filename
```

Specifies filename of bitmap to tile background instead of pen.

```
SCROLL HEIGHT          Type:    numeric
                       Values:  1 to 32,767
2.0 & later only       Default: 10 * INITIAL WIDTH
```

Gives the virtual client area height in increments of SCROLL HEIGHT UNITS. SCROLL HEIGHT * SCROLL HEIGHT UNITS must be <= 32767.

```
SCROLL HEIGHT UNITS    Type:    numeric
                       Values:  1 to 32,767
2.0 & later only       Default: 1
```

Gives pixels per unit of vertical scrolling.

```
SCROLL JUSTIFICATION   Type:    string
                       Values:  "TOP" or "BOTTOM"
2.0 & later only       Default: "TOP"
```

When SIZE CONTROL is set to SCROLLABLE; when the current widget height isn't an integral of SCROLL HEIGHT UNITS, specifies where to justify & pad as necessary.

```
SCROLL WIDTH           Type:    numeric
                       Values:  1 to 32,767
2.0 & later only       Default: 10 * initial width
```

Gives the virtual client area height in increments of SCROLL HEIGHT UNITS. SCROLL WIDTH * SCROLL WIDTH UNITS must be <= 32767.

```
SCROLL WIDTH UNITS     Type:    numeric
                       Values:  1 to 32,767
2.0 & later only       Default: 1
```

Gives pixels per unit of horizontal scrolling.

```
SCROLL X ORIGIN        Type:    numeric
```

```
                       Values:  -32,768 to 0
2.0 & later only       Default: 0
```

Gives the virtual client area origin relative to upper left of displayed area in increments of SCROLL WIDTH UNITS.

```
SCROLL Y ORIGIN        Type:    numeric
                       Values:  -32,768 to 0
2.0 & later only       Default: 0
```

Gives the virtual client area origin relative to upper left of displayed area in increments of SCROLL HEIGHT UNITS.

```
SIZE CONTROL           Type:    string
                       Values:  "NONE" or "SCROLLABLE" or
                                "RESIZE CHILDREN"
2.0 & later only       Default: "NONE"
```

This attribute specifies what the PANEL does with the client area when the PANEL is resized: NONE specifies no action, SCROLL HEIGHT causes scroll bars to appear if the client area is greater than the displayed size, and RESIZE CHILDREN causes children to be automatically resized.

The default attributes and events apply (the "0" indicates "level-0 only"):

```
BACKGROUND          BORDER             CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC         INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)    MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER     SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                  SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                  TITLE (0)        USER DATA
VERSION            VISIBLE             WIDTH            X,Y
```

The PANEL widget provides two unique events:

REPAINT	Occurs if anything happens that requires repainting.
RESIZED	Occurs if PANEL is RESIZEd.

[8.5] SEPARATOR WIDGET ATTRIBUTE REFERENCE

* The SEPARATOR has few attributes:

```
ORIENTATION             Type:    string
                        Values:  "UP" or "DOWN" or "LEFT" or "RIGHT"
                                 "VERTICAL" or "HORIZONTAL"
                        Default: "UP"
```

Specifies ORIENTATION of the SEPARATOR.

```
PEN                     Type:    numeric
                        Values:  valid pen number
                        Default: varies
```

Specifies color of SEPARATOR.

The following default attributes apply (this cannot be a level-0 widget):

BACKGROUND	BORDER	HEIGHT	INSIDE HEIGHT
INSIDE WIDTH	USER DATA	VISIBLE	WIDTH
X	Y		

There are no events associated with this widget.

[8.6] PUSHBUTTON WIDGET ATTRIBUTE REFERENCE

* The PUSHBUTTON widget includes the following attributes:

```
COLUMNS                 Type:    numeric
                        Values:  positive integer
                        Default: 10
```

Gives size of PUSHBUTTON in terms of number of characters it can contain.

```
FONT                   Type:    string
                       Values:  legal font-file name
                       Default: depends on system defaults
```

Gives name of font to be used widget LABEL string.

```
LABEL                  Type:    string
                       Values:  any string
                       Default: "" (null string)
```

Specifies string to be displayed on pushbutton.

```
LABELS                 Type:    string array
2.0 & later only       Values:  any
```

This attribute allows for multiple button states; you can set multiple labels by loading them with a string array. Note you also have to use the STATES attribute to set the number of button states.

```
PANEL DEFAULT          Type:    numeric
                       Values:  0 or 1
                       Default: 0 (not default)
```

If 1, the PUSHBUTTON is the default pushbutton for a PANEL -- pressing RETURN anywhere in the panel will activate the button.

```
PEN                    Type:    numeric
                       Values:  valid pen number
                       Default: depends on system defaults
```

Specifies pen color of PUSHBUTTON's font.

```
SENSITIVE              Type:    numeric
                       Values:  0 or 1
                       Default: 1 (is SENSITIVE)
```

If 0, the PUSHBUTTON is still visible but won't respond to user input.

```
STATE                  Type:    numeric
                       Values:  0 to states - 1
2.0 & later only       Default: 0
```

Represents the current state of the button. If set, will change the label to the string represented by an index into the LABELS string array. The index is assumed to be base 0.

```
STATES                  Type:    numeric
                        Values:  0 to states - 1
2.0 & later only        Default: 0
```

Specifies the number of states the button has.

```
TAB STOP                Type:    numeric
                        Values:  0 or 1
                        Default: 1 (is TAB STOP)
```

If 1, the button becomes the "tab stop" for a "tab group" of PUSHBUTTONS (subsequent buttons that don't have TAB STOP set).

Note that, as with the LABEL, there is an interdependency between the COLUMNS, FONT, and WIDTH. (ROWs does not apply to a PUSHBUTTON.)

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                 TITLE (0)        USER DATA
VERSION            VISIBLE            WIDTH            X,Y
```

There is one unique event associated with the PUSHBUTTON:

ACTIVATED Occurs if mouse button is clicked on PUSHBUTTON.

[8.7] LABEL WIDGET ATTRIBUTE REFERENCE

* The LABEL widget has the following attributes:

```
COLUMNS                 Type:    numeric
```

```
                       Values:  positive integer
                       Default: 20
```

Gives size of LABEL in terms of number of characters it can contain.

```
FONT                   Type:    string
                       Values:  legal font name
                       Default: depends on system defaults
```

Gives name of font to be used in widget.

```
JUSTIFICATION          Type:    string
                       Values:  "LEFT" or "RIGHT" or "TOP" or
                                "BOTTOM" or "CENTER" or
                                "LEFT,TOP" or "LEFT,BOTTOM" or
                                "LEFT,CENTER" or "RIGHT,TOP" or
                                "RIGHT,TOP" or "RIGHT,BOTTOM" or
                                "RIGHT,CENTER"
                       Default: "CENTER"
```

Specifies justification of font in string.

```
PEN                    Type:    numeric
                       Values:  valid pen number (0-255)
                       Default: depends on system defaults
```

Specifies pen color of the font that displays the VALUE.

```
ROWS                   Type:    numeric
                       Values:  positive integer
                       Default: 2
```

Gives number of rows of text in widget.

```
VALUE                  Type:    string, numeric, complex
                       Values:  any string
                       Default: "" (null string)
```

Specifies information to be displayed.

```
WORD WRAP              Type:    numeric
                       Values:  0 or 1
                       Default: 0 (no wrap)
```

If 1, lines will wrap around when line length exceeds window size.

Note that ROWS, COLUMNS, WIDTH, HEIGHT, & FONT have complicated and devious relationships; specifying a COLUMNS value will adjust the WIDTH of the widget relative to the dimensions of the current FONT. Similarly, the ROWS attribute will affect the HEIGHT.

The default attributes and events also apply (the "0" means "level 0 only"):

BACKGROUND	BORDER	CLOSE event (0)	HEIGHT
HELP FILE	HELP TOPIC	INSIDE HEIGHT	INSIDE WIDTH
MAXIMIZABLE (0)	MINIMIZABLE (0)	MOVABLE (0)	RESIZABLE (0)
RESTORE SCREEN (0)	STACKING ORDER	SYSTEM MENU (0)	
SYSTEM MENU COUNT (0)		SYSTEM MENU EVENT (0)	
SYSTEM MENU event (0)		TITLE (0)	USER DATA
VERSION	VISIBLE	WIDTH	X,Y

The LABEL widget has no unique events.

[9.0] BPLUS (9): TOGGLEBUTTON & RADIOBUTTON / NUMBER & KEYPAD Widgets

v2.4 /01 feb 99 / greg goebel

* The TOGGLEBUTTON and RADIOBUTTON widgets are variations on the PUSHBUTTON widget; they are identical to the PUSHBUTTON, except for the fact that they have different appearances, and that both the TOGGLEBUTTON and RADIOBUTTON have a VALUE attribute that is toggled between 1 and 0 each time the widget is clicked with a mouse. The RADIOBUTTON further has the feature that it can be clustered within a tab group of other RADIOBUTTONs, so that when any one of the widgets is set, any other in the group will be automatically cleared.

The NUMBER and KEYPAD widgets are numeric-input widgets -- they allow entry in a wide variety of formats; they are basically the same as their equivalent dialogs.

This chapter describes these four widgets in detail and gives examples of their use.

[9.1] TOGGLEBUTTON WIDGET

* The TOGGLEBUTTON widget acts like the PUSHBUTTON widget in that you click on it and get an event. The only major differences are that the event is called CHANGED and not ACTIVATED, and that, as mentioned above, the TOGGLEBUTTON possesses a VALUE attribute that is toggled between 0 and 1 every time you click on the widget. (A small box changes from light to dark to reflect the value; default is 0, by the way.) The TOGGLEBUTTON is useful for emulating a two-position switch, or setting a mode of operation.

Program Example: xtoggle.rmb

[9.2] RADIOBUTTON WIDGET

* The RADIOBUTTON widget is a slight extension of the concept of TOGGLEBUTTON. Like a TOGGLEBUTTON, a RADIOBUTTON has a VALUE attribute

click on it once, it's set to 1, click on it again, it's cleared back to 0. A CHANGED event can be enabled to occur whenever you click on the RADIOBUTTON. Each RADIOBUTTON has a little diamond in it; set the RADIOBUTTON and the diamond goes black, clear the RADIOBUTTON and the diamond is cleared again.

The difference between a TOGGLEBUTTON and a RADIOBUTTON is that if you set or clear a TOGGLEBUTTON it has no direct effect on anything else in the user interface; but if you create a tab group of RADIOBUTTONs, then only one RADIOBUTTON in the group can be set at a time: set a different RADIOBUTTON in the tab group and any other RADIOBUTTON already set will be cleared. (Note that if one RADIOBUTTON in a tab group is set to 1 and you set another RADIOBUTTON to 1, you get TWO CHANGED events -- one for the original button as it is cleared, the other for the new button as it is set.)

This behavior is convenient for selecting a value from a set whose members are mutually exclusive: in simpler terms, a set where you can choose one or the other or the other but never more than one at a time. (The somewhat obscure term RADIOBUTTON is derived from the now archaic electromechanical car radios where you could push in a button to select a predefined channel, and any other button already pushed in would automatically pop out.)

Program Example: xradio.rmb

[9.3] NUMBER & KEYPAD WIDGETS

* The NUMBER widget is a somewhat unexciting widget; create one:

```
   ASSIGN @Num TO WIDGET "NUMBER"
```

-- and all you get is a box, into which you can enter a number; you can then read the VALUE entered into that box:

```
   STATUS @Num;RETURN ("VALUE":V)
```

The NUMBER widget enforces that a number will be entered in an acceptable format; you can't enter an incorrect character. It also provides the ability to translate numbers entered in binary, octal, or hex directly into BASIC INTEGER values -- using the FORMAT attribute:

```
   CONTROL @Num;SET ("FORMAT":"HEX","FORMAT LENGTH":4)
```

FORMAT can have values of REAL, SHORT INTEGER, BINARY, OCTAL, and HEX. Note that the FORMAT LENGTH attribute here gives a default field of 4 hex digits for the user to enter into.

You can also specify a round-off INCREMENT, and for REAL numbers, the number of fractional digits displayed -- using the REAL RESOLUTION attribute -- and the format for REAL numbers -- using the REAL NOTATION attribute (which can be set to FIXED, SCIENTIFIC, or ENGINEERING).

It is, however, limited in that you can't prevent the user from entering a number outside the range you want; while you can set MINIMUM and MAXIMUM limits on the numbers, these lead to some convoluted error handling -- and you're better off to simply check the input against variables giving the bounds:

```
   STATUS @Num;RETURN ("VALUE":V)
   IF (V<Minval) OR (V>Maxval) THEN
     DIALOG "ERROR","Input out of range!"
   END IF
```

The NUMBER widget supports a RETURN event -- which occurs when the user presses the Enter key to enter a number -- and a DONE event -- which occurs when the user moves the mouse cursor out of the widget. It also supports a KEYSTROKE event -- which occurs each time a character is entered -- and an INVALID NUMBER event -- which occurs when the user tries to input an incorrect value -- but these are less useful than the RETURN and DONE events.

* The KEYPAD widget has exactly the same features as the NUMBER widget, but provides a numeric keypad:

```
   ASSIGN @Pad TO WIDGET "KEYPAD";PARENT @Main
```

-- gives:

```
   +-----------------------+
   |                       |
   +-----+-----+-----+-----+
   |  A  |  B  |  C  |  D  |
   +-----+-----+-----+-----+
   |  7  |  8  |  9  |  E  |
   +-----+-----+-----+-----+
   |  4  |  5  |  6  |  F  |
   +-----+-----+-----+-----+
   |  1  |  2  |  3  | <-- |
   +-----+-----+-----+-----+
   |  -  |  0  |  .  | --> |
   +-----+-----+-----+-----+
   | CLR | DEL | INS | ENT |
   +-----+-----+-----+-----+
```

The only additional attributes supported by the KEYPAD are KEY BACKGROUND and KEY PEN (to set key colors) and SHOW EDIT (which allows you to turn off the edit box on the KEYPAD).

[9.4] TOGGLEBUTTON & RADIOBUTTON WIDGET ATTRIBUTE REFERENCE

* The TOGGLEBUTTON and RADIOBUTTON widgets have exactly the same attributes as the PUSHBUTTON, plus one extra, the VALUE attribute.

```
COLUMNS                Type:    numeric
                       Values:  positive integer
                       Default: 10
```

Gives size of button in terms of number of characters it can contain.

```
FONT                   Type:    string
                       Values:  legal font-file name
                       Default: depends on system defaults
```

Gives name of font to be used widget LABEL string.

```
LABEL                  Type:    string
                       Values:  any string
                       Default: "" (null string)
```

Specifies string to be displayed on button.

```
PANEL DEFAULT          Type:    numeric
                       Values:  0 or 1
                       Default: 0 (not default)
```

If 1, the button is the default pushbutton for a PANEL -- pressing RETURN anywhere in the panel will activate the button.

```
PEN                    Type:    numeric
                       Values:  valid pen number
                       Default: depends on system defaults
```

Specifies pen color of button's font.

```
SENSITIVE              Type:    numeric
                       Values:  0 or 1
                       Default: 1 (is SENSITIVE)
```

If 0, the button is still visible but won't respond to user input.

```
TAB STOP              Type:    numeric
                      Values:  0 or 1
                      Default: 1 (is TAB STOP)
```

If 1, the button becomes the "tab stop" for a "tab group" (subsequent buttons that don't have TAB STOP set).

```
VALUE                 Type:    numeric
                      Values:  0 or 1
                      Default: 0
```

If 1, the TOGGLEBUTTON or RADIOBUTTON is set; if 0, it is cleared.

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER           CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC       INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)  MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER   SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                TITLE (0)        USER DATA
VERSION            VISIBLE           WIDTH            X,Y
```

There is one unique event associated with the TOGGLEBUTTON or RADIOBUTTON; it's basically the same as the ACTIVATED event for the PUSHBUTTON:

```
CHANGED   Indicates mouse button is clicked.
```

[9.5] NUMBER & KEYPAD WIDGET ATTRIBUTE REFERENCE (2.0 & LATER ONLY)

* The NUMBER and KEYPAD widgets have the following attributes:

```
CHECK FOR DONE        Type:    numeric
```

```
                       Values:  0 or 1
                       Default: 1 (ON)

When non-zero, enables loss-of-focus formatting and DONE event.
```

```
COLUMNS                Type:    numeric
                       Values:  non-negative
                       Default: 20

Width of widget in character cells.
```

```
FONT                   Type:    string
                       Values:  legal font specification
                       Default: depends on system defaults

Font used for text.
```

```
FORMAT                 Type:    string
                       Values:  "REAL" or "SHORT INTEGER" or
                                "BINARY" or "OCTAL" or "HEX"
                       Default: "REAL"

Specifies numeric format.
```

```
FORMAT LENGTH          Type:    numeric
                       Values:  0 to 328
                       Default: 0

Defines the number of characters used to display the number, with
zeroes padded on the left; intended for use with BINARY, OCTAL, or
HEX values.
```

```
INCREMENT              Type:    numeric
                       Values:  non-negative; 0 = "don't round"
                       Default: 0

VALUE is "rounded" to the nearest specified INCREMENT.
```

```
KEY BACKGROUND         Type:    numeric
                       Values:  legal PEN numbers (0-255)
KEYPAD only            Default: depends on system defaults

Specifies color of key background.
```

```
KEY PEN                Type:    numeric
```

```
                       Values:  legal PEN numbers (0-255)
KEYPAD only            Default: depends on system defaults
```

Specifies color of key font.

```
MAXIMUM                Type:    numeric
                       Values:  in the range between current
                                FORMAT's maximum and minimum.
                       Default: MAXREAL
```

Maximum value widget will accept.

```
MAXIMUM LENGTH         Type:    numeric
                       Values:  0 to 328
                       Default: 328
```

Maximum number of characters operator can enter; however, when the widget reformats the operator entry, the text length may be longer.

```
MINIMUM                Type:    numeric
                       Values:  in the range between current
                                FORMAT's maximum and minimum.
                       Default: MAXREAL
```

Maximum value widget will accept.

```
MODIFIED               Type:    numeric
                       Values:  0 or 1
                       Default: 0 (NO)
```

When non-zero, indicates operator has changed the contents.

```
PEN                    Type:    numeric
                       Values:  legal PEN numbers (0-255)
                       Default: depends on system defaults
```

Pen used for text.

```
REAL NOTATION          Type:    string
                       Values:  "FIXED" or "SCIENTIFIC" or
                                "ENGINEERING"
                       Default: "SCIENTIFIC"
```

Specifies real-number display format (only applicable to "FORMAT":"REAL")

```
REAL RESOLUTION        Type:    numeric
                       Values:  0 to 16
                       Default: 12
```

Number of fractional digits displayed (only applicable to "FORMAT":"REAL").

```
SHOW EDIT              Type:    numeric
                       Values:  0 or 1
KEYPAD only            Default: 1 (ON)
```

Controls display of editing field.

```
TAB STOP               Type:    numeric
                       Values:  0 or 1
                       Default: 1
```

Controls membership in tab groups.

```
TEXT LENGTH            Type:    numeric (RETURN only)
                       Values:  integer
```

Gives length of text string the user's typed in.

```
VALUE                  Type:    numeric or string
                       Values:  as allowed by "FORMAT", "MINIMUM",
                                and "MAXIMUM"
                       Default: 0
```

Value returned by widget. "BINARY", "OCTAL", "HEX" are all in two's complement (no "-" sign allowed).

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER           CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC       INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)  MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER   SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                TITLE (0)        USER DATA
VERSION            VISIBLE           WIDTH            X,Y
```

There are four unique events associated with these widgets:

DONE	Occurs when the widget loses mouse focus.
KEYSTROKE	Occurs when a key is press that that potentially could change the display.
RETURN	Occurs when the user hits the return key.
INVALID NUMBER	Occurs if an entry causes the display to be invalid in the current "FORMAT"; or on "DONE" or "RETURN", if the number is outside "MINIMUM" or "MAXIMUM".

[10.0] BPLUS (10): STRING & COMBO Widgets

v2.4 / 01 feb 99 / greg goebel

* Given the fundamental PANEL, LABEL, and PUSHBUTTON (etc.) widgets, we can build reasonably intelligent user interfaces; but other features would of course be nice -- for example, the ability to get text inputs from the user. That's what the STRING widget is for.

The COMBO widget extends the STRING widget by allowing the user to enter text and selections from a predefined list of strings.

This chapter describes these widgets and provides reference information for their use.

[10.1] STRING WIDGET

* The STRING widget allows a BPlus user interface to accept strings of text from the user. We can create a STRING widget very easily:

```
   10    CLEAR SCREEN
   15    DIM S$[255]
   20    ASSIGN @S TO WIDGET "STRING"
   30    CONTROL @S;SET ("X":100,"Y":100,"COLUMNS":15)
   35    CONTROL @S;SET ("VALUE":"Type here!")
   40    ON EVENT @S,"RETURN" GOSUB Handler
   50    !
   60    LOOP
   70    END LOOP
   80    !
   90 Handler: !
   100   STATUS @S;RETURN ("VALUE":S$)
   105   CONTROL @S;SET ("VALUE":"")
   110   IF S$="QUIT" THEN Finis
   120   DISP S$
   130   RETURN
   140   !
   150 Finis: END
```

You get a widget of the form:

```
   +----------------+---+
   |      STRING    | X |
   +----------------+---+
   | Type here!         |
   +--------------------+
```

You can input text into the lower half; move the mouse cursor there and just start typing. In this program, when you press RETURN, the string will be displayed at the bottom of the display.

In this case, I used a RETURN EVENT, so that my "Handler" routine wouldn't do anything until RETURN is pressed. I could have also set a KEYSTROKE EVENT to handle character-by-character input, or a DONE EVENT, in which case I would get the EVENT when the mouse cursor was moved elsewhere.

* BPlus version 2.0 introduce a MULTILINE attribute that allowed the STRING widget to accept multiline inputs; a large number of other attributes were added to allow the STRING widget in MULTILINE mode to function as a full-fledged (if modest) text editor.

The best way to illustrate these features is to actually use them to build a text editor. We can build a simple text editor that appears as follows:

```
   +------------------------------------+---+
   |          STRING Widget Editor      | X |
   +------------------------------------+---+
   | Files  Edit                            |
   +----------------------------------------+
   |                                        |
   |                                        |
   |                                        |
   |                                        |
   |                                        |
   |                                        |
   |                                        |
   |                                        |
   |                                        |
   |                                        |
   |                                        |
   +----------------------------------------+
```

-- where the menus have the entries:

```
   | Files  Edit
   +----------+---
   | New      |         Clear editor.
   | Open...  |         Open file.
   | Save...  |         Save file.
   | Merge... |         Merge text from file into editor.
   +----------+
   | Quit     |         Quit editor.
   +----------+
     Edit
  -+----------------+--
   | Cut            |    Delete text and store in "paste buffer".
   | Copy           |    Copy text into paste buffer.
   | Paste          |    Paste text from buffer into editor.
   | Replace        |    Replace editor text with buffer text.
   +----------------+
   | Line Number... |    Find line number.
   | String...      |    Find string from cursor.
   +----------------+
```

Program Example: xstred.rmb

[10.2] COMBO WIDGET

* The COMBO widget is very much the same as its dialog equivalent, and works much the same: it allows you to present a list of ITEMS to the user, and return text through the TEXT attribute from that list, or a string input by the user.

The COMBO widget of course, unlike the dialog version, uses events to detect when it is activated. The COMBO widget has three events: SELECTION, which occurs when the user selects something from the list; RETURN, which occurs when the user types something into the string field and presses Enter; and KEYSTROKE, which occurs every time the user presses a key.

The following (dumb) example demonstrates:

```
   10    CLEAR SCREEN
   20    DIM L$(1:5)[25],S$[50]
   30    INTEGER N
   40    !
   50    DATA "Aardvark","Sidewinder","Kiwi","Pangolin","Marmoset"
   60    READ L$(*)
   70    !
   80    ASSIGN @Combo TO WIDGET "COMBO";SET ("SYSTEM MENU":"Quit")
   90    CONTROL @Combo;SET ("TITLE":"Select Your Favorite Animal!")
   100   CONTROL @Combo;SET ("ITEMS":L$(*))
   110   !
   120   ON EVENT @Combo,"SELECTION" GOSUB Handler
   130   ON EVENT @Combo,"RETURN" GOSUB Handler
   140   ON EVENT @Combo,"SYSTEM MENU" GOTO Finis
   150   !
   160   LOOP
   170   END LOOP
   180   STOP
   190   !
   200 Handler: !
   210   STATUS @Combo;RETURN ("TEXT":S$)
   220   FOR N=1 TO 5
   230     IF S$=L$(N) THEN
   240       S$="List item #"&VAL$(N)&": "&S$
   250       DIALOG "INFORMATION",S$
   260       RETURN
   270     END IF
   280   NEXT N
   290   S$="Animal not in list: "&S$
   300   DIALOG "INFORMATION",S$
   310   RETURN
   320   !
   330 Finis: !
   340   ASSIGN @Combo TO *
   350   END
```

This pops up a widget of the form (shown with the list pulled down):

```
   +---+-------------------------------+---+
   | = | Select Your Favorite Animal! | X |
   +---+-------------------------------++--+
   |                                  |  |
   +----------------------------------+--+
   | Aardvark                         |  |
   | Sidewinder                       |  |
   | Kiwi                             |  |
   | Pangolin                         |  |
   | Marmoset                         |  |
   +----------------------------------+--+
```

Note how both the SELECTION and RETURN events are used to call a handler routine that gets the TEXT returned by the widget, and how the handler simply searches through the ITEMS array to see if there is a match or not. You could also use a SELECT statement to match to the list elements, with a CASE ELSE clause used to handle items typed directly into the COMBO widget.

This is simpler than having a separate routine for getting a list index and for getting text back ... you could have a handler that is called by the SELECTION event and uses the SELECTION attribute to get the list index of the item, but just using the TEXT attribute is simpler for most practical purposes.

[10.3] STRING WIDGET ATTRIBUTE REFERENCE

* The STRING widget supports the following attributes:

AUTOSCROLL Type: numeric

```
                       Values:  0 or 1
                       Default: 0 (OFF)
```

If 1, characters will be scrolled if there are more than can be displayed.

```
COLUMNS                Type:    numeric
                       Values:  positive integer
                       Default: 40
```

Gives size of STRING widget in terms of the number of characters that can fit into it.

```
CONVERT CASE           Type:    string
                       Values:  "NONE" or "UPSHIFT" or "DOWNSHIFT"
2.0 & later only       Default: "NONE"
```

Converts all entered text to upper or lower case.

```
COPY SELECTION         Type:    numeric (write only)
2.0 & later only       Values:  0 (no-op) or 1.
```

Causes current selection copied to the clipboard.

```
CURSOR POSITION        Type:    numeric
2.0 & later only       Values:  0 to LINE LENGTH - 1
```

Position of cursor in current line; the first character is 0.

```
CUT SELECTION          Type:    numeric (write only)
2.0 & later only       Values:  0 (no-op) or 1.
```

Causes current selection copied to the clipboard, then deleted.

```
DELETE SELECTION       Type:    numeric (write only)
2.0 & later only       Values:  0 (no-op) or 1.
```

Delete current selection.

```
FONT                   Type:    string
                       Values:  legal font-file name
                       Default: depends on system defaults
```

Gives name of font to be used in STRING widget.

HORIZONTAL SCROLL	Type:	numeric (write only)
2.0 & later only	Values:	positive integer

Specifies scrolling contents of window the specified number of characters horizontally.

LINE	Type:	string (read only)
2.0 & later only	Values:	valid strings

Text of current line, including any CR-LFs.

LINE LENGTH	Type:	numeric (read only)
2.0 & later only	Values:	non-negative integer.

Length of current line, including CR-LF.

LINE NUMBER	Type:	numeric
2.0 & later only	Values:	0 to LINES - 1

Number of line containing cursor, first line number is 0.

LINES	Type:	numeric (read only)
2.0 & later only	Values:	non-negative integer.

Returns number of lines in widget.

MAXIMUM LENGTH	Type:	numeric
	Values:	positive integer
	Default:	40

Maximum length of text allowed in widget.

MODIFIED	Type:	numeric
	Values:	0 or 1
2.0 & later only	Default:	0 (NO)

When 1, indicates operator has changed the contents.

MULTILINE	Type:	numeric
	Values:	0 or 1
2.0 & later only	Default:	0 (OFF)

Specifies a multiline display; data can be accessed through a string array or a single string with embedded CR-LFs to separate lines.

PASSWORD CHARACTER	Type:	string
	Values:	"" (null string) or any character
2.0 & later only	Default:	"" (null string)

When set to non-null, all entered characters are displayed as the specified password character.

PASTE	Type:	numeric (write only)
2.0 & later only	Values:	0 (no-op) or 1.

Causes text contents of clipboard to be inserted before current cursor position.

PEN	Type:	numeric
	Values:	valid pen number
	Default:	depends on system defaults

Specifies PEN color of STRING widget's font.

READ FILE	Type:	string (write only)
2.0 & later only	Values:	valid filename

Reads file and inserts its contents before current cursor position.

SCROLLBARS	Type:	numeric
	Values:	0 or 1
2.0 & later only	Default:	0 (OFF)

When 1, horizontal or vertical scrollbars appear when the widget contains more text than can be displayed.

SEARCH	Type:	string (write only)
2.0 & later only	Values:	any

Specifies string to be searched for from current position; the search is case-insensitive. If a match is found, the view window is positioned to the match, the match is highlighted, and becomes the current selection; if no match is found, error 59 -- "End of file, buffer, or pipe found" -- is generated.

SELECTION	Type:	string
2.0 & later only	Values:	valid strings

Text of current selection, including any CR-LFs.

```
SELECTION LENGTH       Type:    numeric
2.0 & later only       Values:  non-negative integer.

Length of selected text.
```

```
TAB STOP               Type:    numeric
                       Values:  0 or 1
                       Default: 1 (is TAB STOP)

Used to set up tab groups of widgets.
```

```
TEXT LENGTH            Type:    numeric (read only)
2.0 & later only       Values:  non-negative integer.

Number of characters in the text, including all CR-LFs except the last.
```

```
VALUE                  Type:    string
                       Values:  any legal string
                       Default: "" (null string)

Returns string typed in by user; can also be given an initial value.
```

```
VERTICAL SCROLL        Type:    numeric (write only)
2.0 & later only       Values:  positive integer

Specifies scrolling contents of window the specified number of
characters vertically.
```

```
WRITE FILE             Type:    string (write only)
2.0 & later only       Values:  valid filename

Specifies file to write contents to.
```

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)              TITLE (0)        USER DATA
VERSION           VISIBLE          WIDTH            X,Y
```

[10.4] COMBO WIDGET ATTRIBUTE REFERENCE (2.0 & LATER ONLY)

* The COMBO widget has the following attributes:

```
BACKGROUND              Type:    numeric
                        Values:  legal PEN numbers (0-255)
                        Default: depends on system defaults

Sets color of input box background (ignored on one-plane mono
monitors).
```

```
COLUMNS                 Type:    numeric
                        Values:  0 to 32,767
                        Default: depends on system defaults

Width of widget in character cells.
```

```
DROPDOWN BUTTON         Type:     numeric
                        Values:   0 or 1
                        Default:  1 (ON)

Specifies the presence of a button that controls the visiblity of the
listbox; without the button, the listbox is always displayed
```

```
EDITABLE                Type:     numeric
                        Values:   0 or 1
                        Default:  1 (ON)

Sets up combo box text field as an edit field or a display.
```

```
FONT                    Type:    string
                        Values:  legal font specification
                        Default: depends on system defaults

Font used for text.
```

```
HEIGHT                  Type:    numeric
                        Values:  0 to 32,767
```

	Default: depends on system defaults

On a SET, this attribute includes height of list box, whether visible or not; on a RETURN, it includes list box only if "DROPDOWN BUTTON" is 0.

ITEM COUNT	Type: numeric (read only)
	Values: 0 to 32,767
	Default: 0

Returns number of items in list box.

ITEMS	Type: string array
	Values: any

Values to be displayed in the list box.

LIST BACKGROUND	Type: numeric
	Values: legal PEN numbers (0-255)
	Default: depends on system defaults

Sets color of list box background (ignored on one-plane mono monitors).

LIST PEN	Type: numeric
	Values: legal PEN numbers (0-255)
	Default: depends on system defaults

Specifies pen color of list box text (ignored on one-plane mono monitors).

PEN	Type: numeric
	Values: legal PEN numbers (0-255)
	Default: depends on system defaults

Sets pen color of input box text (ignored on one-plane mono monitors).

ROWS	Type: numeric
	Values: 0 to 32,767
	Default: depends on system defaults

Height of widget in character cells.

SCROLLBAR	Type: numeric
	Values: 0 or 1
	Default: 1 (ON)

Specifies whether the list box has a scroll bar.

SELECTION	Type: numeric Values: list index (0 to ITEM COUNT) -1 if nothing selected Default: -1

Index of selected item; the index always starts at 0.

SENSITIVE	Type: numeric Values: 0 or 1 Default: 1 (YES)

If 0, the widget is still visible but won't respond to user input.

SHOW LIST	Type: numeric Values: 0 or 1 Default: 0 (NO)

Specifies whether list box is shown or not.

TAB STOP	Type: numeric Values: 0 or 1 Default: 1 (YES)

Used to set up tab groups of widgets.

TEXT	Type: string Values: any Default: "" (null string)

Returns string in edit field.

The default attributes and events also apply (the "0" means "level 0 only"):

BACKGROUND	BORDER	CLOSE event (0)	HEIGHT
HELP FILE	HELP TOPIC	INSIDE HEIGHT	INSIDE WIDTH
MAXIMIZABLE (0)	MINIMIZABLE (0)	MOVABLE (0)	RESIZABLE (0)
RESTORE SCREEN (0)	STACKING ORDER	SYSTEM MENU (0)	
SYSTEM MENU COUNT (0)		SYSTEM MENU EVENT (0)	
SYSTEM MENU event (0)		TITLE (0)	USER DATA

VERSION	VISIBLE	WIDTH	X,Y

There are three unique events associated with this widget:

SELECTION	Occurs when user selects an item from the list box.
KEYSTROKE	Occurs when user types a character in the edit field.
RETURN	Occurs when return key is hit.

---

[11.0] BPLUS (11): LIST / FILE / PRINTER / HPGL VIEW / BITMAP Widgets

v2.4 / 01 feb 99 / greg goebel

* The LIST widget allows selection from a list of items; the FILE widget allows selection of a disk volume, directory, or file. The PRINTER, HPGL VIEW, and BITMAP widgets allow you to view text, HPGL graphics, or .BMP or .XWD bitmap files respectively. This chapter describes these widgets in detail and provides reference information for their use.

---

---

[11.1] LIST WIDGET

* The LIST widget is much like a COMBO widget that doesn't allow you to enter a string, just select from a list of items. It has a few more capabilities in this regard than the COMBO widget.

While the COMBO widget allows you to get the text provided by the user, no matter if it comes from a list or string, the LIST widget only gets an index into its list from the user; there's no particular need for it to provide anything else ... it behaves in this fashion much like its sister LIST dialog. For example:

```
   10    CLEAR SCREEN
   20    DIM S$(0:10)[50]
   30    INTEGER N
   40    FOR N=0 TO 9
   50      S$(N)="ITEM "&VAL$(N)
   60    NEXT N
   70    S$(10)="QUIT"
   80    !
   90    ASSIGN @L TO WIDGET "LIST"
   100   CONTROL @L;SET ("X":100,"Y":100,"COLUMNS":10,"ROWS":5)
   110   CONTROL @L;SET ("ITEMS":S$(*))
   120   CONTROL @L;SET ("TOP ITEM":2)
   130   !
   140   ON EVENT @L,"SELECTION" GOSUB Handler
   150   !
   160   LOOP
   170   END LOOP
   175   STOP
   180   !
   190 Handler: !
   200   STATUS @L;RETURN ("SELECTION":N)
   210   IF N=10 THEN Finis
   220   DISP N
   230   RETURN
   240   !
   250 Finis: !
   260   END
```

-- gives the widget:

```
   +--------+---+
   |  LIST  | X |
   +--------+-+-+
   | ITEM 2   |^|
   | ITEM 3   | |
   | ITEM 4   | |
   | ITEM 5   | |
   | ITEM 6   |v|
   +----------+-+
```

Note that we specified 5 rows, so we only get 5 of the items we defined; the LIST widget has provided a SCROLLBAR so we can get at the others. Note also that we specified TOP ITEM as 2, so the item with index 2 is at the top of the list; the ITEM 0 and ITEM 1 strings are hidden above it.

When we click on an item in this list, we get the corresponding index of the item displayed at the bottom of the screen. Note that in this case, we have the defined the string array to start at a bottom index of 0, so this index matches the string array index of the item. If you use a different bottom index, please be aware that the index returned will still be for an array with a bottom address of 0 -- that is, if you declared the array with a bottom address of 1 the index the LIST widget would return would be one less than the actual index of the item.

* In this demo, you can only select one LIST item at a time; but you might want to select any combination of the LIST items instead, and for that reason you can set a MULTISELECT attribute on the LIST widget.

When MULTISELECT is in effect, the SELECTION attribute uses a numeric array as a value; that array has the same number of entries as the ITEMS string array, and will contain a 0 in entries that correspond to ITEMS that are not selected and a 1 in entries that correspond to ITEMS that are selected.

Let's modify our example program a little to operate in MULTISELECT mode:

```
   10    CLEAR SCREEN
   20    DIM S$(0:10)[50]
   30    INTEGER N,Nval(0:10)
   40    FOR N=0 TO 9
   50       S$(N)="ITEM "&VAL$(N)
   60    NEXT N
   70    S$(10)="QUIT"
   80    !
   90    ASSIGN @L TO WIDGET "LIST"
   100   CONTROL @L;SET ("X":100,"Y":100,"COLUMNS":10,"ROWS":5)
   110   CONTROL @L;SET ("MULTISELECT":1)
   120   CONTROL @L;SET ("ITEMS":S$(*))
   130   CONTROL @L;SET ("TOP ITEM":2)
   140   !
   150   ON EVENT @L,"SELECTION" GOSUB Handler
   160   !
   170   LOOP
   180   END LOOP
   185   STOP
   190   !
   200 Handler: !
   210   STATUS @L;RETURN ("SELECTION":Nval(*))
   220   IF Nval(10)=1 THEN Finis
   230   DISP USING "10(D,X)";Nval(*)
   240   RETURN
   250   !
   260 Finis: !
   270   END
```

This yields a widget that looks exactly like the one in the previous example program, but when you click on a LIST item, you get a list of 11 1s and 0s. For example, if you click on ITEM 0 and ITEM 9 you get:

```
   1 0 0 0 0 0 0 0 0 1 0
```

Program Example: xlist.rmb

[11.2] FILE WIDGET

* There isn't much more to the FILE widget than there is to the FILE dialog; like the dialog, it has a PATTERN attribute to define a wildcard file specification, a DIRECTORY attribute to set or return the proper directory, and a SELECTION attribute to set or return the selected file.

Of course, as a widget it also needs an event, so a SELECTION event is defined for the FILE widget that occurs when a file has been selected.

The following short example program demonstrates the FILE widget:

```
   10    CLEAR SCREEN
   20    DIM S$[256],D$[50],F$[50],W$[50],Eol$[10]
   30    Eol$=CHR$(10)&CHR$(10)
   40    INTEGER Btn
   50    !
   60    ASSIGN @File TO WIDGET "FILE";SET ("SYSTEM MENU":"Quit")
   70    CONTROL @File;SET ("TITLE":"FILE Widget Demo")
   80    !
   90    ON EVENT @File,"SELECTION" GOSUB Getfile
   100   ON EVENT @File,"SYSTEM MENU" GOTO Finis
   110   !
   120   LOOP
   130   END LOOP
   140   STOP
   150   !
   160 Getfile: !
   170   STATUS @File;RETURN ("SELECTION":F$,"DIRECTORY":D$,"PATTERN":W$)
   180   S$="File = "&F$&Eol$&"Directory = "&D$&Eol$&"Wildcard = "&W$
   190   DIALOG "INFORMATION",S$
   200   RETURN
   210   !
   220 Finis: !
   230   ASSIGN @Main TO *
   240   END
```

When you run this program, you get the following widget:

```
   +---+-------------------------------+---+
   | = |       File Widget Demo        | X |
   +---+-------------------------------+---+
   |                                       |
   | Current directory:       Directories: |
   | +---------------------+  +--------+-+ |
   | |                     |  |        | | |
   | +---------------------+  +--------+-+ |
   |                                       |
   | File wildcard:           Files:       |
   | +---------------------+  +--------+-+ |
   | |                     |  |        | | |
   | +---------------------+  +--------+-+ |
   |                                       |
   | File selection:                       |
   | +-----------------------------------+ |
   | |                                   | |
   | +-----------------------------------+ |
   +---------------------------------------+
```

You can change the directory and file wildcards at will; when you actually select a file, a dialog pops up and tells you what file, wildcard pattern, and directory you have selected. Like I said ... not much to it.

[11.3] PRINTER Widget

* The PRINTER widget is so simple that it's hard to explain; basically, it just allows you to dump lines of text to a display and watch them scroll upward. The following example program shows how it works:

```
10    CLEAR SCREEN
15    INTEGER N
30    ASSIGN @P TO WIDGET "PRINTER"
40    CONTROL @P;SET ("X":0,"Y":0,"COLUMNS":50,"ROWS":10)
60    ON KBD ALL GOTO Finis
70    !
90    FOR N=1 TO 100
100      CONTROL @P;SET ("APPEND TEXT":"THIS IS A TEST -- LINE "&VAL$(N))
110   NEXT N
120   CONTROL @P;SET ("TOP LINE":20)
130   WAIT 3
140   CONTROL @P;SET ("VALID LINES":0)
150   WAIT 3
170   !
180   END
```

This creates a box on the display and scrolls text up through it:

```
+----------------------------------------+---+
|                  PRINTER               | X |
+----------------------------------------+-+-+
| THIS IS A TEST -- LINE 12                |^|
| THIS IS A TEST -- LINE 13                | |
| THIS IS A TEST -- LINE 14                | |
| THIS IS A TEST -- LINE 15                | |
| THIS IS A TEST -- LINE 16                | |
| THIS IS A TEST -- LINE 17                | |
| THIS IS A TEST -- LINE 18                | |
| THIS IS A TEST -- LINE 19                | |
| THIS IS A TEST -- LINE 20                | |
| THIS IS A TEST -- LINE 21                | |
| THIS IS A TEST -- LINE 22                |v|
+------------------------------------------+-+
```

If you run this program, note that the scrollbar on the right of the box appears only after line 10 appears and the text starts scrolling upward.

The PRINTER widget normally has a text buffer of 100 lines, as given by the sum of the ROWS attribute (which gives the space for lines to be displayed) and the HIDDEN attribute (which gives the space for the remainder that you can't see); you can adjust these parameters if you like. The VALID LINES attribute gives the number of lines that are actually used; and you can reposition the text in the display to any line in the buffer with the TOP LINE attribute.

The APPEND TEXT attribute prints a line of text in the box; you get a new line of text every time you use it. If you want to skip a line, just use it with a null string. You can also invoke it with a string array as a parameter; we can change our example program as follows:

```
   20    DIM S$(1:100)[50]
   ...
   110     FOR N=1 TO 100
   120        S$(N)="THIS IS A TEST -- LINE "&VAL$(N)
   130     NEXT N
   140     CONTROL @P;SET ("APPEND TEXT":S$(*))
   150     WAIT 3
   160     CONTROL @P;SET ("TOP LINE":20)
   170     WAIT 3
```

This dumps the entire string array at one time.

There is also a TEXT attribute that allows you to print a line of text and clear the PRINTER at the same time; you could clear the PRINTER by using TEXT with a null string. You can also set VALID LINES to 0 to do the same thing.

Note that you cannot perform a PRINTER IS to a PRINTER widget.

* A number of enhanced features were added to the PRINTER widget in BPlus 2.0. For example, you can set the DISPLAY TEXT attribute to 0 and PRINTER output won't be shown until to set DISPLAY TEXT back to 1; this is handy for increasing display speed if you are doing line-by-line output.

Other new features allow you to select a line in the PRINTER output using the CURRENT LINE attribute -- and then use the DELETE LINE attribute to delete that line and any number following; or use the INSERT TEXT attribute to insert a string or string array before the CURRENT LINE; or use the CURRENT TEXT attribute to read the current text into a string or string array.

The following (somewhat brain-dead) demo shows how these elements are used, by dumping five time-stamped strings to a PRINTER widget from the top down, and then continually recirculating the entries from the bottom to the top -- by reading in line 5, deleting it, then inserting it again at the top:

```
   10    CLEAR SCREEN
   20    !
   30    DIM S$[50],P$[50],T$[50]
   40   INTEGER Lines
   50   !
   60   ASSIGN @Prn TO WIDGET "PRINTER"
   70   CONTROL @Prn;SET ("SYSTEM MENU":"Quit")
   80   !
   90   ON EVENT @Prn,"SYSTEM MENU" GOTO Finis
   100  !
   110  S$=TIME$(TIMEDATE)&" PRINT TEST LINE 1"
   120  CONTROL @Prn;SET ("APPEND TEXT":S$)
   130  Lines=1
   140  LOOP
   150    REPEAT
   160      P$=S$
   170      S$=TIME$(TIMEDATE)
   180    UNTIL S$<>P$
   190    IF Lines<5 THEN
   191      Lines=Lines+1
   200      T$=S$&" PRINT TEST LINE "&VAL$(Lines)
   210      CONTROL @Prn;SET ("CURRENT LINE":1,"INSERT TEXT":T$)
   230    ELSE
   240      CONTROL @Prn;SET ("CURRENT LINE":5)
   250      STATUS @Prn;RETURN ("CURRENT TEXT":T$)
   260      CONTROL @Prn;SET ("CURRENT LINE":5,"DELETE LINES":1)
   270      CONTROL @Prn;SET ("CURRENT LINE":1,"INSERT TEXT":T$)
   280    END IF
   290  END LOOP
   300  !
   310 Finis: !
   320  ASSIGN @Prn TO *
   330  END
```

This program ends up with print output that looks something like this:

```
   09:45:14 PRINT TEST LINE 3
   09:45:13 PRINT TEST LINE 2
   09:45:12 PRINT TEST LINE 1
   09:45:16 PRINT TEST LINE 5
   09:45:15 PRINT TEST LINE 4
```

There is also a HIGHLIGHT CURRENT attribute that highlights the CURRENT LINE selection -- and allows it to then be changed by the user with a mouse; you can then read the value back through the CURRENT LINE attribute.

Program Example: xfilevue.rmb.

[11.4] HPGL VIEW WIDGET

* The HPGL VIEW widget is a close relative of the PRINTER widget, but it

allows you to display plotter graphics commands instead of text. Operation of the HPGL VIEW widget is simple. All you have to do is create the widget and then execute:

```
   CONTROL @Hpgl;SET ("HPGL FILE":"Graffile")
```

-- where "Graffile" is an arbitrary HPGL graphics file name. If the file's not there or it isn't an HPGL file, you get an error message.

The HPGL VIEW widget does not support a large set of HPGL commands; for example, area fills don't work and graphics files that include them tend to look a little strange. You cannot do a PLOTTER IS to an HPGL VIEW widget.

Note also that if you move or in other ways disturb the HPGL VIEW widget, all the graphics will be redrawn, which can be painful. However, you can set a RETAIN RASTER attribute to 1 -- and then BPlus will take a snapshot of the pixels and retain them, so you don't have to redraw unless you resize the widget.

Program Example: xhpglvue.rmb

[11.5] BITMAP WIDGET

* The BITMAP widget was added in BPlus 2.0 and is similar to the HPGL VIEW widget, except that it displays a raster-graphics file -- either an MS-Windows style .BMP file ("bumpfile") or an X11 style .XWD file.

The file is read into the widget using the BITMAP FILE attribute:

```
   CONTROL @Bitmap;SET ("BITMAP FILE":"CINDY.BMP")
```

Like the HPGL VIEW widget, the BITMAP widget supports RETAIN RASTER, but also supports a number of unique attributes. One of the most useful is AUTO SIZE, which, when set, re-scales the widget to fit the bitmap.

* The BITMAP widget also has a peculiar type of input function associated with it: it allows you to take bitmapped images off a BPlus user interface and store the bits in a file. If you set the DUMP WINDOW attribute to a filename:

```
   CONTROL @Bitmap;SET ("DUMP FORMAT":"BMP,"DUMP WINDOW":"BITSAVE.BMP")
```

-- and then click on the BPlus user interface, the user interface will be saved in the file (with the format given by DUMP FORMAT as shown above ... the format can have the value of either BMP or XWD). There is also a similar DUMP AREA attribute that allows you to store a region of the user interface -- as defined by a mouse click-and-drag operation -- to a file.

* Other minor attributes include FRAME (which puts a frame around the bitmap panel), MAX PENS (which sets a limit on the number of pens a bitmap can use), and PENS NEEDED (which gives the number of pens required by a bitmap file). You can also set a MOUSE CLICKED event to trap a mouse click on a bitmap, and then use the MOUSE CLICK attribute to return the X,Y coordinates of the pixel the mouse clicked on.

* You can get very strange results when dealing with bitmaps on occasion that may lead you to think that the BITMAP widget has a bug -- but it's really only doing what it has to do. For example, color bitmaps usually have either 16 or 256 colors; you can easily display a color bitmap of 16 colors on a display that supports 256, but if you try to display a color bitmap of 256 colors on one that only supports 16 -- you get some very strange colorations.

Another oddity is due to the fact that two bitmap files with 256 colors may not have the same 256 colors -- the display can be reprogrammed to select the 256 colors from a much larger set of colors, and this "color mapping" may change from bitmap file to bitmap file.

So what? Well, if you load two bitmap files consecutively that have different color mappings, the second one loaded will change the display's color map to what it wants, regardless of what the first one set it to; and the colors of the first bitmap will change to a gaudy set of psychedelic colors.

Like I said, these are not bugs -- the BITMAP widget is doing exactly what it's told, it's just that what it's being told leads it to a contradiction.

Program Example: xbmapvue.rmb

[11.6] LIST WIDGET ATTRIBUTE REFERENCE

* The LIST widget has the following attributes:

COLUMNS Type: numeric
Values: positive integer
Default: 10

Gives width of widget in terms of the number of characters it can contain.

FONT Type: string
Values: legal font-file name
Default: depends on system defaults

Gives name of font to be used in LIST widget.

```
ITEMS                  Type:    string array
                       Values:  valid strings
                       Default: zero-element array
```

A string array whose entries define the list displayed by the widget.

```
ITEM COUNT             Type:    numeric (read only)
                       Values:  0 to 32,767
2.0 & later only       Default: 0
```

Returns the number of items in the list box.

```
MULTISELECT            Type:    numeric
                       Values:  0 or 1
                       Default: 0 (single-select mode)
```

Allows only one (0) or multiple (1) list items to be selected.

```
ROWS                   Type:    numeric
                       Values:  positive integer
                       Default: 5
```

Gives size of widget in terms of number or rows of text.

```
SELECTION              Type:    numeric
                       Values:  legal index into ALTERNATIVES array
when MULTISELECT = 0   Default: -1
```

Specifies index of selected ITEM. (Array index always counts from 0.)

```
SELECTION              Type:    numeric array
                       Values:  legal index into ITEMS array
when MULTISELECT = 1   Default: all zeroes
```

This array should match the size of the ITEMS string array; ITEMS entries that are selected will contain a 1, those that aren't will contain a 0.

```
SENSITIVE              Type:    numeric
                       Values:  0 or 1
                       Default: 1 (is SENSITIVE)
```

If 0, the LIST is still visible but won't respond to user input.

TAB STOP	Type: numeric Values: 0 or 1 Default: 1 (is TAB STOP)

Used to set up tab groups of widgets.

TOP ITEM	Type: numeric Values: legal index into ITEMS array Default: index of first item

If the item list is too long to be displayed in the widget, you can move the list with this attribute; the selected array index moves to the top of the widget.

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                 TITLE (0)        USER DATA
VERSION            VISIBLE            WIDTH            X,Y
```

There is only one unique event associated with this widget:

SELECTION Occurs when user selects or deselects a LIST item.

[11.7] FILE WIDGET ATTRIBUTE REFERENCE

* The FILE widget has the following attributes:

DIRECTORY	Type: string Values: valid pathname

```
                       Default: current directory

   Gives directory to be listed.
   ______________________________________________________________________

   PATTERN                Type:    string
                          Values:  legal wildcard pattern
                          Default: *.* (all files)

   Specifies which files will be displayed.
   ______________________________________________________________________

   SELECTION              Type:    string
                          Values:  any file in the current directory
                          Default: "" (null string)

   Specifies selected file.
   ______________________________________________________________________
```

The default attributes and events also apply (the "0" means "level 0 only"):

```
   ______________________________________________________________________

   BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
   HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
   MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
   RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
   SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
   SYSTEM MENU event (0)                 TITLE (0)        USER DATA
   VERSION            VISIBLE            WIDTH            X,Y
   ______________________________________________________________________
```

There is one unique event associated with this widget:

```
   ______________________________________________________________________

   SELECTION      Occurs when the user selects a file.
   ______________________________________________________________________
```

[11.8] PRINTER WIDGET ATTRIBUTE REFERENCE

* The PRINTER widget has the following attributes:

```
   ______________________________________________________________________

   APPEND TEXT            Type:    string or string array
```

```
                       Values:  any valid string
                       Default: "" (the null string)
```

Specifies text to be displayed in PRINTER widget; existing text scrolls up.

```
COLUMNS                Type:    numeric
                       Values:  positive integer
                       Default: 50
```

Gives width of PRINTER widget in terms of the number of characters it can display.

```
CURRENT LINE           Type:    numeric
2.0 & later only       Values:  1 to VALID LINES
```

Line for use with CURRENT TEXT, INSERT LINE, and DELETE LINES attributes.

```
CURRENT TEXT           Type:    string or string array (read only)
2.0 & later only       Values:  valid strings
```

Text to be read, beginning with "CURRENT LINE".

```
DELETE LINES           Type:    numeric (write only)
2.0 & later only       Values:  0 to VALID LINES - CURRENT LINE + 1
```

Deletes the specified number of lines, starting with CURRENT LINE.

```
DISPLAY TEXT           Type:    numeric
                       Values:  0 or 1
2.0 & later only       Default: 1 (YES)
```

When 0, changes to the printer widget are not displayed.

```
ENABLE CLEAR DISP      Type:    numeric
                       Values:  0 or 1
2.0 & later only       Default: 1 (YES)
```

Allows "Clear Display" key to clear widget.

```
FONT                   Type:    string
                       Values:  legal font name
                       Default: varies
```

Gives name of font to be used in PRINTER widget.

```
HIDDEN LINES           Type:    numeric
                       Values:  positive integer
                       Default: 84
```

The number of lines in the PRINTER's text buffer (default 100), minus the number of lines that can be displayed (default 16).

```
HIGHLIGHT CURRENT      Type:    numeric
                       Values:  0 or 1
2.0 & later only       Default: 0 (NO)
```

When non-zero, current line is displayed in inverse video and current line can be set with mouse click.

```
INSERT TEXT            Type:    string or string array (write only)
2.0 & later only       Values:  valid strings
```

Inserts the given text above CURRENT LINE.

```
LINE WRAP              Type:    numeric
                       Values:  0 or 1
                       Default: 0 (no LINE WRAP)
```

If 0, text that is too wide for the display simply remains hidden beyond the right edge; the PRINTER provides a scrollbar so you can read that text. If 1, text simply folds over to the next line.

```
PEN                    Type:    numeric
                       Values:  valid pen number
                       Default: depends on system defaults
```

Specifies pen color of PRINTER font.

```
ROWS                   Type:    numeric
                       Values:  positive integer
                       Default: 16
```

Gives height of widget in terms of the number of rows of text it can display.

```
TAB STOP               Type:    numeric
                       Values:  0 or 1
                       Default: 1 (is TAB STOP)
```

Allows you to set up tab groups of widgets.

```
TEXT                   Type:    string or string array
                       Values:  any valid string
                       Default: "" (the null string)
```

Specifies text to be displayed in PRINTER widget; existing text is erased. (Use TEXT with a "" to erase PRINTER.)

```
TOP LINE               Type:    numeric
                       Values:  1 to VALID LINES
                       Default: 0 (no lines)
```

Specifies which line of text in the buffer will be at the top of the display (allows scrolling text under programmed direction).

```
VALID LINES            Type:    numeric
                       Values:  non-negative integer
                       Default: 0 (no lines)
```

Specified number of lines that have been written to PRINTER widget.

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                 TITLE (0)        USER DATA
VERSION            VISIBLE            WIDTH            X,Y
```

There are no unique events associated with the PRINTER widget.

[11.9] HPGL VIEW WIDGET ATTRIBUTE REFERENCE

* Note that the HPGL VIEW widget only supports a core subset of HPGL commands:

IP	Input scaling points P1 and P2.
IW	Input soft-clip window coordinates.
PA	Plot to absolute coordinates.
PD	Pen down.
PR	Plot to relative location.
PU	Pen up.
SP	Select Pen.

All others will be ignored. It has only 2 unique attributes:

```
HPGL FILE              Type:    string
                       Values:  legal file name
                       Default: ""
```

Specifies file containing HPGL commands.

```
RETAIN RASTER          Type:    numeric
                       Values:  0 or 1
                       Default: 1
```

If RETAIN RASTER is set, BPlus will take a bitmap "snapshot" of the final HPGL image in the widget, allowing it to be redrawn quickly. If the HPGL widget is then changed in some way that affects the image -- resizing it, say -- BPlus will take another snapshot.

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                 TITLE (0)        USER DATA
VERSION            VISIBLE            WIDTH            X,Y
```

The HPGL VIEW widget has no unique widget events.

[11.10] BITMAP WIDGET ATTRIBUTE REFERENCE (2.0 & LATER ONLY)

* The BITMAP widget includes the following attributes:

```
PEN                     Type:    numeric
                        Values:  legal PEN numbers (0-255)
                        Default: depends on system defaults
```

Sets color of LABEL.

```
FONT                    Type:    string
                        Values:  legal font specification
                        Default: depends on system defaults
```

Font used for LABEL.

```
BITMAP FILE             Type:    string
                        Values:  legal filenames
```

Specifies file to display.

```
LABEL                   Type:    string
                        Values:  any string
                        Default: "TITLE"
```

Text displayed beneath bitmap.

```
FRAME                   Type:    numeric
                        Values:  0 or 1
                        Default: 1 (YES)
```

When 1, a picture frame is drawn around the bitmap.

```
RETAIN RASTER           Type:    numeric
                        Values:  0 or 1
                        Default: 0 (OFF)
```

When 1, repaints won't reread file.

```
AUTO SIZE               Type:    numeric
                        Values:  0 or 1
                        Default: 1 (ON)
```

When 1, widget is automatically resized to size of bitmap

```
MAX PENS               Type:    numeric
                       Values:  1 to 256
                       Default: 256
```

Maximum number of pens bitmap can use.

```
PENS NEEDED            Type:    numeric (read only)
                       Values:  integer
```

Gives number of pens required by bitmap.

```
DUMP WINDOW            Type:    string (write only)
                       Values:  valid filename
```

Allows dumping of a window to a file; click on any visible window to dump it to specified bitmap file (click on non-client area and entire window is dumped; click on client area and only client area dumped)

```
DUMP FORMAT            Type:    string
                       Values:  "XWD" or "BMP"
                       Default: "XWD"
```

Specifies format of bitmap for dumping.

```
DUMP AREA              Type:    string (write only)
                       Values:  valid filename
```

Specifies a dump from the display to a specified file; click-and-drag across the specified area to select the dump area.

```
MOUSE CLICK            Type:    numeric X-Y (read only)
                       Values:  integer
```

Returns X-Y coordinates (in bitmap coordinates, relative to its upper left corner) of last mouse click. Values are returned to a two-element array.

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                 TITLE (0)        USER DATA
VERSION            VISIBLE            WIDTH            X,Y
```

There is only one unique event associated with this widget:

MOUSE CLICKED	Occurs when the left button is released over the bitmap.

[12.0] BPLUS (12): SLIDER, SCROLLBAR, & CLOCK Widgets

v2.4 / 01 feb 99 / greg goebel

* The SLIDER and SCROLLBAR input widgets allow a user to input a numeric value using a mouse. The SCROLLBAR widget is a subset of the SLIDER widget, so this chapter will discuss the SLIDER widget and then explain the differences between it and the SCROLLBAR widget. The CLOCK widget provides a digital or analog time-of-day clock, or a digital count-up/down counter. Attributes for these widgets are discussed at the end of the chapter.

[12.1] SLIDER WIDGET

* The SLIDER widget is used to provide numeric inputs. You can create a SLIDER widget as follows:

```
10    CLEAR SCREEN
20    ASSIGN @Slider TO WIDGET "SLIDER"
30    CONTROL @Slider;SET ("X":200,"Y":20,"WIDTH":120,"HEIGHT":300)
50    ON KEY 1 LABEL "  Quit  " GOTO Finis
60    ON EVENT @Slider,"DONE" GOSUB Dispval
80    LOOP
90    END LOOP
100   !
110 Dispval: !
120   STATUS @Slider;RETURN ("VALUE":Sliderval)
130   DISP Sliderval
140   RETURN
150   !
160 Finis: END
```

This yields:

```
+---------+---+
| SLIDER  | X |
+---------+---+
|  [ 100.0 ]  | <-- MAXIMUM limit
|      ^      |
|      |      |
|      |      |
|      |      |
|      |      |
|      |      |
|  [  0.0  ]  | <-- movable slidebar
|      v      |
|  [  0.0  ]  | <-- MINIMUM limit
+-------------+
```

You can take your mouse and move the slidebar in the center of the SLIDER widget; the value on the slidebar will change to reflect its position along the scale of 1 to 100 ... and the event generated in this program will also DISP that value. You can change this maximum range to anything you like, using the MINIMUM and MAXIMUM attributes:

```
   CONTROL @Slider; SET ("MINIMUM":-500,"MAXIMUM":500)
```

You can also click the mouse on the arrows at the ends of the scale, and the slidebar will move in the appropriate direction by 1. Click on the scale between an arrow and the slidebar, and the slidebar will move in that direction by 10. You can change these two increments, the MINOR INCREMENT and MAJOR INCREMENT, from their defaults of 1 and 10 to anything you like, say 5 and 25:

```
   CONTROL @Slider; SET ("MINOR INCREMENT":5, "MAJOR INCREMENT":25)
```

Now, when you click on the arrow the slider will move by an increment of 5. Note that we generate an event called DONE for the SLIDER, which means that an event will occur only when you have released the mouse button after moving the slidebar. There is an alternate event called CHANGED which allows you to continuously interrupt while the slidebar is being moved.

By default, if you click at the ends of the scale, the SLIDER only increments once. An attribute added in BPlus 2.0, AUTO REPEAT, allows the SLIDER to increment as long as the mouse button is depressed (or runs into the end).

Note that we can also move the slidebar using various keys:

up / right cursor:	Increment by MINOR INCREMENT.
down / left cursor:	Decrement by MINOR INCREMENT.
shift up / right cursor:	Move to MAXIMUM.

```
   shift down / left cursor:    Move to MINIMUM.
   Page Up (Prev):              Decrement by MAJOR INCREMENT.
   Page Down (Next):            Increment by MAJOR INCREMENT.
```

The SLIDER has a number of other interesting features:

* If you set an attribute called DIRECT MOVE to 1, then when you click on the space between an arrow and the slidebar, the slidebar will move directly to that position. Of course, DIRECT MOVE disables MAJOR INCREMENT.

* You can set an attribute called LOGARITHMIC to 1, and then the slidebar will increment in a logarithmic fashion.

* You can set the ORIENTATION attribute of the SLIDEBAR to HORIZONTAL, as an alternative to the default orientation of VERTICAL, and the SLIDEBAR will take a horizontal format.

There are a number of other attributes: you can turn the limits boxes on and off, and specify (to an extent) numeric formats.

* Now let's build a SLIDER demo. It provides a user interface as follows:

```
   +------------------------------------------+
   |                SLIDER Test               |
   +------------------------------------------+
   | Menu                                     |
   +------------------------------------------+
   |   +-----------------+    +------------+   |
   |   |      0.1        |    | [ 100.0 ] |   |
   |   +-----------------+    |     ^      |   |
   |                          |     |      |   |
   |                          |     |      |   |
   |                          |     |      |   |
   |                          |     |      |   |
   |                          |     |      |   |
   |                          |     |      |   |
   |                          |     |      |   |
   |                          |     |      |   |
   |                          |     |      |   |
   |                          | [  0.0  ] |   |
   |                          |     v      |   |
   |                          | [  0.0  ] |   |
   |                          +------------+   |
   +------------------------------------------+
```

The menu lets you toggle LOGARITHMIC or DIRECT MOVE mode, or exit the program:

```
| User                                          |
+-----------------+-----------------------+
| Set LOG Mode    |                       |
| Set DIRECT MOVE |
+-----------------+
| Quit            |
+-----------------+
|
```

Program Example: xslider.rmb

[12.2] SCROLLBAR WIDGET

* The SCROLLBAR widget is a "stripped-down" version of the SLIDER widget; the designed purpose of the SCROLLBAR widget is to allow a customer to "scroll" a PANEL that is too large for the display.

The SCROLLBAR widget has no LOGARITHMIC mode; it can only be operated in a linear fashion. It also counts in the reverse direction from a SLIDER: if you move the slide up in a SLIDER, the VALUE increments; if you do that in a SCROLLBAR, it decrements. The appearance of the SCROLLBAR is substantially different from that of a SLIDER as well: all it consists of is a trough with a slide, with arrows at each end.

One peculiarity of the SCROLLBAR is that its WIDTH cannot be set; it has a given WIDTH on a given display, and to position the SCROLLBAR properly you have to read its WIDTH and then move the widget accordingly. (I am referring to WIDTH given the default ORIENTATION of VERTICAL; change the ORIENTATION to HORIZONTAL, and the same comments then apply to HEIGHT instead of WIDTH.)

* However, once you understand these differences, then the SCROLLBAR widget is very little different from the SLIDER widget in terms of setting it up and extracting events from it. You can set MAXIMUM and MINIMUM, MINOR and MAJOR INCREMENT and DIRECT MOVE; and set up CHANGED and DONE events on this widget to obtain its VALUE.

SCROLLBARs are very handy and will are used in other demos in this document.

[12.3] CLOCK WIDGET

* The CLOCK widget provides either a time-of-day clock or an up-down timer. The mode of operation is set by the TYPE attribute, which can be ANALOG, DIGITAL, MIXED (both ANALOG and DIGITAL), or TIMER. You can set an ALARM event on the CLOCK and specify an ALARM TIME (which is in the form of an

"HH:MM:SS" string) for the event to occur.

In TIMER mode, you get a digital timer that counts from a time in milliseconds down to 0, or the reverse. You specify the direction with TIMER DIRECTION, and the initial count with TIMER VALUE; then initiate the count by setting TIMER STATE to RUNNING. (You set it to STOPPED to stop the timer, and to RESET to clear the timer.) There's a TIMER event that occurs when the count completes; you can perform TIMER actions cyclically by setting TIMER REPEAT to 1.

* The demo program for this section uses the CLOCK widget to build an alarm clock. The program has the layout:

```
   +---+------------------------------------------------+
   | = |               Alarm Clock                      |
   +---+------------------------------------------------+
   | +-----------------+ +------------+ +-------------+ |
   | |        *        | |            | |             | |
   | |    *   |   *    | |   Enable   | | KILL ALARM  | |
   | |        |        | |            | |             | |
   | |  * ----*     *  | +------------+-+-------------+ |
   | |                 | |                            | |
   | |    *       *    | |         08:30:00           | |
   | |        *        | |                            | |
   | +-----------------+ +----------------------------+ |
   | +------------------------------------------------+ |
   | |<                                              >| |
   | +------------------------------------------------+ |
   +----------------------------------------------------+
```

Program Example: xclock.rmb

[12.4] SLIDER WIDGET ATTRIBUTE REFERENCE

* The SLIDER widget has the following attributes:

AUTO REPEAT	Type:	numeric
	Values:	0 or 1
2.0 & later only	Default:	0 (OFF)

If 1, the value of the slider will be periodically incremented when the user presses the arrow or the trough. The increment times can be set in the CONFIG file with the "initial_delay" and "repeat_delay" variables.

BACKGROUND	Type:	numeric

	Values: legal PEN numbers (0-255)
2.0 & later only	Default: depends on system defaults

Sets the color of the background of the slider.

COLUMNS	Type: numeric
	Values: positive integer
	Default: 7

Specifies the width of the SLIDER in terms of the number of characters that can be contained in its width.

DECIMAL DIGITS	Type: numeric
	Values: 0 to 15
	Default: 1

This attribute specifies the number of digits after the decimal point that will be displayed in the slidebar.

DIRECT MOVE	Type: numeric
	Values: 0 or 1
	Default: 0 (MAJOR INCREMENT in effect)

If 1, clicking the mouse on any location on the SLIDER moves the indicator to the clicked location (overrides MAJOR INCREMENT attribute).

DISPLAY BACKGROUND	Type: numeric
	Values: legal PEN numbers (0-255)
2.0 & later only	Default: depends on system defaults

Sets the color of the minimum/maximum/value backgrounds.

EXPONENTIAL FORMAT	Type: numeric
	Values: 0 or 1
	Default: 0 (fixed-point format)

If set to 1, the numbers in the value and limits boxes are in exponential format, instead of the default fixed-point format.

FONT	Type: string
	Values: legal font-file name
	Default: depends on system defaults

Specifies font to be used in VALUE and limits boxes.

```
LOGARITHMIC            Type:    numeric
                       Values:  0 (linear) or 1 (logarithmic)
                       Default: 0 (linear)
```

If 1, numeric incrementing is log-scaled; if 0, linear.

```
MAJOR INCREMENT        Type:    numeric
                       Values:  >= 0
                       Default: 10
```

Specifies the amount by which the SLIDER increments or decrements if the user clicks on the SLIDER trough.

```
MAXIMUM                Type:    numeric
                       Values:  any real number
                       Default: 100
```

Specifies the top numeric limit of the SLIDER.

```
MINIMUM                Type:    numeric
                       Values:  any real number
                       Default: 1
```

Specifies the bottom numeric limit of the SLIDER.

```
MINOR INCREMENT        Type:    numeric
                       Values:  >= 0
                       Default: 1
```

Specifies the amount by which the SLIDER increments or decrements if the user clicks on the SLIDER arrows.

```
ORIENTATION            Type:    string
                       Values:  "VERTICAL" or "HORIZONTAL"
                                "UP" or "RIGHT"
                       Default: "VERTICAL"
```

Specifies orientation of the SLIDER.

```
PEN                    Type:    numeric
                       Values:  legal PEN numbers (0-255)
2.0 & later only       Default: depends on system defaults
```

Sets text and slot border color.

SHOW LIMITS	Type: numeric Values: 0 or 1 Default: 1 (limits boxes are visible)

If SHOW LIMITS is set to 0, then the MINIMUM and MAXIMUM boxes disappear.

SHOW VALUE	Type: numeric Values: 0 or 1 Default: 1 (value box is visible)

If SHOW VALUE is set to 0, then the value box on the slidebar disappears.

TAB STOP	Type: numeric Values: 0 or 1 Default: 1 (is TAB STOP)

If 1, allows you to "tab" into SLIDER and use it with cursor keys.

VALUE	Type: numeric Values: any real number Default: 0

This specifies the current value of the SLIDER.

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                 TITLE (0)        USER DATA
VERSION            VISIBLE            WIDTH            X,Y
```

There are two unique events associated with the SLIDER:

CHANGED   Indicates SLIDER has moved and value has changed.

DONE	Indicates mouse button released on SLIDER.

[12.5] SCROLLBAR WIDGET ATTRIBUTE REFERENCE

* The SCROLLBAR widget has the following attributes:

```
DIRECT MOVE            Type:    numeric
                       Values:  0 or 1
                       Default: 0 (MAJOR INCREMENT in effect)
```

If 1, clicking the mouse on any location on the SCROLLBAR moves the slidebar to the clicked location (overrides MAJOR INCREMENT attribute).

```
MAJOR INCREMENT        Type:    numeric
                       Values:  any positive real
                       Default: 10
```

Specifies the amount by which the SCROLLBAR increments or decrements if the user clicks on the SCROLLBAR trough.

```
MAXIMUM                Type:    numeric
                       Values:  any real number
                       Default: 100
```

Specifies the top numeric limit of the SCROLLBAR.

```
MINIMUM                Type:    numeric
                       Values:  any real number
                       Default: 0
```

Specifies the bottom numeric limit of the SCROLLBAR.

```
MINOR INCREMENT        Type:    numeric
                       Values:  any positive real
                       Default: 1
```

Specifies the amount by which the SCROLLBAR increments or decrements if the user clicks on the SCROLLBAR arrows.

ORIENTATION	Type:	string
	Values:	"VERTICAL" or "HORIZONTAL"
		"UP" or "RIGHT"
	Default:	"VERTICAL"

Specifies direction of the SCROLLBAR.

TAB STOP	Type:	numeric
	Values:	0 or 1
	Default:	1

If 1, allows you to "tab" into SCROLLBAR and move it with arrow keys.

VALUE	Type:	numeric
	Values:	any real number
	Default:	0

This specifies the current value of the SCROLLBAR.

The default attributes and events also apply (the "0" means "level 0 only"):

BACKGROUND	BORDER	CLOSE event (0)	HEIGHT
HELP FILE	HELP TOPIC	INSIDE HEIGHT	INSIDE WIDTH
MAXIMIZABLE (0)	MINIMIZABLE (0)	MOVABLE (0)	RESIZABLE (0)
RESTORE SCREEN (0)	STACKING ORDER	SYSTEM MENU (0)	
SYSTEM MENU COUNT (0)		SYSTEM MENU EVENT (0)	
SYSTEM MENU event (0)		TITLE (0)	USER DATA
VERSION	VISIBLE	WIDTH	X,Y

There are two unique events associated with the SCROLLBAR:

CHANGED	Indicates SCROLLBAR has moved and VALUE has changed.
DONE	Indicates mouse button released on SCROLLBAR.

[12.6] CLOCK WIDGET ATTRIBUTE REFERENCE (2.0 & LATER ONLY)

* The CLOCK widget supports the following attributes:

```
ALARM TIME              Type:     string
                        Values:   time in format HH:MM:SS (24 hour)
                        Default:  00:00:00
```

Sets the time that the alarm will be generated.

```
FONT                    Type:    string
                        Values:  legal font specification
                        Default: depends on system defaults
```

Font used for text.

```
PEN                     Type:    numeric
                        Values:  legal PEN numbers (0-255)
                        Default: depends on system defaults
```

Sets foreground color.

```
SECONDS VISIBLE         Type:     numeric
                        Values:   0 or 1
                        Default:  1 (visible)
```

Determines whether seconds will be visible or not.

```
TIMER DIRECTION         Type:    string
                        Values:  "UP" or "DOWN"
                        Default: "UP"
```

Specifies whether timer is a count-up or count-down timer.

```
TIMER REPEAT            Type:    numeric
                        Values:  0 or 1
                        Default: 0 (don't repeat)
```

Specifies whether timer repeats its count when it times out, or not.

```
TIMER STATE             Type:    string (read only)
                        Values:  "RESET" or "STOPPED" or "RUNNING"
                        Default: "RESET"
```

Returns status of timer when TYPE is TIMER, returns invalid attribute error otherwise.

TIMER UPDATE	Type:	numeric
	Values:	1 to 32,767
	Default:	1

Sets timer to update every TIMER UPDATE millseconds.

TIMER VALUE	Type:	numeric
	Values:	0 to 21,474,836,447
	Default:	0

Specifies duration of timer in milliseconds.

TYPE	Type:	string
	Values:	"ANALOG" or "DIGITAL" or "MIXED" or "TIMER"
	Default:	"ANALOG"

Sets the type of clock to be displayed: ANALOG, DIGITAL, or MIXED (both) and whether the clock is set up as a TIMER or not.

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                 TITLE (0)        USER DATA
VERSION            VISIBLE            WIDTH            X,Y
```

There are two unique events associated with this widget:

ALARM	Occurs when the time reaches the time specified in the ALARM TIME attribute or within the ALARM RESOLUTION attribute.

TIMER	Occurs when TIMER counts out.

[13.0] BPLUS (13): BAR, METER, LIMITS, and BARS Widgets

v2.4 / 01 feb 99 / greg goebel

* The BAR, METER, and LIMITS widgets provide numeric-value output indicators; the BAR provides a single bar indicator, the METER provides a single moving-needle indicator, and the LIMITS widget provides a moving-marker indicator. The BARS widget is a variation on the BAR widget that provdes a multiple-bar display. This chapter provides a description of these widgets and a detailed reference for their attributes.

Note that the attribute reference in this chapter differs from that in others in that attributes common to the BAR, METER, and LIMITS widgets are discussed in their own section, while the "deltas" from the common attributes are discussed in individual sections. This not only makes the discussion more compact, but also highlights the most important attributes.

[13.1] BAR WIDGET

* You can demonstrate a BAR widget with the following simple program:

```
   10    CLEAR SCREEN
   15    INTEGER N,M
   20    ASSIGN @Bar TO WIDGET "BAR"
   30    CONTROL @Bar;SET ("X":100,"Y":20,"WIDTH":200,"HEIGHT":300)
   40    CONTROL @Bar;SET ("ALARM RANGES":"LOW,HIGH","ALARM TYPE":"BEEP")
   50    CONTROL @Bar;SET ("LOW LIMIT":10,"HIGH LIMIT":90)
   60    !
   80    FOR M=1 TO 3
   90      FOR N=1 TO 100
   100       WAIT .15
   110       CONTROL @Bar;SET ("VALUE":N)
   120     NEXT N
   130     WAIT 1
   140   NEXT M
   150   !
   160   END
```

This creates the following widget:

```
   +--------------+---+
   |       BAR    | X |
   +--------------+---+
   |                  |
   |                  |
   |                  |
   |                  |
   |       +--+       |
   |       |  |       |
   |       |  |       |
   |       |  |       |
   +-------+--+-------+
```

As the program runs, the bar creeps up from the bottom to the top; an important point to note is that it does this under program control, by writes to the VALUE attribute. (There is a tendency when you first see the BAR widget and its kin that they operate autonomously, but no: you have to write VALUEs to them or they just sit there.)

Since we set the HIGH LIMIT to 90 and the LOW LIMIT to 10, we set the ALARM RANGES to "LOW,HIGH", and we set the ALARM TYPE to "BEEP" (actually the default), it beeps ten times at the bottom and ten times at the top.

Beeping being thoroughly obnoxious, we can change our example program to generate an event in the LOW and HIGH ranges instead:

```
   40    CONTROL @Bar;SET ("ALARM RANGES":"LOW,HIGH","ALARM TYPE":"EVENT")
   50    CONTROL @Bar;SET ("LOW LIMIT":10,"HIGH LIMIT":90)
   55    ON EVENT @Bar,"ALARM" GOSUB Handler
   ...
   170 Handler: !
   180   STATUS @Bar;RETURN ("VALUE":Nval)
   190   DISP Nval
   200   RETURN
```

This displays the values from 1 to 10 and from 90 to 100. As you can see, operation of the BAR widget is very simple, although event handling is a little devious; there's three simple rules to remember:

* You can get an event on any or all of the three ranges: "LOW","MIDDLE", or "HIGH".

* The events are "level-triggered", not "edge-triggered": that is, you

get an event any time you have a VALUE in the designated ALARM RANGE, not merely when the VALUE transitions from one range to the other.

* You have to specifically set the ALARM TYPE to event before you get an event, otherwise all you get is beeps.

The BAR widget of course has other attributes: you can set MINIMUM and MAXIMUM limit values; you can set it to LOGARITHMIC-scaled operation as well. You can set its ORIENTATION to HORIZONTAL, as opposed to the default VERTICAL ORIENTATION shown here. And you can set different PEN colors for the LOW, MIDDLE, and HIGH ranges.

[13.2] METER WIDGET

* The METER widget looks very different from the BAR widget, but from the programming point of view its operation is virtually identical (with a few additions). In fact, we can slightly modify our short BAR demo program to run with a METER widget, as follows:

```
10    CLEAR SCREEN
15    INTEGER N,M
20    ASSIGN @M TO WIDGET "METER"
30    CONTROL @M;SET ("X":100,"Y":20,"WIDTH":300,"HEIGHT":300)
40    CONTROL @M;SET ("ALARM RANGES":"LOW,HIGH","ALARM TYPE":"BEEP")
50    CONTROL @M;SET ("LOW LIMIT":10,"HIGH LIMIT":90)
60    CONTROL @M;SET ("SWEEP ANGLE":360)
70    !
90    FOR M=1 TO 3
100     FOR N=1 TO 100
110       WAIT .15
120       CONTROL @M;SET ("VALUE":N)
130     NEXT N
140     WAIT 1
150   NEXT M
160   !
170   END
```

This yields a widget that resembles:

```
                     +---------------------------+---+
                     |             METER         | X |
                     +---------------+-----------+---+
   MINIMUM limit --> |       1       |      100      | <-- MAXIMUM limit
                     +---------------+---------------+
                     |            *  *  *            |
                     |        *             *        |
                     |     *                   *     |
                     |   *                       *   |
                     |  *                         *  |
                     | *                           * |
                     | * ------------              * |
                     | *                           * |
                     |  *                         *  |
                     |   *                       *   |
                     |     *   *           *   *     |
                     |        *             *        |
                     |       *               *       |
                     |                               |
                     +-------------------------------+
                     |               25              | <-- VALUE box
                     +-------------------------------+
```

The appearance of the indicator itself is not only different, but the METER widget also displays its MIMIMUM limit, MAXIMUM limit, and current VALUE in boxes. (Note that I set the SWEEP ANGLE of the meter to 360 degrees to give this widget a tachometer-like appearance; normally the sweep angle is 60 degrees and it more resembles the magnetic-movement meter on an old-style voltmeter.)

The comments on BAR widget operation and event handling in the last section apply perfectly to the METER widget: most attributes for the BAR widget described work the same on the METER. Of course, with the different appearance, there are many more attributes available to modify it. I already showed how to change the SWEEP ANGLE; you can also change the ARC WIDTH and NEEDLE WIDTH (in units of pixels). You can set background and font colors for the meter itself, for the limits boxes, or for the VALUE box. You can turn the LIMITS or VALUE boxes off, if you like.

[13.3] LIMITS WIDGET

* The LIMITS widget is another form of the BAR or METER widget; in the case of the LIMITS widget, a range of values is displayed, while a marker moves through the range -- rather like a "flattened" version of a METER widget. For example:

```
   ASSIGN @Lim TO WIDGET "LIMITS"
   CONTROL @Lim; SET ("MINIMUM":0,"MAXIMUM":100)
   CONTROL @Lim; SET ("LOW LIMIT":10,"HIGH LIMIT":90)
```

-- gives:

```
   +------------------------------+---+
   |              LIMITS          | X |
   +------------------------------+---+
   |    [ 10 ]            [ 90 ]      |
   |                                  |
   |   +--+-------------------+--+    |
   |   |  |           I       |  |    |
   |   +--+-------------------+--+    |
   |                                  |
   | [  0  ]                [ 100 ]   |
   +----------------------------------+
   |                63                |
   +----------------------------------+
```

The limits boxes at top move as the LOW LIMIT and HIGH LIMIT are changed. As with the METER widget, you can set the colors of the VALUE and LIMITS boxes, but you can't turn them on or off.

However, the LIMITS widget differs from the other two in the fact that it does not support LOW PEN / MIDDLE PEN / HIGH PEN -- instead it only supports OUTSIDE PEN (the color outside the limits) and INSIDE PEN (the color inside the limits) -- you can't set the upper and lower color bounds to different colors.

Program Example: xbarmtlm.rmb

[13.4] BARS WIDGET

* The BARS widget is a variation on the BAR widget; it's like an assemblage of BAR widgets in the same box. Programming it is very similar to programming the BAR widget, with some extensions. If you simply create a BARS widget without specifying much in the way of attributes, say with the following simple program:

```
   10    CLEAR SCREEN
   15    INTEGER M,N
   20    ASSIGN @Bars TO WIDGET "BARS"
   30    CONTROL @Bars;SET ("BAR LABEL":"Junk")
   40    ON KBD ALL GOTO Finis
   50    FOR M=1 TO 5
   60      FOR N=1 TO 100
   70        CONTROL @Bars;SET ("VALUE":N)
   80      NEXT N
   90    NEXT M
   100   END
```

-- you get the following widget:

```
                     +-----------------------+---+
                     |           BARS        | X |
                     +-----------------------+---+
   MAXIMUM limit --> | [ 100 ] +---------------+ |
                     |         |               | |
                     |         |               | |
                     |         |               | |
                     |         |               | |
                     |         |               | |
                     |         |               | |
                     |         |     +---+     | |
                     |         |     |   |     | |
                     |         |     |   |     | |
   MINUMUM limit --> | [  1  ] +-----+---+-----+ |
                     |         [       24      ] | <-- VALUE box
                     |         [      Junk     ] | <-- LABEL box
                     +---------------------------+
```

This doesn't look much different from a BAR widget, enhanced with some of the nice features of the METER widget, like the limits boxes to the left of the bar and the value and label boxes beneath it; and in fact, this particular example can be programmed in a fashion virtually identical to that used with the BAR widget.

However, let's tweak our little thrown-together program a bit to add some more bars:

```
   10    CLEAR SCREEN
   15    INTEGER M,N
   20    REAL Barval(1:3)
   30    ASSIGN @Bars TO WIDGET "BARS"
   40    CONTROL @Bars;SET ("HEIGHT":300,"WIDTH":400)
   50    !
   60    CONTROL @Bars;SET ("BAR COUNT":3)
   70    CONTROL @Bars;SET ("CURRENT BAR":1)
   80    CONTROL @Bars;SET ("BAR LABEL":"Number")
   90    CONTROL @Bars;SET ("CURRENT BAR":2)
   100   CONTROL @Bars;SET ("BAR LABEL":"Root")
   101   CONTROL @Bars;SET ("CURRENT BAR":3)
   102   CONTROL @Bars;SET ("BAR LABEL":"Ln")
   110   !
   130   FOR M=1 TO 5
   140     FOR N=1 TO 100
   150       Barval(0)=N
   160       Barval(1)=SQR(N)
   170       Barval(2)=LOG(N)
   180       CONTROL @Bars;SET ("VALUES":Barval(*))
   190     NEXT N
   200   NEXT M
   220   END
```

This gives:

```
   +--------------------------------------+---+
   |                    BARS              | X |
   +--------------------------------------+---+
   | [ 100 ] +------------------------------+ |
   |         |                              | |
   |         |                              | |
   |         | +---+                        | |
   |         | |   |                        | |
   |         | |   |                        | |
   |         | |   |                        | |
   |         | |   |                        | |
   |         | |   |      +---+             | |
   |         | |   |      |   |      +---+  | |
   | [  1  ] +-+---+------+---+------+---+--+ |
   |         [   64   ] [   8    ] [  4.16  ] |
   |         [ Number ] [  Root  ] [  Ln    ] |
   +------------------------------------------+
```

In this new program, we specified three bars:

```
   CONTROL @Bars;SET ("BAR COUNT":3)
```

-- gave individual labels to each (through selection with the CURRENT BAR attribute):

```
   CONTROL @Bars;SET ("CURRENT BAR":1)
   CONTROL @Bars;SET ("BAR LABEL":"Number")
   CONTROL @Bars;SET ("CURRENT BAR":2)
   CONTROL @Bars;SET ("BAR LABEL":"Root")
   CONTROL @Bars;SET ("CURRENT BAR":3)
   CONTROL @Bars;SET ("BAR LABEL":"Ln")
```

-- and then updated all three at once using the VALUES (not VALUE!) attribute with an array parameter:

```
   Barval(0)=N
   Barval(1)=SQR(N)
   Barval(2)=LOG(N)
   CONTROL @Bars;SET ("VALUES":Barval(*))
```

This shows the essential features of the BARS widget versus the BAR widget:

* You can set multiple bars using the BAR COUNT attribute.
* You can select each bar and modify it using the CURRENT BAR attribute.
* You can update all bars in parallel using the VALUES attribute with an array parameter.

* Once you understand these three points, you basically understand the BARS widget; it can appear extremely complicated if you look at the attribute listing in the command reference manual, but at root it's pretty simple.

However, there are a large number of niceties to consider. The most important is that there are some attributes that you can set for each individual bar and some attributes that are set for all the bars together.

For example, you can set the HIGH, MIDDLE, and LOW limits and the corresponding PEN colors for each individual bar; however, the MAXIMUM and MINIMUM range is the same for all (obvious if you think about it). You also cannot individually set the colors for the BACKGROUND and PENS in the LABEL and VALUE boxes for the bars; nor can you selectively turn on LABEL or VALUE boxes for certain bars, but not for others.

One interesting effect of this is that if you set an event for one of the bars taking a value within a certain range, you set that event using the widget handle for the entire widget; there's no other way to do it! And if you get an event you then have to check all the bars to see which one did it.

There are the usual wide variety of attribute options for this widget -- you can turn off the LIMITS or VALUE or LABEL boxes, set all kinds of colors, and all that jazz. BPlus 2.0 added some minor features, such as the ability to set up a very large number of bars and then scroll them into view (by pixel count or by bar); draw lines to indicate limits; and so on.

Program Example: xbars.rmb

[13.5] COMMON BAR / METER / LIMIT ATTRIBUTES

* The following attributes are common to all three widgets:

```
ALARM RANGES           Type:    string
                       Values:  "" or "LOW" or "MIDDLE" or "HIGH"
                       Default: "" (null string)
```

Specifies ranges of values that will trip an alarm.

```
ALARM TYPE             Type:    string
```

	Values: "BEEP" or "EVENT"
	Default: "BEEP"

Specifies alarm type -- either a simple BEEP or an interrupting EVENT.

AUTOSCALE	Type: numeric
	Values: 0 or 1
	Default: 0 (OFF)

If 1, the widget will automatically adjust the MAXIMUM and MININUM limits when given a new value that exceeds the old limits.

HIGH LIMIT	Type: numeric
	Values: any real number (positive if log)
	Default: MAXREAL

Specifies a high bound on meter values; values exceeding this limit can trip an alarm.

LOGARITHMIC	Type: numeric
	Values: 0 (linear) or 1 (logarithmic)
	Default: 0 (linear)

If 1, specifies log scale; if 0, linear.

LOW LIMIT	Type: numeric
	Values: any real number (positive if log)
	Default: MAXREAL

Specifies a low threshold on widget values; values falling below this limit can trip an alarm.

MAXIMUM	Type: numeric
	Values: any real number
	Default: 100

Specifies the top limit of the widget range.

MINIMUM	Type: numeric
	Values: any real number
	Default: 1

Specifies bottom limit of widget's range.

ORIENTATION	Type: string

```
                        Values:  "UP" or "DOWN" or "LEFT" or "RIGHT"
                                 "VERTICAL" or "HORIZONTAL"

   Specifies the orientation of the widget.
```

```
   VALUE                Type:    numeric
                        Values:  any real number
                        Default: 0

   The value you want the widget to display.
```

The default attributes and events also apply (the "0" means "level 0 only"):

```
   BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
   HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
   MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
   RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
   SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
   SYSTEM MENU event (0)                 TITLE (0)        USER DATA
   VERSION            VISIBLE            WIDTH            X,Y
```

There is only one unique event associated with these widgets:

```
   ALARM  Occurs when VALUE falls in the range of the of the specified
          ALARM RANGES.
```

[13.6] SPECIFIC BAR ATTRIBUTES

* The BAR widget has the following specific attributes:

```
   BAR WIDTH            Type:    numeric
                        Values:  any positive integer
                        Default: 25

   Gives width in pixels of visible portion of bar.
```

HIGH PEN	Type: numeric
	Values: legal PEN numbers (0-255)
	Default: depends on system defaults

Specifies pen color of bar above HIGH LIMIT (ignored on mono monitors).

LIMITS LINE	Type: numeric
	Values: 0 or 1
2.0 & later only	Default: 0 (NO)

Draws lines pointing to where the low limit and high limits are for the bar.

LOW PEN	Type: numeric
	Values: legal PEN numbers (0-255)
	Default: depends on system defaults

Specifies pen color below LOW LIMIT (ignored on one-plane mono monitors).

MIDDLE PEN	Type: numeric
	Values: legal PEN numbers (0-255)
	Default: depends on system defaults

Specifies pen color of bar between LOW LIMIT and HIGH LIMIT (ignored on one-plane mono monitors).

ORIGIN	Type: numeric
	Values: REAL
2.0 & later only	Default: -MAXREAL

Moves the origin of the bar to the specified location.

ORIGIN PEN	Type: numeric
	Values: legal PEN numbers (0-255)
2.0 & later only	Default: black

Sets color of origin line.

[13.7] SPECIFIC METER WIDGET ATTRIBUTES

* The METER widget has the following specific attributes:

ARC WIDTH	Type: numeric Values: any positive integer Default: 2

Width in pixels of the line used to draw the meter arc.

CENTER 2.0 & later only	Type: numeric Values: 0 or 1 Default: 0 (NO)

Centers the meter in the middle of the panel. This is useful especially for sweep angles > 180.

FONT	Type: string Values: legal font-file name Default: depends on system defaults

Gives name of font to be used in labels.

HIGH PEN	Type: numeric Values: legal PEN numbers (0-255) Default: depends on system defaults

Specifies pen color of meter above HIGH LIMIT.

LIMITS BACKGROUND	Type: numeric Values: valid PEN number (0-255) Default: depends on system defaults

Defines the color of the limits backgrounds.

LIMITS PEN	Type: numeric Values: valid PEN number (0-255) Default: depends on system defaults

Defines the color of the limits-box characters.

LOW PEN	Type: numeric Values: legal PEN numbers (0-255) Default: depends on system defaults

Specifies pen color below LOW LIMIT.

```
METER BACKGROUND       Type:    numeric
                       Values:  legal PEN numbers (0-255)
                       Default: depends on system defaults

Specifies pen color of meter background.
```

```
MIDDLE PEN             Type:    numeric
                       Values:  legal PEN numbers (0-255)
                       Default: depends on system defaults

Specifies pen color of bar between LOW LIMIT and HIGH LIMIT.
```

```
NEEDLE PEN             Type:    numeric
                       Values:  legal PEN numbers (0-255)
                       Default: depends on system defaults

Specifies needle pen color.
```

```
NEEDLE WIDTH           Type:    numeric
                       Values:  non-negative integer
                       Default: 1

Specifies width of needle in pixels.
```

```
SHOW LIMITS            Type:    numeric
                       Values:  0 or 1
                       Default: 1 (show limits in boxes)

If 1, values written to meter are also displayed in a value box.
```

```
SHOW VALUE             Type:    numeric
                       Values:  0 or 1
                       Default: 1 (show values in box)

If 1, values written to meter are also displayed in a value box.
```

```
SWEEP ANGLE            Type:    numeric
                       Values:  1 to 360 (degrees)
                       Default: 60 degrees

Range of meter deflection, in degrees.
```

```
TICK LENGTH            Type:    numeric
```

```
                        Values:  any positive integer
                        Default: 12 pixels
```

Specifies length in pixels of tick marks at boundaries of meter arc.

```
TICK LOCATION           Type:    string
                        Values:  "CENTER" or "OUTSIDE" or "INSIDE"
2.0 & later only        Default: "CENTER"
```

Allows tick mark to be placed inside of the arc, outside of the arc, or centered in the arc.

```
TICK PEN                Type:    numeric
                        Values:  valid PEN number (0-255)
                        Default: varies
```

Specifies pen color of tick marks.

```
VALUE BACKGROUND        Type:    numeric
                        Values:  valid PEN number (0-255)
                        Default: depends on system defaults
```

Defines the pen color of the background of the meter value and limits boxes.

```
VALUE PEN               Type:    numeric
                        Values:  valid PEN number (0-255)
                        Default: depends on system defaults
```

Defines the pen color of the fonts in the value and limits boxes.

[13.8] SPECIFIC LIMITS WIDGET ATTRIBUTES (2.0 & LATER ONLY)

* The LIMITS widget has the following specific attributes.

```
OUTSIDE PEN             Type:    numeric
                        Values:  legal PEN numbers (0-255)
                        Default: depends on system defaults
```

Specifies pen color of bar outside of limits (ignored on mono monitors).

INSIDE PEN	Type:	numeric
	Values:	legal PEN numbers (0-255)
	Default:	depends on system defaults

Specifies pen color within limits (ignored on one-plane mono monitors).

LIMITS BACKGROUND	Type:	numeric
	Values:	legal PEN numbers (0-255)
	Default:	depends on system defaults

Specifies pen color of limit boxes (ignored on one-plane mono monitors).

LIMITS PEN	Type:	numeric
	Values:	legal PEN numbers (0-255)
	Default:	depends on system defaults

Specifies pen color of limit value font (ignored on one-plane mono monitors).

LIMITS WIDTH	Type:	numeric
	Values:	any positive integer
	Default:	depends on system defaults

Gives width in pixels of widget.

MARKER PEN	Type:	numeric
	Values:	legal PEN numbers (0-255)
	Default:	depends on system defaults

Specifies pen color of limits marker (ignored on one-plane mono monitors).

MARKER WIDTH	Type:	numeric
	Values:	any positive integer
	Default:	?

Gives width in pixels of widget marker.

VALUE BACKGROUND	Type:	numeric
	Values:	legal PEN numbers (0-255)
	Default:	depends on system defaults

Specifies pen color of widget value background (ignored on one-plane mono monitors).

VALUE PEN	Type: numeric Values: legal PEN numbers (0-255) Default: depends on system defaults

Specifies pen color of widget value (ignored on one-plane mono monitors).

[13.9] BARS WIDGET ATTRIBUTE REFERENCE

* The BARS widget, as noted, has two levels of attributes: those that affect the entire widget and those that affect an individual bar (selected by the CURRENT BAR attribute).

The overall widget attributes consist of:

AUTOPOSITION	Type: numeric Values: 0 or 1 Default: 1 (ON)

If set to 1, the BARS widget will automatically adjust the value of the BAR SEPARATION attribute so that the single-bars evenly fill the areas designated for the BARS workspace. This value is affected by the WIDTH or BAR WIDTH attribute.

AUTOSCALE	Type: numeric Values: 0 or 1 Default: 1 (ON)

If 1, the widget will automatically adjust the MAXIMUM and MININUM limits when given a new value that exceeds the old limits.

BAR BACKGROUND	Type: numeric Values: valid PEN number (0-255) Default: depends on system defaults

Defines the color of the bars backdrop.

BAR COUNT	Type: numeric Values: any positive integer Default: 1 (one BAR)

Gives the number of bars in the BARS widget.

```
BAR SEPARATION         Type:    numeric
                       Values:  any positive integer
                       Default: depends on WIDTH & BAR WIDTH
```

Gives the number of pixels between bars; if you set BAR SEPARATION to any value, AUTOPOSITION is automatically disabled (set to 0).

```
BAR WIDTH              Type:    numeric
                       Values:  any positive integer
                       Default: 25
```

Gives the width of each bar in pixels.

```
CURRENT BAR            Type:    numeric
                       Values:  1 to BAR COUNT
                       Default: 1
```

Selects one of the bars for access with individual attributes.

```
FIRST BAR              Type:    numeric
                       Values:  0 to 32,767
2.0 & later only       Default: 0
```

If the SCROLLABLE attribute is set, this will place the bar specified at the left side of the graph, and move the the scroll bar accordingly.

```
FONT                   Type:    string
                       Values:  legal font-file name
                       Default: depends on system defaults
```

Gives name of font to be used in labels.

```
LABEL BACKGROUND       Type:    numeric
                       Values:  valid PEN number (0-255)
                       Default: depends on system defaults
```

Defines the color of the label background.

```
LABEL PEN              Type:    numeric
                       Values:  valid PEN number (0-255)
                       Default: depends on system defaults
```

Defines the colors of the labels.

```
LABEL WIDTH            Type:    numeric
                       Values:  any positive integer
                       Default: autosizing
```

Give width in pixels of label box near each bar, valid only for HORIZONTAL (RIGHT) widgets.

```
LIMITS BACKGROUND      Type:    numeric
                       Values:  valid PEN number (0-255)
                       Default: depends on system defaults
```

Defines the color of the limits backgrounds.

```
LIMITS PEN             Type:    numeric
                       Values:  valid PEN number (0-255)
                       Default: depends on system defaults
```

Defines the color of the limits-box characters.

```
LIMITS WIDTH           Type:    numeric
                       Values:  any positive integer
                       Default: autosizing
```

Give width in pixels of limits boxes near each bar, valid only for VERTICAL (UP) widgets.

```
LOGARITHMIC            Type:    numeric
                       Values:  0 (linear) or 1 (logarithmic)
                       Default: 0 (linear)
```

If 1, specifies log scale; if 0, linear.

```
MAXIMUM                Type:    numeric
                       Values:  any real number (positive if log)
                       Default: 100
```

Specifies the top limit of the bar.

```
MINIMUM                Type:    numeric
                       Values:  any real number (positive if log)
                       Default: 1
```

Specifies bottom limit of bar.

```
ORIENTATION              Type:    string
                         Values:  "UP" or "VERTICAL" or "RIGHT" or
                                  "HORIZONTAL"
                         Default: "VERTICAL"
```

Specifies rotation angle of the BARS widget.

```
SCROLLABLE               Type:    numeric
                         Values:  0 or 1
2.0 & later only         Default: 0 (NO)
```

If the attribute is set to TRUE, then the graph will have a scroll bar appear if the bars extend past what could normally be visible. The orientation of the scroll bar matches the orientation of the BAR graph. The scroll increment can be 1 pixel or 1 bar. Note that as the number of bars increases, the amount of time to scroll also increases. This is \Inot\i the same as scrolling the entire bar graph, since all the limits labels stay in place (see the SCROLLABLE attribute on the PANEL widget).

```
SCROLL INCREMENT         Type:    string
                         Values:  "BAR" or "PIXEL"
2.0 & later only         Default: "BAR"
```

If the SCROLLABLE attribute is set, the graph will be scrolled by one pixel at a time or one bar at a time, depending on the value set.

```
SHOW LABELS              Type:    numeric
                         Values:  0 or 1
                         Default: 1 (ON)
```

If 1, labels boxes will be shown for the BARS; if 0, they'll be invisible.

```
SHOW LIMITS              Type:    numeric
                         Values:  0 or 1
                         Default: 1 (ON)
```

If 1, limits boxes will be displayed; if 0, they'll be invisible.

```
SHOW VALUES              Type:    numeric
                         Values:  0 or 1
                         Default: 1 (ON)
```

If 1, values boxes will be displayed; if 0, they'll be invisible.

```
VALUE BACKGROUND       Type:    numeric
                       Values:  valid PEN number (0-255)
                       Default: depends on system defaults
```

Defines the background color of the value boxes.

```
VALUE PEN              Type:    numeric
                       Values:  valid PEN number (0-255)
                       Default: depends on system defaults
```

Defines the colors of the values in the value boxes.

```
VALUE WIDTH            Type:    numeric
                       Values:  any positive integer
                       Default: autosizing
```

Give width in pixels of values boxes near each bar, valid only for VERTICAL (UP) widgets; otherwise the width of the value boxes is related to the BAR WIDTH and BAR SEPARATION attributes (with VALUE WITDH ~= BAR WIDTH + BAR SEPARATION).

```
VALUES                 Type:    numeric or numeric string array
                       Values:  any real numbers
                       Default: 1-element array with 1.0
```

Values for all the bars can be sent to the bar chart at one time as a array parameter. If you want a particular numeric format in your values boxes, use a numeric string array with the desired format.

---

* You select an individual bar using the CURRENT BAR attribute; you can then tweak the following attributes:

```
ALARM RANGES           Type:    string
                       Values:  "" or "LOW" or "MIDDLE" or "HIGH"
                       Default: "" (null string)
```

Specifies ranges of values that will trip an alarm; you can use several in combination.

```
ALARM TYPE             Type:    string
                       Values:  "BEEP" or "EVENT"
```

	Default: "BEEP"

Specifies alarm type -- either a simple BEEP or an interrupting event.

BAR LABEL	Type: string
	Values: any string
	Default: number of CURRENT BAR

This gives the string below each bar that identifies it.

HIGH LIMIT	Type: numeric
	Values: any real number (positive for log)
	Default: MAXREAL

Specifies a high bound on bar values.

HIGH PEN	Type: numeric
	Values: legal PEN number (0-255)
	Default: depends on system defaults

Specifies pen color of bar above HIGH LIMIT (ignored on 1-plane mono monitors).

LIMITS LINE	Type: numeric
	Values: 0 or 1
2.0 & later only	Default: 0 (NO)

Draws lines pointing to where the low limit and high limits are for the given bar.

LOW LIMIT	Type: numeric
	Values: any real number
	Default: -MAXREAL

Specifies a low bound on bar values; values falling below this limit can trip an alarm.

LOW PEN	Type: numeric
	Values: legal PEN numbers (0-255)
	Default: depends on system defaults

Specifies pen color below LOW LIMIT (ignored on one-plane mono monitors).

MIDDLE PEN	Type: numeric

```
                       Values:  legal PEN numbers (0-255)
                       Default: depends on system defaults

Specifies pen color of bar between LOW LIMIT and HIGH LIMIT (ignored
on one-plane mono monitors).
_____________________________________________________________________

ORIGIN                 Type:    numeric
                       Values:  REAL
2.0 & later only       Default: -MAXREAL

Moves the origin of the bar to the specified location.
_____________________________________________________________________

ORIGIN PEN             Type:    numeric
                       Values:  legal PEN numbers (0-255)
2.0 & later only       Default: black

Sets color of origin line.
_____________________________________________________________________

VALUE                  Type:    numeric
                       Values:  any real number (positive if log)
                       Default: 1

The value you want the bar to display.
_____________________________________________________________________
```

The default attributes and events also apply (the "0" means "level 0 only"):

```
BACKGROUND          BORDER            CLOSE event (0)  HEIGHT
HELP FILE           HELP TOPIC        INSIDE HEIGHT    INSIDE WIDTH
MAXIMIZABLE (0)     MINIMIZABLE (0)   MOVABLE (0)      RESIZABLE (0)
RESTORE SCREEN (0)  STACKING ORDER    SYSTEM MENU (0)
SYSTEM MENU COUNT (0)                 SYSTEM MENU EVENT (0)
SYSTEM MENU event (0)                 TITLE (0)        USER DATA
VERSION            VISIBLE            WIDTH            X,Y
```

* There is only one unique event associated with the BARS widget:

```
ALARM  Occurs when VALUE exceeds one of the specified ALARM limits.
```

```
  ------------------------------------------------------------------------
```

[14.0] BPLUS (14): STRIPCHART & XYGRAPH Widgets

v2.4 / 01 feb 99 / greg goebel

* The STRIPCHART and XYGRAPH widgets are complicated widgets; the STRIPCHART allows you to display running traces of data values, while the XYGRAPH displays set traces defined by sets of X,Y pairs. Each has a large number of options and setting them up is confusing. This chapter explains how to configure these two widgets and gives example programs for each; since the attributes for the two widgets are similar, a single reference is given for them both at the end.

```
  ------------------------------------------------------------------------
```

```
  ------------------------------------------------------------------------
```

[14.1] STRIPCHART WIDGET

* The STRIPCHART widget allows you to trace up to 100 different streams of values on a scrolling graphics display. This display has a large number of format options; you can set ranges and tick styles, labels and numeric formats along both the X and Y axes, and different colors for the traces.

The operation of STRIPCHART widget is a little hard to explain. It is easiest to point out that it has three classes of attributes:

   * Those that apply to the entire widget ("global" attributes, if you like).
   * Those that apply to either the X or Y axis.
   * Those that apply to an individual trace.

-- and explain each in turn; then show how to actually get the STRIPCHART widget to display traces.

* If you simply create a STRIPCHART widget:

```
   10  CLEAR SCREEN
   20  ASSIGN @Stp TO WIDGET "STRIPCHART"
   30  CONTROL @Stp;SET ("X":50,"Y":50,"WIDTH":400,"HEIGHT":250)
   40  WAIT 10
   50  END
```

-- you get a widget that looks, *very* roughly, like this:

```
                    +--------------------------------------------------+---+
                    |                      STRIPCHART                 | X |
                    +----+-+------------------------------------------+---+
                    |    | |                                              |
                    |    | | <-- Y-axis tick mark box                     |
                    |    | |                                              |
  Y-axis number --> |    | |                                              |
   & label box      |    | |            TRACE DISPLAY AREA                |
                    |    | |                                              |
                    |    | |                                              |
                    |    | |                                              |
                    |    | |          X-axis tick mark box                |
                    |    +-+----------------------------------------------+
                    +----+-+----------------------------------------------+
                    |    |          X-axis number and label box           |
                    +----+------------------------------------------------+
```

There are relatively few attributes that apply to the entire widget. Some of them are obvious -- ORIENTATION (VERTICAL or HORIZONTAL) and TRACE BACKGROUND (sets the PEN color of the trace display area). Others aren't so obvious: for example, TRACE COUNT gives the number of traces that you want to display.

MINIMUM SCROLL tells how much the traces scrolling across the STRIPCHART display will scroll at one time, in terms of a percent of the X-axis; a small value (say one that defines one pixel of the width) will give a very smooth scroll, but usually that isn't useful; if your X-axis range is 10, and you perform updates in increments of 1, then you might as well set MINIMUM SCROLL to 10 (percent).

You can write each trace independently, but it is usually more practical to set up the STRIPCHART so that you can write a single X value (using the POINT LOCATION attribute) and then update all the traces at one time using an array (using the VALUES attribute). The SHARED X attribute is set to 1 to allow updating all the traces in parallel.

* Now we can use these global attributes to modify our example program to show three traces, scroll 1/10th of the display, and update all traces in parallel:

```
   CONTROL @Stp; SET ("TRACE COUNT":3, "MIMIMUM SCROLL":10, "SHARED X":1)
```

Setting these attributes makes absolutely no difference in the appearance of the display; for that we have to tweak the per-axis and per-trace values. We can do this using two other overall-widget attributes: CURRENT AXIS and CURRENT TRACE.

* The CURRENT AXIS attribute allows us to select one of the two STRIPCHART display axes, X or Y. Once we select an axis, we can define its ORIGIN and RANGE; if the axis will be LOGARITHMIC-scaled; an AXIS LABEL; various formats for TICK SPACING and appearance; as well as various NUMBER FORMATs

(AUTOmatic, FIXED, ENGINEERING, SCIENTIFIC, and time formats like CLOCK12).

If you don't want to mess around with scaling the axes, you can set them to AUTOSCALE, and the STRIPCHART will adjust them to the data it displays; if you don't want to mess around with the numbering and tick marks, you can turn them off by setting SHOW NUMBERING and SHOW TICKS to 0.

* Using these attributes, we will set the Y-axis ORIGIN to -4, its RANGE to 8, and give it an AXIS LABEL of "Y Stuff"; we will also set the X-axis ORIGIN to 0, its RANGE to 10, and give it an AXIS LABEL of "X Stuff":

```
   CONTROL @Stp; SET ("CURRENT AXIS":"Y")
   CONTROL @Stp; SET ("ORIGIN":-4, "RANGE":8, "AXIS LABEL":"Y Stuff")
   CONTROL @Stp; SET ("CURRENT AXIS":"X")
   CONTROL @Stp; SET ("ORIGIN":0, "RANGE":10, "AXIS LABEL":"X Stuff")
```

We leave the number and tick formats at AUTOmatic on both axes for convenience. Note that the x-axis ORIGIN is not fixed; once the STRIPCHART starts scrolling, the x-axis ORIGIN will scroll off the widget's display to the left. (The RANGE will remain the same, however.)

* With the axes set up, we can then set up the colors and labels for the individual traces using the CURRENT TRACE attribute to select a trace, then the TRACE PEN attribute to select a color and the TRACE LABEL attribute to select a label. You can also tell the STRIPGRAPH widget how many previous trace data points you want to store using the POINT CAPACITY attribute.

Using these attributes, we will set up our three traces and set a POINT CAPACITY of 1, since we don't really need to save the data in this example:

```
   CONTROL @Stp; SET ("CURRENT TRACE":1)
   CONTROL @Stp; SET ("TRACE PEN":1,"TRACE LABEL":"Trace 1") ! White trace.
   CONTROL @Stp; SET ("CURRENT TRACE":2)
   CONTROL @Stp; SET ("TRACE PEN":2,"TRACE LABEL":"Trace 2") ! Red trace.
   CONTROL @Stp; SET ("CURRENT TRACE":3)
   CONTROL @Stp; SET ("TRACE PEN":3,"TRACE LABEL":"Trace 3") ! Yellow trace.
   CONTROL @Stp; SET ("CURRENT TRACE":0, "POINT CAPACITY":0)
```

When you add these statements to the program, the labels "Trace 1", "Trace 2", and "Trace 3" appear in their respective TRACE PEN colors at the bottom of the widget, below the X-axis numbering box.

* Given these tools, we can easily display some arbitrary data:

```
   DIM Nval(1:3)
   ...
   DATA 1,-1,3
   READ Nval(*)
   FOR N=0 TO 1000
      CONTROL @Stp; SET ("POINT LOCATION":N,"VALUES":Nval(*))
      Nval(1)=-Nval(1)
      Nval(2)=-Nval(2)
      Nval(3)=-Nval(3)
   NEXT N
```

This traces out a pattern of triangle waves on the STRIPCHART widget. Note that if we had not set SHARED X to 1, we would have had to set CURRENT TRACE for each trace, set a POINT LOCATION for that trace, and then set a VALUE for that trace -- rather a nuisance in this case, though there are certainly other situations where it might be necessary.

* BPlus 2.0 added two features to the STRIPCHART and XYGRAPH: the SHOW GRID attribute, which allowed a grid to be displayed on the graphic surface, and the MARKER attribute, which allowed "markers" to be displayed on one or two traces that could be moved by the user with a mouse and interrogated for the trace values. There's not much to say about SHOW GRID -- more will be said about MARKER in the next section of this chapter.

Program Example: xstrip.rmb

[14.2] XYGRAPH WIDGET

* The XYGRAPH has almost all the same attributes as the STRIPCHART, but is intended to display sets of X,Y data points. In the STRIPCHART, you assign an X-coordinate POINT LOCATION along the axis and then load an array of Y-coordinate VALUES for that X-coordinate; as you keep incrementing the POINT LOCATION and writing an array with new Y-coordinate VALUES, the STRIPCHART display scrolls and all the traces are updated in parallel.

In the XYGRAPH, the display appears the same and is set up the same, but it doesn't scroll; the X axis is fixed. And instead of setting an X value and then updating the Y values of all traces, you simultaneously provide ALL the X and Y coordinates for a trace to plot that trace along its entire path:

```
   CONTROL @Xy; SET ("CURRENT TRACE":3,"X DATA":X3(*),"Y DATA":Y3(*))
   CONTROL @Xy; SET ("CURRENT TRACE":4,"X DATA":X4(*),"Y DATA":Y4(*))
   CONTROL @Xy; SET ("CURRENT TRACE":5,"X DATA":X5(*),"Y DATA":Y5(*))
```

If all your data sets have the same X coordinates, you can set SHARED X and only load the X data set once:

```
   CONTROL @Xy; SET ("SHARED X":0,"CURRENT TRACE":1,"X DATA":X(*))
   CONTROL @Xy; SET ("CURRENT TRACE":3,"Y DATA":Y3(*))
   CONTROL @Xy; SET ("CURRENT TRACE":4,"Y DATA":Y4(*))
   CONTROL @Xy; SET ("CURRENT TRACE":5,"Y DATA":Y5(*))
```

Note that in SHARED X mode on the XYGRAPH, the X DATA array is associated with trace 1, as shown above. Everything else -- all the format options, CURRENT AXIS, CURRENT TRACE, and so on -- works exactly the same as they do for the STRIPCHART. (In fact, they're different implementations of the same widget.)

* The MARKER attribute, as mentioned in the previous section, allows you to set the operation of one or two trace markers that the user can move along the specified trace. MARKER can be set to several modes: NONE (no markers), ONE marker, TWO markers, DELTA (difference between X & Y marker values), and RATIO (ratio of marker values). When markers are turned on, the appropriate marker values are displayed on the bottom of the XYGRAPH.

The traces to be marked are designated with the attributes MARKER1 TRACE and MARKER2 TRACE; the positions of the markers on the trace can be set or queried using the attributes MARKER1 X / MARKER1 Y, and MARKER2 X / MARKER2 Y. You can trap marker movements with the MARKER MOVED event.

Program Example: xxymark.rmb

[14.3] STRIPCHART & XYGRAPH ATTRIBUTE REFERENCE

* The STRIPCHART and XYGRAPH widgets have a large number of attributes, most of which are common to both; as a result, the attributes for both are discussed together. The attributes apply to both types of widgets unless otherwise specified. As noted, for these widgets there are three levels of attributes:

* Attributes applying to the entire widget.
* Attributes applying to an individual axis.
* Attributes applying to an individual trace.

The attributes that apply to the entire widget are as follows:

```
CURRENT AXIS           Type:    string
                       Values:  "X" or "Y"
                       Default: "X"
```

Specifies the axis (X or Y) to which the CURRENT AXIS attributes apply.

```
CURRENT TRACE          Type:    numeric
                       Values:  0 to TRACE COUNT
                       Default: 1
```

Specifies the trace to which CURRENT TRACE attributes apply. Note that 0 specifies ALL traces; if CURRENT TRACE is 0, then CONTROL statements will affect all the traces, but STATUS statements will generate an error.

```
MARKER                  Type:    string
                        Values:  "NONE" or "ONE" or "TWO" or "DELTA"
                                 or "RATIO"
2.0 & later only        Default: "NONE"
```

Specifies type and number of markers; note that "DELTA" gives two markers, with the difference between them displayed below the trace, and "RATIO" gives two markers, with the ratio between them displayed below the trace.

```
MARKER1 TRACE           Type:    numeric
                        Values:  any
2.0 & later only        Default: 1

MARKER2 TRACE           Type:    numeric
                        Values:  any
2.0 & later only        Default: 1
```

The trace on which the markers are displayed.

```
MARKER1 X               Type:    numeric
2.0 & later only        Values:  any

MARKER1 Y               Type:    numeric
2.0 & later only        Values:  any

MARKER2 X               Type:    numeric
2.0 & later only        Values:  any

MARKER2 Y               Type:    numeric
2.0 & later only        Values:  any
```

Sets or reads position of markers.

```
MINIMUM SCROLL          Type:    numeric
                        Values:  0 to 100
** STRIPCHART ONLY! **  Default: 5
```

The minimum amount the STRIPCHART is scrolled, in a fraction of its displayed width.

```
ORIENTATION             Type:    string
                        Values:  "UP" or "DOWN" or "LEFT" or "RIGHT"
                        Default: "UP"
```

Specifies the direction of the widget.

```
POINT LOCATION          Type:    numeric
                        Values:  any number
** STRIPCHART ONLY! ** Default: 0
```

Sets X value used for loading Y values with VALUE or VALUES attribute.

```
SHARED X                Type:    numeric
                        Values:  0 or 1
                        Default: 0 (not shared)
```

Conserves memory by using one X value for all Y values displayed.

```
SHOW GRID               Type:    string
                        Values:  "OFF" or "MAJOR" or "MINOR"
2.0 & later only        Default: "OFF"
```

Specify grid-drawing pattern.

```
TAB STOP                Type:    numeric
                        Values:  0 or 1
** STRIPCHART ONLY! ** Default: 1 (is TAB STOP)
```

Sets up widget as TAB STOP for key input (needed on STRIPCHART scrollbar).

```
TRACE BACKGROUND        Type:    numeric
                        Values:  valid PEN number (0-255)
                        Default: depends on system defaults
```

Defines background color of trace area and trace label area.

```
TRACE COUNT             Type:    numeric
                        Values:  1 to 100
                        Default: 4
```

Number of traces in the graph (affects memory usage).

```
VALUES                  Type:    numeric array
                        Values:  any numbers
** STRIPCHART ONLY! ** Default: zero array
```

Provides Y-values for latest update to the traces in the graph, after the X-value is set with POINT LOCATION. The array index corresponds to trace number.

---

* The attributes that apply to an individual axis (X or Y, as set by CURRENT AXIS) are as follows:

```
AUTOSCALE               Type:    numeric
                        Values:  0 or 1
                        Default: 0 (no AUTOSCALE)
```

Allows rescaling along Y-axis if a value exceeds the current range.

```
AUTOTICK                Type:    numeric
                        Values:  0 or 1
                        Default: 1 (AUTOTICK on)
```

Enables automatic setting of tick attributes.

```
AXIS LABEL              Type:    string
                        Values:  any string
                        Default: "" (null string)
```

String that gives name of axis.

```
DIGITS                  Type:    numeric
                        Values:  1 to 18
** STRIPCHART ONLY! **  Default: 6
```

Gives number of digits to be used when numbering the X axis.

```
LOGARITHMIC             Type:    numeric
                        Values:  0 (linear) or 1 (logarithmic)
                        Default: 0 (linear)
```

If 1, specifies log scale; if 0, linear.

```
LOG TICKS               Type:    numeric
                        Values:  1 to 5
                        Default: 4
```

Specifies one of five styles for ticks on log-scaled axes. The following table specifies the styles, with the numbers in parenthesis representing minor ticks (which will be invisible if MINOR TICKS is

0) and the others representing major ticks:

```
style 1:   1 (3) 10
style 2:   1 (2) (5) 10
style 3:   1 (2) (3) (4) (5) (6) (7) (8) (9) 10
style 4:   1 (1.5) (2) (2.5) 3 (4) (5) (6) (7) (8) (9) 10
style 5:   1 (1.5) 2 (3) (4) 5 (6) (7) (8) (9) 10
```

```
MINOR TICKS            Type:    numeric
                       Values:  0 to 100
                       Default: 1
```

The number of ticks in between each "major" interval: 0 disables minor ticks, 1 gives one tick, and so on. If LOGARITHMIC is set, the minor ticks will be determined by the LOG TICKS attribute.

```
NUMBER FORMAT          Type:    string
                       Values:  "AUTO" or "FIXED" or
                                "ENGINEERING" or "SCIENTIFIC" or
                                "MINUTES" or "HOURS" or "DAYS" or
                                "CLOCK12" or "CLOCK24"
                       Default: "AUTO"
```

The numbering format used to number the axes.

```
ORIGIN                 Type:    numeric
                       Values:  any number
                       Default: 0
```

The data coordinate of the axis specified by CURRENT AXIS. Must be greater than 0 if LOGARITHMIC is set.

```
RANGE                  Type:    numeric
                       Values:  any non-zero number
                       Default: 1
```

Range of axis (in decades if LOGARITHMIC is set). Cannot be 0.

```
SHOW NUMBERING         Type:    string
                       Values:  0 or 1
                       Default: 1 (numbering window on)
```

Turns on or off the numbering window for the current axis.

```
SHOW TICKS             Type:    string
                       Values:  0 or 1
                       Default: 1 (TICKS on)

Turns on or off the tick windows for current axis.
_____________________________________________________________________

TICK ORIGIN            Type:    numeric
                       Values:  any number
                       Default: 0

Sets starting point for major ticks.
_____________________________________________________________________

TICK SPACING           Type:    numeric
                       Values:  any non-zero number
                       Default: 1

Spacing for major ticks (not valid in LOGARITHMIC mode:  always 1).
Cannot be set to 0.
_____________________________________________________________________

USER SCROLL            Type:    numeric
                       Values:  0 or 1
** STRIPCHART ONLY! ** Default: 0 (no scroll)

Provides a scrollbar so the user can scroll the graph along the
X-axis.
_____________________________________________________________________
```

* The attributes that apply to an individual trace using the CURRENT TRACE attribute are as follows:

```
   _____________________________________________________________________

   POINT CAPACITY         Type:    numeric
                          Values:  0 to memory capacity
                          Default: 100

   Defines the amount of storage reserved for the trace; 16 bits is
   needed to store a single value.  SHARED X will ensure that only one X
   coordinate will be needed for all Y coordinates.
   _____________________________________________________________________

   POINT SYMBOL           Type:    numeric
                          Values:  0 to 15
                          Default on color monitor: 0 (none)
                          Default on mono monitor: 1 + CURRENT TRACE

   Gives type of symbol used for representing traces on mono displays;
   there are 15 different patterns, such as squares, diamonds, crosses,
   and various sorts of triangles.
   _____________________________________________________________________
```

```
TRACE LABEL             Type:    string
                        Values:  any valid string
                        Default: "" (null string)
```

Specifies the name of the trace, to be drawn in the box below the STRIPCHART.

```
TRACE PEN               Type:    numeric
                        Values:  valid PEN numbers (0-255)
                        Default: varies
```

Specifies pen color of trace.

```
TRACE VISIBLE           Type:    numeric
                        Values:  0 or 1
                        Default: 1 (visible)
```

Turns a trace and its label on or off (still can have data added).

```
VALID POINTS            Type:    numeric
                        Values:  0 to POINT CAPACITY
                        Default: 0
```

Specifies how many points have been added to trace.

```
X DATA                  Type:    numeric array
                        Values:  any numbers
... XYGRAPH ONLY! ...   Default: none
```

Allows X values to be loaded to a trace on the XYGRAPH widget.

```
Y DATA                  Type:    numeric array
                        Values:  any numbers
... XYGRAPH ONLY! ...   Default: none
```

Allows Y values to be loaded to a trace on the XYGRAPH widget.

```
VALUE                   Type:    numeric
                        Values:  any number
** STRIPCHART ONLY! ** Default: 0
```

Provides a single trace Y coordinate (linked to X provided by last POINT LOCATION attribute).

The default attributes and events also apply (the "0" means "level 0 only"):

BACKGROUND	BORDER	CLOSE event (0)	HEIGHT
HELP FILE	HELP TOPIC	INSIDE HEIGHT	INSIDE WIDTH
MAXIMIZABLE (0)	MINIMIZABLE (0)	MOVABLE (0)	RESIZABLE (0)
RESTORE SCREEN (0)	STACKING ORDER	SYSTEM MENU (0)	
SYSTEM MENU COUNT (0)		SYSTEM MENU EVENT (0)	
SYSTEM MENU event (0)		TITLE (0)	USER DATA
VERSION	VISIBLE	WIDTH	X,Y

There are two events (SCROLLED only works with the SCROLLBAR):

SCROLLED	Occurs when user scrolls X-axis with SCROLLBAR.
MARKER MOVED 2.0 & later	Indicates that the marker has been moved and the left mouse button has been released.

[15.0] BPLUS (15): Building Menu Systems With BASIC Plus

v2.4 / 01 feb 99 / greg goebel

* One of the most useful features of BPlus is its capability for building pull-down menu systems. However, this capability is also one of the trickier to understand ... not that there's anything inherently difficult about it, we've already built simple pulldown menus in earlier chapters; it just requires a bit of explanation.

[15.1] AN EXAMPLE

* As a starting point, let's consider an arbitrary pulldown menu that can be implemented by BPlus. This menu system resides in a PANEL that contains a PRINTER output widget; you select an element of a menu, and a string indicating the menu selection is dumped to the PRINTER widget.

This example menu system looks like this:

```
   +---+-----------------------------------------------------------+
   | = |                    MENU Widget Demo                       |
   +---+-----------------------------------------------------------+
   | PullDown_1     PullDown_2     PullDown_3                      |
   +---------------------------------------------------------------+
   |  +---------------------------------------------------------+  |
   |  |                                                         |  |
   |  |                                                         |  |
   |  |                                                         |  |
   |  |                                                         |  |
   |  |                                                         |  |
   |  |                                                         |  |
   |  |                                                         |  |
   |  |                                                         |  |
   |  |                                                         |  |
   |  +---------------------------------------------------------+  |
   +---------------------------------------------------------------+
```

Just for completeness, it includes a SYSTEM MENU (as shown by the toaster box in the upper left corner), if merely for contrast with the BPlus menu widgets on which this chapter is based:

```
+---+-----------------------------------------------------------+
| = |                    MENU Widget Demo                       |
+---+------+----------------------------------------------------+
| SysMenu1 | wn_1     PullDown_2     PullDown_3                 |
| SysMenu2 | -------------------------------------------------------+
| SysMenu3 |
+----------+
|
```

Since you should be familiar with the SYSTEM MENU by now, we won't say any more about it than we have to. So ... as for the menu widgets -- "PullDown_1" expands as follows:

```
+-----------------------------------------------
| PullDown_1     PullDown_2     PullDown_3
+-------------+------------------------------
| Button_1     |
| ------------ |
| Cascade_1  > |
| Cascade_2  > |
+-------------+
|
```

This menu includes a button ("Button_1"), a separator line, and two links to so-called "cascade" menus ("Cascade_1" and "Cascade_2") ... which I'll get to momentarily ... after I describe the rest of the top-level menu bar. "PullDown_2" expands into:

```
+------------------------------------------------
|  PullDown_1     PullDown_2     PullDown_3
+-------------+--------------+-----------------
|             | Button_1     |
|             | Button_2     |
|             +--------------+
|
```

-- and "PullDown_3" expands into:

```
+------------------------------------------------
|  PullDown_1     PullDown_2     PullDown_3
+-----------------------------+--------------+--
|                             | Button_1     |
|                             | Button_2     |
|                             | Button_3     |
|                             | ------------ |
|                             | Quit         |
|                             +--------------+
```

Pretty boring, huh? Let's go back to "PullDown_1". The "Cascade_1" menu entry expands as follows:

```
   +-------------------------------------------------
   |  PullDown_1     PullDown_2     PullDown_3
   +--------------+----------------------------------
   | Button_1     |
   | ------------ +--------+
   | Cascade_1  > | Item_1 |
   | Cascade_2  > | Item_1 |
   +--------------+ Item_1 |
   |              +--------+
```

Note that the "Cascade_1" entry is an "attachment point" for another, lower-level "cascade" menu. The "Cascade_2" menu entry similarly expands as follows:

```
   +-------------------------------------------------
   |  PullDown_1     PullDown_2     PullDown_3
   +--------------+----------------------------------
   | Button_1     |
   | ------------ |
   | Cascade_1  > +-------------+
   | Cascade_2  > | Toggle_1  * |
   +--------------+ Toggle_2  * |
   |              | Toggle_3  * |
   |              | Cascade_3 > |
   |              +-------------+
```

Note that when you click on the "Toggle" entries, the "*" comes and goes. Finally, the "Cascade_3" menu entry expands as follows:

```
   +-------------------------------------------------
   |  PullDown_1     PullDown_2     PullDown_3
   +--------------+----------------------------------
   | Button_1     |
   | ------------ |
   | Cascade_1  > +-------------+
   | Cascade_2  > | Toggle_1  * |
   +--------------+ Toggle_2  * |
   |              | Toggle_3  * +-------------+
   |              | Cascade_3 > | Toggle_1  * |
   |              +-------------+ Button_1    |
   |                            +-------------+
```

Okay, that's what the menu system looks like -- now how do you build the thing?

* First, some theory. BPlus builds a menu system like this with five

different types of widgets:

PULLDOWN MENU	Creates an "attachment point" for a pulldown menu.
MENU BUTTON	Creates an event when clicked.
MENU TOGGLE	Creates an event when clicked and also has a VALUE.
MENU SEPARATOR	Simply draws a line inside a menu.
CASCADE MENU	Creates an "attachment point" for a cascade menu.

The menu system described on the previous page (ignoring the SYSTEM MENU) is implemented with these widgets as follows:

```
   +---------------------------------------------------------------------+
   | PULLDOWN MENU     PULLDOWN MENU     PULLDOWN MENU                   |
   +-+-----------------+--------------------+----------------------------+
     |                 |                    |
     |                 v                    v
     |              +-----------------+  +-----------------+
     |              | MENU BUTTON     |  | MENU BUTTON     |
     |              | MENU BUTTON     |  | MENU BUTTON     |
     |              | MENU BUTTON     |  +-----------------+
     |              +-----------------+
     v
   +----------------+
   | MENU BUTTON    |
   | MENU SEPARATOR |    +----------------+
   | CASCADE MENU > +--->| MENU BUTTON    |
   | CASCADE MENU > +-+  | MENU BUTTON    |
   +----------------+ |  | MENU BUTTON    |
                      |  +----------------+
                      |
                      |  +----------------+
                      +->| MENU TOGGLE    |
                         | MENU TOGGLE    |
                         | MENU TOGGLE    |    +----------------+
                         | CASCADE MENU > +--->| MENU TOGGLE    |
                         +----------------+    | MENU BUTTON    |
                                               +----------------+
```

To create the menu system is simply to create all these widgets and add the appropriate labels for each widget; each widget has its own widget handle, allowing each to interact individually with the RMB program that creates it.

The next question is, of course: given this pile of widgets, how do you place them in that particular order?

The answer is: you don't! ... at least, not in detail. You simply define parent-child relationships between the widgets, and BPlus creates the menu system based on those relationships, as well as the sequence in which you create the widgets.

In the example menu system we are dealing with here, the widgets have the following relationships (where the children of a widget are indented from the parent):

PANEL		@P
PRINTER		@Prn
PULLDOWN MENU	"PullDown_1"	@Pd1
MENU BUTTON	"Button_1"	@B11
MENU SEPARATOR	---------	@S12
CASCADE MENU	"Cascade_1"	@C13
MENU BUTTON	"Item_1"	@B131
MENU BUTTON	"Item_2"	@B132
MENU BUTTON	"Item_3"	@B133
CASCADE MENU	"Cascade_2"	@C14
MENU TOGGLE	"Toggle_1"	@T141
MENU TOGGLE	"Toggle_2"	@T142
MENU TOGGLE	"Toggle_3"	@T143
CASCADE MENU	"Cascade_3"	@C144
MENU TOGGLE	"Toggle_1"	@T1441
MENU BUTTON	"Button_1"	@B1442
PULLDOWN MENU	"PullDown_2"	@Pd2
MENU BUTTON	"Button_1"	@B21
MENU BUTTON	"Button_2"	@B22
MENU BUTTON	"Button_3"	@B23
PULLDOWN MENU	"PullDown_3"	@Pd3
MENU BUTTON	"Button_1"	@B31
MENU BUTTON	"Button_2"	@B32

This table defines the widgets in the menu system, outlines their hierarchical relationships, and assigns the proper label and useful widget handle to each. (This is actually a handy way to lay out a menu system for implementation in an RMB program.)

As noted above, it is the hierarchical relationships of the widgets and the order in which they are created that defines the layout of the menu system; however, note that the requirement for ordering widgets to make sure they are in the proper sequence on the display only applies to each level of the menu system!

For example, consider the top-level PULLDOWN MENU widgets. You could create the widgets following the order in the table, and the three would be in the proper order; or you could create the three PULLDOWN MENU widgets and then go back and populate the lower-level widgets. The only requirement is that "PullDown_1" be created before "PullDown_2", and that "PullDown_2" be created before "PullDown_3".

But this discussion is straining the limits of what theory can explain. Now we move on to the actual implementation.

Program Example: xmenu.rmb

We start with some preliminaries: declaring variables and other housekeeping, laying out the user interface, and creating the backdrop parent PANEL widget:

```
10    ! *******************************************************************
20    !
30    ! Menu Demo Program for BPlus
40    !
50    ! *******************************************************************
60    !
70    INTEGER Black,White,Red,Yellow,Green,Blue,Cyan,Magenta
80    DATA 0,1,2,3,4,5,6,7
90    READ Black,White,Red,Yellow,Green,Blue,Cyan,Magenta
100   !
110   ! Define some variables:
140   !
150   !    S$:                          General-purpose string variable.
170   !    M$(*):                       System menu elements.
180   !    Cursor:                      Stores cursor value.
200   !    State:                       Gets state of MENU TOGGLE widgets.
240   !    N:                           General-purpose INTEGER variable.
270   !    D(*):                        Array to get hard-clip values.
300   !    Dw,Dh:                       Display dimensions.
320   !    X,Y:                         Location of main PANEL.
350   !    Panelwidth, Panelheight:     Dimensions for main PANEL.
370   !    Prx,Pry,Prwidth,Prheight:    Location & dimensions of main PANEL.
400   !    Ih,Iw:                       Interior dimensions of PANEL.
420   !
430   ! Note that the main panel is set up so it automatically scales to the
440   ! entire display, except for the DISP, INPUT, and softkeys lines.
450   !
460   DIM S$[80],M$(0:2)[80]
470   INTEGER Cursor,State,N,D(1:4)
480   INTEGER Dw,Dh,X,Y
490   INTEGER Panelwidth,Panelheight,Prx,Pry,Prwidth,Prheight,Iw,Ih
500   !
510   GESCAPE CRT,3;D(*)              ! Get display resolution.
520   Dw=D(3)-D(1)+1
530   Dh=D(4)-D(2)+1
540   Panelwidth=Dw*.75               ! Size PANEL.
550   Panelheight=Dh*.75
560   X=(Dw-Panelwidth)/2
570   Y=(Dh-Panelheight)/2
```

```
   580   !
   590   STATUS CRT,10;Cursor            ! Turn off cursor, etc.
   600   CONTROL CRT,10;0
   610   RUNLIGHT OFF
   620   KEY LABELS OFF
   630   !
   640   ! ******************************************************************
   650   !
   660   ! Create the PANEL widget.
   670   !
   680   CLEAR SCREEN
   690   ASSIGN @P TO WIDGET "PANEL";SET ("VISIBLE":0)
   700   CONTROL @P;SET ("RESIZABLE":1)
   710   CONTROL @P;SET ("X":X,"Y":Y,"WIDTH":Panelwidth,"HEIGHT":Panelheight)
   720   CONTROL @P;SET ("TITLE":"MENU Widget Demo")
   730   CONTROL @P;SET ("MAXIMIZABLE":0,"RESIZABLE":0)
   740   !
   750   ! ******************************************************************
```

Now that the background is in place, we can start building the menu system itself. First we create the topmost PULLDOWN MENU widgets as follows:

```
   750   ! ******************************************************************
   760   !
   770   ! Create the PULLDOWN MENU widgets, using the PANEL widget as parent.
   780   !
   790   S$="PullDown_1"
   800   ASSIGN @Pd1 TO WIDGET "PULLDOWN MENU";PARENT @P,SET ("LABEL":S$)
   810   !
   820   S$="PullDown_2"
   830   ASSIGN @Pd2 TO WIDGET "PULLDOWN MENU";PARENT @P,SET ("LABEL":S$)
   840   !
   850   S$="PullDown_3"
   860   ASSIGN @Pd3 TO WIDGET "PULLDOWN MENU";PARENT @P,SET ("LABEL":S$)
   870   !
   880   ! ******************************************************************
```

Note that I store the string "PullDown_n" in a string variable (S$); the only reason I do this is so I can fit things onto one line and eliminate some clutter.

The top-level menu automatically appears in a bar across the PANEL widget:

```
   +-----------------------------------------------------------------------+
   |  PullDown_1     PullDown_2     PullDown_3                             |
   +-----------------------------------------------------------------------+
   |                                                                       |
```

Note what this implies: telling BPlus to create a PULLDOWN MENU widget in a PANEL automatically creates a menu bar, and the PULLDOWN MENU entries are loaded into the bar in the order in which they are created.

Next, we set up the PRINTER widget; we had to wait until the menu bar was created to this because we couldn't get a valid INSIDE WIDTH and INSIDE HEIGHT before that point.

```
880   ! ****************************************************************
890   !
900   ! Now that you've set up the PULLDOWN MENU, get interior dimensions
910   ! of the PANEL and set up PRINTER widget accordingly.
920   !
930   STATUS @P;RETURN ("INSIDE WIDTH":Iw,"INSIDE HEIGHT":Ih)
940   Prx=Iw*.02
950   Pry=Ih*.02
960   Prwidth=Iw*.96
970   Prheight=Ih*.96
980   ASSIGN @Prn TO WIDGET "PRINTER";PARENT @P
990   CONTROL @Prn;SET ("X":Prx,"Y":Pry,"WIDTH":Prwidth,"HEIGHT":Prheight)
1000  !
1010  ! ****************************************************************
```

Next step: create the pulldown menus themselves. Just to knock them out of the way, let's create the simple pulldown menu for "PullDown_2" (and then "PullDown_3") first:

```
1010  ! ****************************************************************
1020  !
1030  ! Create menu for "PullDown_2" (using @Pd2 as PARENT).
1040  !
1050  S$="Button_1"
1060  ASSIGN @B21 TO WIDGET "MENU BUTTON";PARENT @Pd2,SET ("LABEL":S$)
1070  !
1080  S$="Button_2"
1090  ASSIGN @B22 TO WIDGET "MENU BUTTON";PARENT @Pd2,SET ("LABEL":S$)
1100  !
1110  ! ****************************************************************
```

This creates:

```
+----------------------------------------------------------------------+
|  PullDown_1     PullDown_2     PullDown_3                            |
+--------------+--------------+----------------------------------------+
|              | Button_1     |                                        |
|              | Button_2     |                                        |
|              +--------------+                                        |
|                                                                      |
```

Note that, as with the main menu, the items in the pulldown menu are automatically arranged in the order in which they are created. Now we create "PullDown_3":

```
1110  ! ****************************************************************
1120  !
1130  ! Create menu for "PullDown_3" (using @Pd3 as PARENT).
```

```
   1140  !
   1150  S$="Button_1"
   1160  ASSIGN @B31 TO WIDGET "MENU BUTTON";PARENT @Pd3,SET ("LABEL":S$)
   1170  !
   1180  S$="Button_2"
   1190  ASSIGN @B32 TO WIDGET "MENU BUTTON";PARENT @Pd3,SET ("LABEL":S$)
   1200  !
   1210  S$="Button_3"
   1220  ASSIGN @B33 TO WIDGET "MENU BUTTON";PARENT @Pd3,SET ("LABEL":S$)
   1230  !
   1240  ASSIGN @S34 TO WIDGET "MENU SEPARATOR";PARENT @Pd3
   1250  !
   1260  S$="Quit"
   1270  ASSIGN @Quit TO WIDGET "MENU BUTTON";PARENT @Pd3,SET ("LABEL":S$)
   1280  !
   1290  ! *****************************************************************
```

This creates:

```
   +------------------------------------------------------------------+
   |  PullDown_1     PullDown_2     PullDown_3                         |
   +---------------------------+-------------+------------------------+
   |                           | Button_1     |                        |
   |                           | Button_2     |                        |
   |                           | Button_3     |                        |
   |                           | ------------ |                        |
   |                           | Quit         |                        |
   |                           +-------------+                         |
```

Note that the MENU SEPARATOR simply draws a line through the PANEL. This may seem like a lot to go through to put a lousy line in the pulldown menu, but remember BPlus has to keep track of everything, and it can't do that unless you tell it how. Also note that the QUIT MENU BUTTON allows you to exit gracefully from this demo program.

That was simple; going back and creating the more complicated pulldown menu for "PullDown_1" is a little more challenging. The first entry is nothing but another button, same as above:

```
   1290  ! *****************************************************************
   1300  !
   1310  ! Create menu for "PullDown_1" (using @Pd1 as PARENT).
   1320  !
   1330  S$="Button_1"
   1340  ASSIGN @B11 TO WIDGET "MENU BUTTON";PARENT @Pd1,SET ("LABEL":S$)
   1350  !
```

A MENU SEPARATOR widget follows, to keep the MENU BUTTON cosmetically distinct from the CASCADE MENU widgets below it:

```
   1360  ! Add menu separator.
   1370  !
   1380  ASSIGN @S12 TO WIDGET "MENU SEPARATOR";PARENT @Pd1
   1390  !
```

Two CASCADE MENU widgets follow the MENU SEPARATOR:

```
   1400  ! Add CASCADE MENU widgets to "PullDown_1" (using @Pd1 as PARENT).
   1410  !
   1420  S$="Cascade_1"
   1430  ASSIGN @C13 TO WIDGET "CASCADE MENU";PARENT @Pd1,SET ("LABEL":S$)
   1440  !
   1450  S$="Cascade_2"
   1460  ASSIGN @C14 TO WIDGET "CASCADE MENU";PARENT @Pd1,SET ("LABEL":S$)
   1470  !
```

Voila! We have the pulldown menu:

```
   +-----------------------------------------------------------------------+
   |  PullDown_1     PullDown_2     PullDown_3                             |
   +--------------+--------------------------------------------------------+
   | Button_1     |                                                        |
   | ------------ |                                                        |
   | Cascade_1  > |                                                        |
   | Cascade_2  > |                                                        |
   +--------------+                                                        |
   |                                                                       |
```

This still leaves the "Cascade_1" and "Cascade_2" entries dangling; let's kill off "Cascade_1" first by assigning its MENU BUTTON entries:

```
   1480  ! Create menu for "Cascade_1" (using @C1 as PARENT).
   1490  !
   1500  S$="Item_1"
   1510  ASSIGN @B131 TO WIDGET "MENU BUTTON";PARENT @C13,SET ("LABEL":S$)
   1520  !
   1530  S$="Item_2"
   1540  ASSIGN @B132 TO WIDGET "MENU BUTTON";PARENT @C13,SET ("LABEL":S$)
   1550  !
   1560  S$="Item_3"
   1570  ASSIGN @B133 TO WIDGET "MENU BUTTON";PARENT @C13,SET ("LABEL":S$)
   1580  !
```

This results in:

```
   +-----------------------------------------------------------------------+
   |  PullDown_1     PullDown_2     PullDown_3                             |
   +--------------+--------------------------------------------------------+
   | Button_1     |                                                        |
   | ------------ +--------+                                               |
   | Cascade_1  > | Item_1 |                                               |
   | Cascade_2  > | Item_1 |                                               |
   +--------------+ Item_1 |                                               |
   |              +--------+                                               |
```

The "Cascade_2" menu is populated using the techniques shown so far, with the added wrinkle of its use of MENU TOGGLE widgets:

```
1590  ! Create menu for "Cascade_2" (using @C14 as PARENT).
1600  !
1610  S$="Toggle_1"
1620  ASSIGN @T141 TO WIDGET "MENU TOGGLE";PARENT @C14,SET ("LABEL":S$)
1630  !
1640  S$="Toggle_2"
1650  ASSIGN @T142 TO WIDGET "MENU TOGGLE";PARENT @C14,SET ("LABEL":S$)
1660  !
1670  S$="Toggle_3"
1680  ASSIGN @T143 TO WIDGET "MENU TOGGLE";PARENT @C14,SET ("LABEL":S$)
1690  !
1700  ! Add "Cascade_3" as entry in "Cascade_2".
1710  !
1720  S$="Cascade_3"
1730  ASSIGN @C144 TO WIDGET "CASCADE MENU";PARENT @C14,SET ("LABEL":S$)
1740  !
```

This gives:

```
+-----------------------------------------------------------------+
|  PullDown_1     PullDown_2     PullDown_3                       |
+--------------+--------------------------------------------------+
| Button_1     |                                                  |
| ------------ |                                                  |
| Cascade_1  > +-------------+                                    |
| Cascade_2  > | Toggle_1  * |                                    |
+--------------+ Toggle_2  * |                                    |
|              | Toggle_3  * |                                    |
|              | Cascade_3 > |                                    |
|              +-------------+                                    |
```

Almost done ... we populate the menu for "Cascade_3" as follows:

```
1750  ! Populate menu for "Cascade_3".
1760  !
1770  S$="Toggle_1"
1780  ASSIGN @T1441 TO WIDGET "MENU TOGGLE";PARENT @C144,SET ("LABEL":S$)
1790  !
1800  S$="Button_1"
1810  ASSIGN @B1442 TO WIDGET "MENU BUTTON";PARENT @C144,SET ("LABEL":S$)
1820  !
1830  ! *****************************************************************
```

This gives:

```
+------------------------------------------------------------------------+
|  PullDown_1     PullDown_2     PullDown_3                              |
+--------------+---------------------------------------------------------+
| Button_1     |                                                         |
| ------------ |                                                         |
| Cascade_1  > +-------------+                                           |
| Cascade_2  > | Toggle_1  * |                                           |
+--------------+ Toggle_2  * |                                           |
|              | Toggle_3  * +-------------+                             |
|              | Cascade_3 > | Toggle_1  * |                             |
|              +-------------+ Button_1    |                             |
|                            +-------------+                             |
```

This finishes off the pulldown menu system, so let me recapitulate: the pulldown menu is created one element at a time, with creation of each guided by BPlus under the direction of the sequence of the entries and their parents; each entry has its own widget handle.

Contrast this to the SYSTEM MENU, which we add almost as an afterthought to the main PANEL ... and then make the whole assembly visible:

```
1830  ! ******************************************************************
1840  !
1850  ! Create SYSTEM menu.
1860  !
1870  M$(0)="SysMenu1"
1880  M$(1)="SysMenu2"
1890  M$(2)="SysMenu3"
1900  CONTROL @P;SET ("SYSTEM MENU":M$(*))
1910  !
1920  CONTROL @P;SET ("VISIBLE":1)
1930  !
1940  ! ******************************************************************
```

The menu system is complete, but it's no good unless you can do something with it. So we set up events on each entry (or, more precisely, each MENU BUTTON or MENU TOGGLE entry, since the PULLDOWN MENU, CASCADE MENU, and MENU SEPARATOR widgets are largely invisible from the program once they are created) and execute the appropriate subroutine on those events:

```
1940  ! ******************************************************************
1950  !
1960  ! Set up menu events.
1970  !
1980  ON EVENT @B11,"ACTIVATED" GOSUB Button11
1990  ON EVENT @B21,"ACTIVATED" GOSUB Button21
2000  ON EVENT @B22,"ACTIVATED" GOSUB Button22
2010  ON EVENT @B31,"ACTIVATED" GOSUB Button31
2020  ON EVENT @B32,"ACTIVATED" GOSUB Button32
2030  ON EVENT @B33,"ACTIVATED" GOSUB Button33
2040  ON EVENT @B131,"ACTIVATED" GOSUB Button131
2050  ON EVENT @B132,"ACTIVATED" GOSUB Button132
```

```
2060  ON EVENT @B133,"ACTIVATED" GOSUB Button133
2070  ON EVENT @T141,"CHANGED" GOSUB Toggle141
2080  ON EVENT @T142,"CHANGED" GOSUB Toggle142
2090  ON EVENT @T143,"CHANGED" GOSUB Toggle143
2100  ON EVENT @T1441,"CHANGED" GOSUB Toggle1441
2110  ON EVENT @B1442,"ACTIVATED" GOSUB Button1442
2120  !
2130  ON EVENT @P,"SYSTEM MENU" GOSUB Sysmenu
2140  !
2150  ON EVENT @Quit,"ACTIVATED" GOTO Finis
2160  !
2170  LOOP
2180  END LOOP
2190  STOP
2200  !
2210  ! ********************* END OF MAIN PROGRAM ************************
```

Note how the SYSTEM MENU is the "odd one out" in this list of ON EVENT statement, both in terms of the name of the event and, more importantly, in that the event is set on the main PANEL, not on a menu widget.

The subroutines for these events are very simple:

```
2220  !
2230  ! The following routines are handlers for the various menu buttons.
2240  ! All they do is print to the printer widget what they are.
2250  !
2260 Button11: !
2270  S$="PullDown_1 / Button_1"
2280  CONTROL @Prn;SET ("APPEND TEXT":S$)
2290  RETURN
2300  !
2310 Button21: !
2320  S$="PullDown_2 / Button_1"
2330  CONTROL @Prn;SET ("APPEND TEXT":S$)
2340  RETURN
2350  !
2360 Button22: !
2370  S$="PullDown_2 / Button_2"
2380  CONTROL @Prn;SET ("APPEND TEXT":S$)
2390  RETURN
2400  !
2410 Button31: !
2420  S$="PullDown_3 / Button_1"
2430  CONTROL @Prn;SET ("APPEND TEXT":S$)
2440  RETURN
2450  !
2460 Button32: !
2470  S$="PullDown_3 / Button_2"
2480  CONTROL @Prn;SET ("APPEND TEXT":S$)
2490  RETURN
2500  !
2510 Button33: !
2520  S$="PullDown_3 / Button_3"
2530  CONTROL @Prn;SET ("APPEND TEXT":S$)
```

```
2540  RETURN
2550  !
2560 Button131: !
2570  S$="PullDown_1 / Cascade_1 / Item_1"
2580  CONTROL @Prn;SET ("APPEND TEXT":S$)
2590  RETURN
2600  !
2610 Button132: !
2620  S$="PullDown_1 / Cascade_1 / Item_2"
2630  CONTROL @Prn;SET ("APPEND TEXT":S$)
2640  RETURN
2650  !
2660 Button133: !
2670  S$="PullDown_1 / Cascade_1 / Item_3"
2680  CONTROL @Prn;SET ("APPEND TEXT":S$)
2690  RETURN
2700  !
2710 Toggle141: !
2720  S$="PullDown_1 / Cascade_2 / Toggle_1: "
2730  STATUS @T141;RETURN ("VALUE":State)
2740  CONTROL @Prn;SET ("APPEND TEXT":S$&VAL$(State))
2750  RETURN
2760  !
2770 Toggle142: !
2780  S$="PullDown_1 / Cascade_2 / Toggle_2: "
2790  STATUS @T142;RETURN ("VALUE":State)
2800  CONTROL @Prn;SET ("APPEND TEXT":S$&VAL$(State))
2810  RETURN
2820  !
2830 Toggle143: !
2840  S$="PullDown_1 / Cascade_2 / Toggle_3: "
2850  STATUS @T143;RETURN ("VALUE":State)
2860  CONTROL @Prn;SET ("APPEND TEXT":S$&VAL$(State))
2870  RETURN
2880  !
2890 Toggle1441: !
2900  S$="PullDown_1 / Cascade_2 / Cascade_3 / Toggle_1: "
2910  STATUS @T1441;RETURN ("VALUE":State)
2920  CONTROL @Prn;SET ("APPEND TEXT":S$&VAL$(State))
2930  RETURN
2940  !
2950 Button1442: !
2960  S$="PullDown_1 / Cascade_2 / Cascade_3 / Button_1"
2970  CONTROL @Prn;SET ("APPEND TEXT":S$)
2980  RETURN
2990  !
3000 Sysmenu: !
3010  STATUS @P;RETURN ("SYSTEM MENU EVENT":N)
3020  S$="SYSTEM MENU / SysMenu"&VAL$(N+1)
3030  CONTROL @Prn;SET ("APPEND TEXT":S$)
3040  RETURN
3050  !
3060  ! *****************************************************************
```

Note again how the SYSTEM MENU is the odd one; to determine the SYSTEM MENU entry, the program interrogates the PANEL for the index of the entry.

And, finally, this part of the program restores the RMB key labels and so on and exits the program:

```
3060  ! ******************************************************************
3070  !
3080  ! Go here when done.
3090  !
3100 Finis: !
3110  ASSIGN @P TO *              ! Kill off main panel.
3120  KEY LABELS ON
3130  RUNLIGHT ON
3140  CONTROL CRT,10;Cursor
3150  END
3160  !
3170  ! ********************** That's All, Folks! **********************
```

[15.2] ADVANCED MENU TECHNIQUES

* Pulldown menus are a particularly useful set of widgets, in that they allow you to implement a complicated system of program controls with a minimum amount of clutter on the user interface; a user interface build around pulldown menu widgets is also much easier to design, implement, and maintain, since you don't have to worry about such concerns as widget placement, size, or color.

There is a tendency when you first start working with menu systems to make them very static -- that is, to have the program bring up a menu system that remains unchanged through the rest of the program. But pulldown menu systems are much more flexible than that; for example:

* The menu widgets have an ATTRIBUTE called SENSITIVE that allows you to disable the widget; if SENSITIVE is cleared to 0, the menu widget label is printed in gray and a mouse click is ignored. You can use SENSITIVE to selectively disable menus and menu entries, depending on the mode of your program's operation.

  For example, if your program toggles between two different user displays via a set of menu entries, it is considerate to disable the one for the current display to reduce the user's confusion. It is also handy to use SENSITIVE to disable operation of routines that might disrupt some critical operation in the program.

* You can also use menu buttons to display data, not simply provide command inputs. This is very easy to do. Suppose you have a routine that changes a timeout value, and you call it with a MENU BUTTON; you could display the timeout value with the widget label as follows: Timeout = 30

Click on this MENU BUTTON, and it calls a routine to change the timeout; you change it to, say, 15 seconds, and the routine changes the label appropriately before it returns to the main routine: Timeout = 15

This is an extremely useful technique when the information controlled by the menu entry doesn't need to be visible except when it is to be changed; this trick allows the menu entry to do double duty as both an input and an display, reducing the number of widgets in your user interface, as well as interface clutter.

* While routines are normally called by events set on menu entries, it is also possible to have a routine simply poll a MENU TOGGLE and check its value in cases where handling events is inconvenient.

* Finally, it is also possible, though it is not necessarily easy, to dynamically rearrange menus within a program -- deleting menu entries that are not useful to a particular routine while adding new ones that are.

[15.3] PULLDOWN MENU WIDGET ATTRIBUTE REFERENCE

* The PULLDOWN MENU widget has the following attributes:

```
LABEL                  Type:    string
                       Values:  any string
                       Default: "" (null string)

Specifies a string to be used as a label on the menu bar.
```

```
SENSITIVE              Type:    numeric
                       Values:  0 or 1
                       Default: 1 (not sensitive)

If 0, the label is "grayed out" and unresponsive to user input.
```

```
USER DATA              Type:    string
                       Values:  any string
                       Default: "" (null string)

A string that the user can used for any purpose he or she wants.
```

There are no events associated with this widget.

[15.4] CASCADE MENU WIDGET ATTRIBUTE REFERENCE

* The CASCADE MENU has the following attributes:

```
LABEL                  Type:    string
                       Values:  any string
                       Default: "" (null string)
```

Specifies string to be used as label on the cascade menu.

```
SENSITIVE              Type:    numeric
                       Values:  0 or 1
                       Default: 1 (sensitive)
```

If 0, the label is visible but the cascade menu entry won't respond to user input.

```
USER DATA              Type:    string
                       Values:  any string
                       Default: "" (null string)
```

A string variable with no fixed purpose, simply provided as a convenience to the programmer.

The CASCADE MENU has no widget events.

[15.5] MENU BUTTON WIDGET ATTRIBUTE REFERENCE

* The MENU BUTTON widget has the following attributes:

```
LABEL                  Type:    string
                       Values:  any string
                       Default: "" (null string)
```

Specifies label to be used on the menu button.

```
SENSITIVE              Type:    numeric
                       Values:  0 or 1
                       Default: 1 (sensitive)
```

If 0, the menu button is visible but won't respond to mouse inputs.

USER DATA	Type:	string
	Values:	any string
	Default:	"" (null string)

A string variable with no fixed purpose, simply provided as a convenience to the programmer.

There is a single event associated with the MENU BUTTON:

ACTIVATED Occurs if mouse button is clicked on MENU BUTTON.

[15.6] MENU TOGGLE WIDGET ATTRIBUTE REFERENCE

* The MENU TOGGLE widget has the following attributes:

LABEL	Type:	string
	Values:	any string
	Default:	"" (null string)

Specifies label to be used in the menu toggle.

SENSITIVE	Type:	numeric
	Values:	0 or 1
	Default:	1 (sensitive)

If 0, the LABEL is visible but the MENU TOGGLE won't respond to mouse inputs.

VALUE	Type:	numeric
	Values:	0 or 1
	Default:	0

Value of MENU TOGGLE, switches between 0 and 1 with every mouse click.

USER DATA	Type:	string

	Values: any string Default: "" (null string)

A string variable with no fixed purpose, simply provided as a convenience to the programmer.

There is one event associated with the MENU TOGGLE:

CHANGED	Indicates mouse button was clicked on MENU TOGGLE.

[15.7] MENU SEPARATOR WIDGET ATTRIBUTE REFERENCE

* The MENU SEPARATOR widget has a single attribute:

USER DATA	Type: string Values: any string Default: ""

A string variable with no fixed purpose, simply provided as a convenience to the programmer.

There is no event associated with this widget.

[16.0] BPLUS (16): SYSTEM Widget

v2.4 / 01 feb 99 / greg goebel

* The chapter on BPlus applications showed how the Screen Builder could be used to generate a file that describes a user interface, and mentioned that the SYSTEM widget had to be used to integrate this file into a program. This chapter shows how to use the SYSTEM widget to allow a program to use a Screen Builder-generated user interface -- as well as be used on its own for building user interfaces programmatically, and (in particular) for building arrays of widgets.

[16.1] OVERVIEW / SCREEN BUILDER INTERACTION (*LOAD)

* The SYSTEM widget is different from any other widget in BPlus in that it has no attributes that can control its appearance; in fact, it can't even be made visible. The SYSTEM widget's intended purpose in life is to control other widgets.

As noted, the Screen Builder generates a file containing the names and specified attributes of the widgets defined with the tool. This file is read by the SYSTEM widget to connect it to an program, using the "*LOAD" attribute:

```
   ASSIGN @Sys TO WIDGET "SYSTEM";SET("*LOAD":<filename>)
```

This must be done every time you run the program -- the file must be present, or the program won't run. (Note that all SYSTEM widget attributes begin with a "*" to help distinguish the widget from all the others.)

Once the Screen Builder file has been loaded, then any of the widgets defined in the final can be controlled via the SYSTEM widget by selecting it with the *NAME attribute. For example, if a main PANEL is defined in the file with the name "Main", it can be accessed with:

```
   CONTROL @Sys;SET("*NAME":"Main", "WIDTH":10, "HEIGHT":20)
```

Once a widget has been selected with *NAME, all widget CONTROL and STATUS statements affect that widget and that widget only. Similarly, if "Main" has a child widget named "Meter2", that child can be accessed with:

```
   CONTROL @Sys;SET("*NAME":"Main/Meter2", "X":10, "Y":20)
```

This scheme implies that widgets -- at least those defined by the PANEL BUILDER utility -- can now be defined in string variables (or arrays):

```
   W$="Main/Bar1"
   STATUS @Sys;RETURN("*NAME":W$, "X":I, "Y":J)
```

Events can be trapped through the SYSTEM widget as well, though all you can do is specify the event type to be trapped -- not the particular event source:

```
   ON EVENT @Sys,"CLICKED" GOTO Handler
   ON EVENT @Sys,"DONE" GOTO Handler
```

You can determine the source by using the *QUEUED EVENT attribute:

```
   Ev$(1:2)[50]
   STATUS @Sys;RETURN("*QUEUED EVENT":Ev$(*)
   PRINT "Widget name:  ";Ev$(1);" Widget event: ";Ev$(2)
```

By default, the SYSTEM widget can only trap one event -- that is, if you don't read the events as they happen, you'll lose all but the last one. In most case this isn't a problem -- the user clicks on a button, the program traps the event and does something, and then the user does something else.

You can, however, arrange for the SYSTEM widget to queue up events, but you normally won't want to do that. (More on this later.) For now, let's consider a simple example of using the SYSTEM widget with the Screen Builder.

There's an old (and very dumb) computer game called "Bomb Squad", in which you pretend you have a bomb wired up with 10 wires. Disconnect the proper 4 wires, and the bomb is disarmed; cut the wrong 2 wires, and it blows up in your face. The other wires are dummies. Just to make it interesting, the bomb is ticking as you try to disarm it, and if you don't disarm it in time, it blows up.

Now this game is purely one of luck, but it does make a fun demo. We can build the game with ten TOGGLE BUTTONs (the wires), a CLOCK widget set to TIMER mode (the bomb timer, of course), and a PRINTER widget (for user feedback).

You can use the Screen Builder to lay out the user interface:

```
   +---+-------------------------------------------------+
   | = |               BOMB SQUAD Game                   |
   +---+---------+---------------------------------------+
   | [ ] Wire 1  |                                       |
   +-------------+         +------------------+          |
   | [ ] Wire 2  |         |                  |          |
   +-------------+         |     00:00:00     |          |
   | [ ] Wire 3  |         |                  |          |
   +-------------+         +------------------+          |
   | [ ] Wire 4  |                                       |
   +-------------+  +---------------------------------+  |
   | [ ] Wire 5  |  |                                 |  |
   +-------------+  |                                 |  |
   | [ ] Wire 6  |  |                                 |  |
   +-------------+  |                                 |  |
   | [ ] Wire 7  |  |                                 |  |
   +-------------+  |                                 |  |
   | [ ] Wire 8  |  |                                 |  |
   +-------------+  |                                 |  |
   | [ ] Wire 9  |  |                                 |  |
   +-------------+  +---------------------------------+  |
   | [ ] Wire 10 |                                       |
   +-------------+---------------------------------------+
```

You specify the elements as follows:

* The parent PANEL is named "Main", and has MAXIMIZABLE and RESIZABLE set turned off; a TITLE of "BOMB SQUAD Game"; and a SYSTEM MENU of "Quit".

* The 10 TOGGLE BUTTONs have names "T1" through "T10" (meaning they have full pathnames of "Main/T1" through "Main/T10"), and have labels "Wire 1" through "Wire 10".

* The CLOCK widget is named "Clock" ("Main/Clock") and has TYPE set to TIMER.

* The PRINTER widget is named "Prt" ("Main/Prt").

This user interface is stored in a description file named "XBSQUAD.SCR", which has the contents that look (cut down quite a bit) like this:

```
   13
   WIDGET 5:PANEL 4:Main -1 93 33 509 356 1
   11:MAXIMIZABLE 0
   9:RESIZABLE 0
   5:TITLE 15:BOMB SQUAD Game
   10:S_SYS_MENU 4:Quit
   4::END
   WIDGET 7:PRINTER 3:Prt 0 218 161 480 314 1
   4::END
    ...

   WIDGET 12:TOGGLEBUTTON 3:T10 0 95 326 192 356 1
   5:LABEL 7:Wire 10
   4::END
```

There's clearly a system to the contents of this file, but that's neither here nor there ... the file is listed just to give some idea of what it contains. The file is loaded into the program with:

```
   CONTROL @Sys;SET ("*LOAD":"XBSQUAD.SCR")
```

Program Example: xbsquad1.rmb | xbsquad.scr

[16.2] USING THE SYSTEM WIDGET ON ITS OWN (*CREATE)

* The SYSTEM widget can also be used without a Screen Builder file with the *CREATE attribute. For example, to create a PANEL named "Main", you would execute:

```
   CONTROL @Sys; SET("*NAME":"Main","*CREATE":"PANEL")
```

(Note that *NAME has to be specified before *CREATE can create the widget.) You can specify widget pathnames to define a parent-child relationship; for example, to create a child CLOCK widget named "Clock", you would execute:

```
   CONTROL @Sys; SET("*NAME":"Main/Clock","*CREATE":"CLOCK")
```

Program Example: xbsquad2.rmb

* Completely creating a user interface with the SYSTEM widget using *CREATE does not really make sense, however; if all your widgets are the different, it actually takes more work than building it in the normal fashion.

Where it does give an advantage, however, is when you have a set of widgets that are exactly the same -- say, a grid of PUSHBUTTON, STRING, or LABEL widgets; since you define the widgets using names given as strings in a string variable or string array avoids the normal BPlus restriction on having a widget handle for each widget. (This provides a rough equivalent to the "control arrays" available under Visual BASIC.)

You can set up your user interface in the conventional way, and then use the SYSTEM widget as a child of the main PANEL to define the widget grid.

Program Example: xsysch.rmb

You could similarly build a grid of LABELS or STRING widgets (possibly matched to a two-dimensional string array) to build a spreadsheet-like user interface.

* Note that in the case of this program, the SYSTEM widget can only trap one event at a time; any later events are lost until the first one is handled. In this case (and most others), that's fine, since it's almost impossible for someone to click on the PUSHBUTTONs faster than the program can handle them.

But in other cases you may need to guarantee that events are stored if they are not handled right away. You can set up event queueing by setting the *QUEUE EVENTS attribute to 1; the SYSTEM widget will then store all events that happen within it in the order they occur. You can determine the number of events in the queue with the *QUEUED EVENTS attribute:

```
   STATUS @Sys;RETURN ("*QUEUED EVENTS":Numevents)
```

-- and then read each event in sequence using the *QUEUED EVENT attribute as in the program above. Note that the SYSTEM widget has three attributes that have similar names -- *QUEUE EVENT, *QUEUE EVENTS, and *QUEUED EVENTS -- but have different functions -- to get the last event, to set event handling, and to get the number of events in the queue, respectively.

You may want to get all events of a certain type at the same time, however, so to allow you to sort them out, the SYSTEM widget has two attributes to allow you to specify which widget and event type will be read from the queue (leaving all the others remaining) -- *EVENT NAME FILTER and *EVENT WIDGET FILTER.

Program Example: xsysev.rmb

There are a number of other features to the SYSTEM widget; check the following attribute list for details.

[16.3] SYSTEM WIDGET ATTRIBUTE REFERENCE (2.0 & LATER ONLY)

* The SYSTEM widget supports the following attributes:

*CREATE	Type:	string (write-only)
	Values:	valid string

Specifies widget to be created; you specify a "pathname" to define a parent-child hierarchy. You must specify "*NAME" before you perform *CREATE.

```
*EVENT NAME FILTER     Type:    string
                       Values:  legal event names
                       Default: NULL (match all events)
```

Specifies the name of the event to retrieve from *QUEUED EVENTS or *FLUSH QUEUED EVENTS.

```
*EVENT WIDGET FILTER   Type:    string
                       Values:  legal assigned widget name
                       Default: NULL (match all widgets)
```

Specifies the *NAME of the widget for which to retrieve *QUEUED EVENTS or *FLUSH QUEUED EVENTS.

```
*FLUSH QUEUED EVENTS   Type:    numeric (read-only)
                       Values:  non-negative values
                       Default: 0
```

If *QUEUE EVENTS is set to 0 (event queueing is off), this attribute returns 0. If *QUEUE EVENTS is set to 1 (event queueing is on), performing a RETURN on this attribute will delete all queued events specified by the *EVENT WIDGET FILTER and *EVENT NAME FILTER, and return the number deleted.

```
*LOAD                  Type:    string (write-only)
                       Values:  valid file names
```

Specifies widget descriptor file to load.

```
*NAME                  Type:    string
                       Values:  valid string
```

Specifies the name of a widget contained in the SYSTEM widget; all following attributes will apply to that widget until an attribute that begins with "*" is specified, at which point "*NAME" must be respecified.

```
*QUEUE EVENTS          Type:    numeric
                       Values:  0 or 1
                       Default: 0 (off)
```

Specifies whether or not to maintain a queue of events generated by the widgets within the SYSTEM widget.

```
*QUEUED EVENT          Type:    2-element string array (read only)
                       Values:  legal *NAME of a widget (element 1)
                                legal event name (element 2)
```

If *QUEUE EVENTS is 0 (event queueing is off), this attribute returns the last event generated. If *QUEUE EVENTS is 1 (queueing is on), this attribute returns the oldest event specified by the *EVENT WIDGET FILTER and *EVENT NAME FILTER attributes and removes that event from the queue.

When queuing and none of the specified events are in the queue, NULL strings are returned. If a SYSTEM MENU event is returned, the widget name will actually be SYSTEM MENU:N, where "N" gives the index of the SYSTEM MENU string array.

```
*QUEUED EVENTS         Type:    numeric (read only)
                       Values:  nonnegative integer
```

Returns the number of events currently in the event queue.

```
*WIDGETS               Type:    numeric (read-only)
                       Values:  0 to number of widgets
```

Tells how many widgets are contained in the SYSTEM widget.

```
*WIDGET NAMES          Type:    string array (read-only)
                       Values:  legal string values
```

Returns the names of all widgets contained in the SYSTEM widget.

There are no unique events associated with the SYSTEM widget; as noted, it inherits the events of the widgets it contains.

--

[17.0] BPLUS (17): The HELPX Language

v2.4 / 01 feb 99 / greg goebel

* As noted in earlier chapters, BPlus version 2.0 introduced the ability to a allow users to build their own Help files using the HELPX language, and the ability to integrate online help into BPlus programs.

This chapter gives a quick overview of HELPX and "context-sensitive help". Note that HELPX works on RMB platforms only ... for HBW you must build the help files using standard Windows help tools, the use of which are well beyond the scope of this document. However, context-sensitive help applies to all forms of HP BASIC.

--

--

[17.1] INTRODUCTION TO HELPX

* To build a Help file, you must first create a Help source file using a text editor. This source file contains the text that make up the topics in the Help file and special embedded codes, or "tags", that tell the Help Compiler utility how to format and interpret the text. The formatted text in this file is a specialized subset of the "HP Tag" document formatting language used for creating many HP manuals.

The Help Compiler app, described in earlier chapters, is then used to create a compiled Help file -- which can then be loaded into the Help utility for use.

* There are two types of formatting tags in HELPX:

```
   <!--HELP_TITLE-->          An "online tag".
   <s1>                       A "text-formatting" tag.
```

Tags are used to define the beginning and end of topics (that is, the individual sections of a Help system); to format text within a topic; and to create "links" or "hyperlinks" that allow the user to jump from topic to topic.

The syntax of the tags is derived, as noted, from HP Tag (which is in turn a variant of the Standard Graphics Markup Language, or SGML ... note that the World Wide Web's Hypertext Markup Language, or HTML, is also derived from SGML and so the HELPX language has a vague similarity to HTML and conceptually much in common with it). In HP Tag, a comment -- text that is not printed -- is marked up like this:

```
   <!-- this is a comment -->
```

Since HELPX must support features that are not available in a printed document, such as hyperlinks and pop-up information panels, some mechanism had to be added to allow functions not supported in HP Tag -- so they were defined as specialized types of comments. This type of tag will be referred to as an "online tag" to distinguish it from the more conventional text formatting tags.

The other type of tag is the same as is used in HP Tag and are is mainly for formatting text. These are generally in two forms; a "long form":

```
   <tag_name>
   text to format
   <
```

-- and a "short form":

```
   <tag_name|text to format|
```

This type of tag is called a "text-formatting tag" to distinguish it from online tags.

* The basic elements of a source Help file can be explained by building a very simple Help file, piece by piece, and providing some explanations as we go. This Help file will describe the operation of a fictitious device, the ACME Laser Incineration System (LIS), and has absolutely nothing to do with RMB or anything (thankfully!) you can do with RMB -- it's just an example.

Program Example: xtest1.src

The (mandatory) first line in the Help source file gives the title:

```
   <!-- HELP_TITLE "ACME LIS: " -->
```

This gives the title of the Help system as "ACME LIS:"; this string will appear in the title bar of the Help utility, with the title of the current Help topic appended to it. The title doesn't have anything to do with the name of the Help file ... you can give the Help file any name you like.

* This title is followed by a number of topics. There are five in this example; the first is the table of contents:

```
   <!-- BEGIN TEXT acme.TOC
```

```
       TITLE "Contents"
   -->
   <s1>Table of Contents:

   <vex>
   <!-- LINK: acme.intro --> Introducing The LIS <!-- \LINK -->
   <!-- LINK: acme.use --> Using the LIS <!-- \LINK -->
   <!-- LINK: acme.warn --> Warnings & Cautions <!-- \LINK -->
   <\vex>

   <!-- END -->
```

The BEGIN TEXT tag defines this as a TEXT-type topic; you can define other types of topics, but we'll get to them later. The materials of this topic are bounded by the BEGIN TEXT and END tags.

Note that BEGIN TEXT uses two parameters: a topic name -- "acme.TOC" in this example -- and a title -- "Contents". The topic name is never seen by the user of the Help file; it's just a "handle" that identifies sections for setting up links and the like. The title string for the topic is appended to the title string for the Help system when that topic is displayed:

```
   ACME LIS:  Contents
```

Note also that the topic name "acme.TOC" has the .TOC extension; this defines it as the "table of contents" topic, which is the topic that appears when you click on the "Contents" button in the Help utility. There must be one, and only one, table of contents topic in a Help source file.

The "s1" tag defines a top-level section header:

```
   <s1>Table of Contents:
```

This simply prints the text in big bold letters. You have have as many "s1" section headers in one topic as you like. (There are also "s2", "s3", and "s4" section headers, but they all just use smaller bold print; they're not particularly useful.)

After that comes the body of the topic. In this case it is a set of hyperlinks to the three other topics that make up the body of this Help system. Each link has the form:

```
   <!-- LINK: acme.intro --> Introducing the LIS <!-- \LINK -->
```

The LINK tag uses as parameters the name of the topic being linked to -- "acme.intro" in this case -- and a string of highlighted text that will be printed in the Help utility -- "Introducing the LIS"; when you click on this string, you will jump to that topic.

Note the "vex" keyword; this forces the three hyperlinks to be displayed as they were written in the Help source file. Normally, the Help utility will perform a "fill" on topic text, adjusting paragraphs to fit into the Help utility display. However, if this was done with the three lines of text defined by the LINK tags, they would all end up on the same line -- so the "vex" tag is used to print the text as-is, using a fixed-spacing font.

There is also an "image" tag that prints text as-is, but uses any font that is available. The "vex" tag is preferred for making tables and the like.

* The second topic is the one identified as "acme.intro" in the table of contents:

```
<!-- BEGIN TEXT acme.intro
     TITLE "Introducing the LIS"
-->
<s1>Introducing the ACME LIS

The ACME Laser Incineration System (LIS) is a sophisticated materials
incineration system.  With the ACME LIS, you can:

<list order>
*Vaporize toxic materials down to atomic constituents.

*Destroy waste products too dangerous to handle by conventional methods.

*Cut the most rugged objects into pieces for transport and disposal.
<\list>

The ACME LIS is designed for many years of use.  Thank you for doing
business with ACME!

<!-- END -->
```

After reading the first section, you shouldn't have any trouble figuring this out -- except maybe for the "list order" tag, which generates a list of the items following it; the asterisks are replaced by a number. Note that you can specify a bulleted list (using "list") or a list with no numbering or bulleting ("list plain"), and that lists can be nested (most tag constructs can't).

* The next two topics are those identified by "acme.use" and "acme.warn" in the table of contents:

```
<!-- BEGIN TEXT acme.use
     TITLE "Using The LIS"
-->
<s1>Using the ACME LIS
```

```
   The ACME LIS has extreme power requirements; a direct hookup to a local
   nuclear reactor is highly recommended.  Safety considerations dictate
   siting of the device at least 10 kilometers from populated areas.  In
   operation, you should wear an asbestos suit and smoked glasses.

   After installation and hookup -- see the diagram in the shipping box for
   details -- its operation is simplicity itself.  Simply point the ACME
   LIS at the object you wish to atomize, and press the big red button that
   says "FIRE".  The object will be immediately and satisfyingly
   obliterated.

   <!-- END -->

   <!-- BEGIN TEXT acme.warn
        TITLE "Warnings And Cautions"
   -->
   <s1>Warnings & Cautions

   !!Misuse of the ACME LIS is potentially dangerous or fatal!!.  Care
   should be exercised at all times in its operation.

   ++ACME Corporation++ <!-- POP:owner.pop --> takes no responsibility for
   harm caused by inappropriate operation!

   <!-- END -->
```

Note the use of "!!"; this is a "shorthand" version of a text-formatting tag named "emph", and is used for defining emphasized text.

Also note the POP tag, which leads to the last topic in this Help source file.

* The POP tag identifies a "pop-up" in the help file; this is a panel that pops up when you click on a highlighted text field to provide certain information. In this case:

```
   ++ACME Corporation++ <!-- POP:owner.pop --> ...
```

-- the highlighted field is "ACME Corporation", and the pop-up will be defined by the pop-up topic "owner.pop", which is:

```
   <!-- BEGIN POP owner.pop -->
   ACME Corporation is a fully-owned subsidiary of Jones Industries INC.
   <!--END-->
```

Note that the "++" is a shorthand form of a text-formatting tag named "term"; it is only used in conjunction with POP. Note also that a pop topic can only contain unformatted text.

* All this done, you can then compile the Help source file with the Help Compiler, and then load it into the Help utility. You get the initial display:

```
+---+--------------------------------------------------------+---+---+
| = |                   ACME LIS: Contents                   | x | X |
+---+--------------------------------------------------------+---+---+
| File  SeeAlso  Programs                                            |
+----------+--------+------+-----------+------+---------------------+
| Contents | Search | Back | Copy Code | Quit |                     |
+----------+--------+------+-----------+------+---------------------+
|                                                                    |
| Table of Contents:                                                 |
|                                                                    |
|   Introducing the LIS                                              |
|   Using the LIS                                                    |
|   Warnings & Cautions                                              |
|                                                                    |
```

Select the first item in the table of contents and you get the display:

```
+---+--------------------------------------------------------+---+---+
| = |               ACME LIS: Introducing the LIS            | x | X |
+---+--------------------------------------------------------+---+---+
| File  SeeAlso  Programs                                            |
+----------+--------+------+-----------+------+---------------------+
| Contents | Search | Back | Copy Code | Quit |                     |
+----------+--------+------+-----------+------+---------------------+
|                                                                    |
| Introducing the ACME LIS                                           |
|                                                                    |
|    The ACME Laser Incineration System (LIS) is a sophisticated     |
|    materials incineration system. With the ACME LIS, you can:      |
|                                                                    |
|    1. Vaporize toxic materials down to atomic constituents.        |
|                                                                    |
|    2. Destroy waste products too dangerous to handle by            |
|       conventional methods.                                        |
|                                                                    |
|    3. Cut the most rugged objects into pieces for transport and    |
|       disposal.                                                    |
|                                                                    |
|    The ACME LIS is designed for many years of use.  Thank you for  |
|    doing business with ACME!                                       |
|                                                      line 16 of 16 |
+--------------------------------------------------------------------+
```

The second help topic appears as follows:

```
+---+--------------------------------------------------------+---+---+
   | = |                   ACME LIS: Using the LIS             | x | X |
   +---+---------------------------------------------------------+---+---+
   | File  SeeAlso  Programs                                             |
   +----------+--------+------+----------+------+---------------------+
   | Contents | Search | Back | Copy Code | Quit |                     |
   +----------+--------+------+----------+------+---------------------+
   |                                                                    |
   | Using the ACME LIS                                                 |
   |                                                                    |
   |   The ACME LIS has extreme power requirements; a direct hookup     |
   |   to a local nuclear reactor is highly recommended. Safety         |
   |   considerations dictate siting of the device at least 10          |
   |   kilometers from populated areas. In operation, you should        |
   |   wear an asbestos suit and smoked glasses.                        |
   |                                                                    |
   |   After installation and hookup -- see the diagram in the shipping |
   |   box for details -- its operation is simplicity itself. Simply    |
   |   point the ACME LIS at the object you wish to atomize, and press  |
   |   the big red button that says "FIRE".  The object will be         |
   |   immediately and satisfyingly obliterated.                        |
   |                                                                    |
```

-- and the third help topic shows up as:

```
   +---+---------------------------------------------------------+---+---+
   | = |               ACME LIS: Warnings and Cautions           | x | X |
   +---+---------------------------------------------------------+---+---+
   | File  SeeAlso  Programs                                             |
   +----------+--------+------+----------+------+---------------------+
   | Contents | Search | Back | Copy Code | Quit |                     |
   +----------+--------+------+----------+------+---------------------+
   |                                                                    |
   | Warnings & Cautions                                                |
   |                                                                    |
   |    *Misuse of the ACME LIS is potentially dangerous or fatal*.     |
   |    Care should be exercised at all times in its operation.         |
   |                                                                    |
   |    *ACME Corporation* takes no responsibility for harm caused by   |
   |    inappropriate operation!                                        |
   |                                                                    |
```

Click on the highlighted "ACME Corporation" string and you get the pop-up panel over it:

```
   +---------------------------------------------------------------------+
   | ACME Corporation is a fully-owned subsidiary of Jones Industries   |
   | INC.                                                                |
   +---------------------------------------------------------------------+
```

[17.2] RMB PROGRAMS & HELPX

* In reality, of course, you won't be using HELPX to write Help systems for laser incinerators; you'll be using it to document an RMB program. HELPX, as a result, contains several features for embedding program examples in the Help file that can be sent to the RMB editor. The "ex", PROGRAM, and EXAMPLE tags are used to support these features.

* The "ex" tag may be used anywhere within the formatted text block of a TEXT topic; it marks one or more lines of example RMB program statements that are inserted into the RMB editor when you click on an "e>" that marks the example.

If the example is only one line long, it should not include a line number. If the example is more than one line long, it may include line numbers; the line numbering will be adjusted when the code is inserted into the editor. For example:

```
   <ex|Degc=(Degf-32)*(5/9)|
```

-- provides a single-line example, while:

```
   <ex>
   100  Kilos=Pounds*2.2046
   110  Km=Miles*1.609
   120  Watts=Hp*746
   <\ex>
```

-- provides a multi-line example.

* The BEGIN PROGRAM tag is much like the BEGIN TEXT tag in that it defines a Help system topic; in the case of the BEGIN PROGRAM tag, this topic is a complete RMB program. When the PROGRAM topic is displayed, the "Copy Code" button on the Help utility can then be used to load the program into the RMB editor. For example:

```
   <!-- BEGIN PROGRAM printer.test
        TITLE "Print Data"
   -->
   RMB program listing
   <!-- END -->
```

You could also store the program in a file and then use the INCLUDE online tag to merge the file:

```
   <!-- BEGIN PROGRAM topic.name
        TITLE "Program Title"
   -->
   <!-- INCLUDE: myfile.rmb -->
   <!-- END -->
```

* The EXAMPLE tag allows you to create selections in the "Programs" pulldown menu of the Help utility; it is not so much a tag in itself as a secondary tag, used with BEGIN TEXT, and defined as follows:

```
   <!-- BEGIN TEXT topic.name
        TITLE      "The Topic Title"
        SEE_ALSO   "Related Features" rmb.other
        EXAMPLE    "RMB Example 1"    rmb.ex1
        EXAMPLE    "RMB Example 2"    rmb.ex2
   -->
```

In this case, the SEE_ALSO secondary tag is also used; it defines a cross-reference to a topic or topics in the "See Also" menu of the Help Utility.

* The following example Help file illustrates RMB code handling through a Help file that provides vector-algebra routines.

Note that there are two tags in this Help source file that have not yet been introduced: "rsect" and "rsub" (meaning "reference section" and "reference sub"). Nothing to them, really; they're just another flavor of section header, much like "s1" and "s2". You could replace "rsect" with "s1" and "rsub" with "s2" ... and not really notice much difference.

```
   <!-- HELP_TITLE "Vector Lib:  " -->

   <!-- BEGIN TEXT vectors.TOC
        TITLE "Vector Algrebra Routines Library"
   -->
   <s1>Function Dictionary

   <vex>
   <!-- LINK:vectors.over --> Vector Library Overview <!-- \LINK -->
   <!-- LINK:vectors.addsub --> Vector Add & Subtract <!-- \LINK -->
   <!-- LINK:vectors.norm --> Vector Length (Norm) <!-- \LINK -->
   <!-- LINK:vectors.mult --> Vector Dot & Cross Products <!-- \LINK -->
   <!-- LINK:vectors.angle --> Angle Between Vectors <!-- \LINK -->
   <!-- LINK:vectors.prog --> Example Program<!-- \LINK -->
   <\vex>

   <!-- END -->


   <!-- BEGIN TEXT vectors.over
        TITLE "Vector Library Overview"
   -->
```

```
<s1> Vectors & Vector Operations

In most everyday math operations -- in shopping or home finance or other
run-of-the-mill activities -- we do calculations on one number at a
time, adding them up, subtracting them, and so on, to get a single value
as a result -- like the money we owe, or how much something will cost.

In engineering work, it is often useful to not merely indicate a simple
quantity, but also a direction -- like the speed and direction of an
aircraft, for example.

If you were to put an aircraft in an XYZ coordinate system, its speed
and direction could be defined by three numbers, giving the component
of the speed of the aircraft in each of the X-Y-Z directions.  This
triplet of numbers is known as a "vector", while a simple quantity is
in contrast referred to as a "scalar".

There are a number of operations defined on vectors:

<list>
*Vector addition & subtraction.
*Vector magnitude or "norm".
*Vector "dot product" and "cross product".
*Deriving the angle between two vectors.
<\list>

The SUBs and functions contained in this Help file provide these
operations; for details on the operations, just consult the appropriate
entries.

Note that these routines expect as arguments REAL arrays with indexes
from 1 to 3; for example:

<ex|DIM A(1:3),B(1:3),C(1:3)|

Note also that since this Help file contains SUBs, you must take care
when attempting to insert them into RMB code; RMB prefers to have SUBs
inserted at the very end of the program, and will complain if you try
to insert them someplace else.

<!-- END -->


<!-- BEGIN TEXT vectors.addsub
     TITLE "Vector Add and Subtract"
-->
<rsect> Vector Add & Subtract Routines

Vector addition and subtraction are extremely simple operations; all you
have to do is add or subtract the three components of the vectors.  Of
course, the result is another vector.
```

```
<rsub> Vector Addition SUB:

<ex>
SUB Vadd(A(*),B(*),C(*))
  C(1)=A(1)+B(1)
  C(2)=A(2)+B(2)
  C(3)=A(3)+B(3)
SUBEND
<\ex>

<rsub> Vector Subtraction SUB:

<ex>
SUB Vsub(A(*),B(*),C(*))
  C(1)=A(1)-B(1)
  C(2)=A(2)-B(2)
  C(3)=A(3)-B(3)
SUBEND
<\ex>

<!-- END -->


<!-- BEGIN TEXT vectors.norm
     TITLE "Vector Norm"
-->
<rsect> Vector Norm (Magnitude)
```

Sometimes when you have a vector you may be fickle and decide you simply want a scalar after all. It's easy to get this magnitude, or "norm" in vector talk, by an extension of the Pythagorean theorem: just take the square root of the sum of the squares of all the components.

Of course, the norm is a scalar.

```
<rsub> Vector Norm Function:

<ex>
DEF FNNorm(A(*))
  RETURN SQR(A(1)^2+A(2)^2+A(3)^2)
FNEND
<\ex>

<!-- END -->


<!-- BEGIN TEXT vectors.mult
     TITLE "Vector Dot and Cross Products"
-->
<rsect> Vector Dot & Cross Products
```

There are two types of vector multiplcation operations: vector dot product and vector cross product.

The vector dot product is the sum of multiplying the three components of two vectors together; it of course returns a scalar. It is equivalent to the operation:

```
   Norm(Vector A) * Norm(Vector B) * COS(Angle between A and B)
```

It is often used to determine the component of one vector along another.

The vector cross product is a more complicated computation that involves cross-multiplying the elements of two vectors to yield another vector. (See the Vcross SUB below for details.) This vector has the magnitude:

```
   Norm(Vector A) * Norm(Vector B) * SIN(Angle between A and B)
```

-- and is directed at a right angle to the plane defined by vectors A and B (with the orientation determined by the so-called "right-hand screw rule").

The cross product has application in the analysis of fields and flows that are too complicated to explain here.

<rsub> Vector Dot Product Function:

```
<ex>
DEF FNDot(A(*),B(*))
  RETURN A(1)*B(1)+A(2)*B(2)+A(3)*B(3)
FNEND
<\ex>
```

<rsub> Vector Cross Product SUB:

```
<ex>
SUB Vcross(A(*),B(*),C(*))
  C(1)=A(2)*B(3)-A(3)*B(2)
  C(2)=A(3)*B(1)-A(1)*B(3)
  C(3)=A(1)*B(2)-A(2)*B(1)
SUBEND
<\ex>
```

<!-- END -->

```
<!-- BEGIN TEXT vectors.angle
     TITLE "Angle Between Two Vectors"
-->
```

<rsect> Angle Between Two Vectors

Since the dot product is given by:

```
   Norm(Vector A) * Norm(Vector B) * COS(Angle between A and B)
```

-- you can then figure out the angle between two vectors with:

```
   ACS(Dot(Vector A, Vector B)/(Norm(Vector A)*Norm(Vector B)))
```

-- where ACS is the inverse cosine. Of course, this returns a scalar.

<rsub> Vector Angle Function:

```
<ex>
DEF FNAngle(A(*),B(*))
  RETURN ACS(FNDot(A(*),B(*))/(FNNorm(A(*))*FNNorm(B(*))))
FNEND
<\ex>
```

<!-- END -->

```
<!-- BEGIN PROGRAM vectors.prog
     TITLE "Vector Library Example Program"
-->
10    ! ************************************************************
20    !
30    ! Vector-Algebra Functions In RMB
40    !
50    ! ************************************************************
60    !
70    CLEAR SCREEN
80    DEG
90  Vi: IMAGE K,X,3(4D.3D,X)         ! IMAGE for printing vectors.
100 Si: IMAGE K,X,(4D.3D)            ! IMAGE for printing scalars.
110   !
120   REAL A(1:3),B(1:3),C(1:3)      ! Sample vectors.
130   DATA 1,2,3,-6,-4,-5
140   READ A(*),B(*)
150   !
160   PRINT USING Vi;"Sample Vector 1: ";A(*)
170   PRINT USING Vi;"Sample Vector 2: ";B(*)
180   PRINT
190   !
200   CALL Vadd(A(*),B(*),C(*))
210   PRINT USING Vi;"Vector Add:      ";C(*)
220   !
230   CALL Vsub(A(*),B(*),C(*))
240   PRINT USING Vi;"Vector Subtract: ";C(*)
250   !
260   CALL Vcross(A(*),B(*),C(*))
270   PRINT USING Vi;"Cross Product:   ";C(*)
280   !
290   PRINT
300   PRINT USING Si;"Vector 1 Norm:   ";FNNorm(A(*))
310   PRINT USING Si;"Vector 2 Norm:   ";FNNorm(B(*))
```

```
320   PRINT USING Si;"Dot Product:     ";FNDot(A(*),B(*))
330   PRINT USING Si;"Angle:           ";FNAngle(A(*),B(*))
340   !
350   END
360   !
370   ! ************************************************************
380   !
390   SUB Vadd(A(*),B(*),C(*))
400     C(1)=A(1)+B(1)
410     C(2)=A(2)+B(2)
420     C(3)=A(3)+B(3)
430   SUBEND
440   !
450   DEF FNAngle(A(*),B(*))
460     RETURN ACS(FNDot(A(*),B(*))/(FNNorm(A(*))*FNNorm(B(*))))
470   FNEND
480   !
490   SUB Vcross(A(*),B(*),C(*))
500     C(1)=A(2)*B(3)-A(3)*B(2)
510     C(2)=A(3)*B(1)-A(1)*B(3)
520     C(3)=A(1)*B(2)-A(2)*B(1)
530   SUBEND
540   !
550   DEF FNDot(A(*),B(*))
560     RETURN A(1)*B(1)+A(2)*B(2)+A(3)*B(3)
570   FNEND
580   !
590   DEF FNNorm(A(*))
600     RETURN SQR(A(1)^2+A(2)^2+A(3)^2)
610   FNEND
620   !
630   SUB Vsub(A(*),B(*),C(*))
640     C(1)=A(1)-B(1)
650     C(2)=A(2)-B(2)
660     C(3)=A(3)-B(3)
670   SUBEND
680   !
690   ! ********************* That's All, Folks! *******************
<!-- END -->
```

Program Example: xtest2.src

[17.3] HELP INDEXING

* If you run the previous Help files, you'll notice something odd: the "Search" button on the Help utility is grayed out.

That's because "Search" requires you to build an index before it can be used. In the two examples shown previously, they've been so simple that defining an index would be pointless, but an index would be virtually mandatory for a more complex Help file.

There are two tags used to define an index: "idx" and (if you feel the need) SEARCH_TITLE. The "idx" tag defines an index entry; it can either be just a simple index:

```
   <idx|vector operations|
```

-- or a two-level index, with a "primary" and "secondary" index item:

```
   <idx|vector multiplication:dot product|
```

These indexes are placed in the individual topics to identify them to the "Search" system. When you identify a desired item in the "Search" system, it lists the target topics by the topic TITLE, or the optional SEARCH_TITLE -- which simply gives an alternate title for the "Search" system only -- or by the "secondary" index name.

We could add indexing to the previous example as follows:

```
   <!-- BEGIN TEXT vectors.over
        TITLE "Vector Library Overview"
        SEARCH_TITLE "Vector Operations / Scalars"
   -->

   <idx|vector operations|
   <idx|scalars|
   ...

   <!-- BEGIN TEXT vectors.addsub
   ...
   <idx|vector addition|
   <idx|vector subtraction|
   ...

   <!-- BEGIN TEXT vectors.norm
   ...
   <idx|vector norm|
   <idx|vector magnitude|
   ...

   <!-- BEGIN TEXT vectors.mult
   ...
   <idx|vector dot product|
   <idx|vector cross product|
   <idx|vector multiplication:dot product|
   <idx|vector multiplication:cross product|
   ...
   <!-- BEGIN TEXT vectors.angle
   ...
   <idx|angle between vectors|
   ...
```

Program Example: xtest3.src

When you compile this modified Help source file and perform a "Search" for, say, "vector multiplication", you get:

```
   +---+--------------------------------------------------------+---+---+
   | = |                     HP BASIC: SEARCH                   | x | X |
   +---+--------------------------------------------------------+---+---+
   | Search for:                                                        |
   | +---------------------------------------+                          |
   | | vector multiplication                 |  +- Extended Search -+ |
   | +---------------------------------------+  |    +----------+    | |
   | Index Entries                              |    |  Start   |    | |
   | +------------------------------------+-+  |    +----------+    | |
   | | vector cross product              |^|  |    |  Stop    |    | |
   | | vector dot product                | |  |    +----------+    | |
   | | vector magnitude                  | |  +--------------------+ |
   | |[vector multiplication            ]| |       +----------+        |
   | | vector norm                       |v|       |  Cancel |        |
   | +------------------------------------+-+       +----------+        |
   | ================================================================  |
   | Topics Found: 1                                                    |
   | +---------------------------------------+       +----------+      |
   | |[cross product                       ]|       |  Go To   |      |
   | | dot product                           |       +----------+      |
   | |                                       |                          |
   | +---------------------------------------+                          |
   +--------------------------------------------------------------------+
```

Perform a "Go To" and you will end up the appropriate topic.

[17.4] CONTEXT-SENSITIVE HELP

* As noted in previous chapters, BPlus 2.0 introduced the neat feature of "context-sensitive Help" -- which allows you to link a Help file and Help topic to a particular widget in a user interface; when you click on the widget with your secondary mouse button, the Help topic is automatically displayed. The Help file and Help topic for that widget are defined by two attributes named, not surprisingly, HELP FILE and HELP TOPIC.

* As an example, let's build a simple user interface and a Help file to go along with it. This user interface consists only of a PANEL and three PUSHBUTTONs and appears as:

```
+---+------------------------------------------------------------------+
| = |                         Weapons Systems                          |
+---+------------------------------------------------------------------+
|  +-----------------+ +----------------------+ +----------------+  |
|  |   Fire Phasers   | | Fire Photon Torpedoes | |   Stand Down   |  |
|  +-----------------+ +----------------------+ +----------------+  |
+----------------------------------------------------------------------+
```

We'll build a Help file that has a separate Help topic for each of the three PUSHBUTTONs, and a table-of-contents (TOC) Help topic for the main PANEL. The Help source file follows below:

```
  <!-- HELP_TITLE "Weapons Systems:  " -->

  <!-- BEGIN TEXT weapons.TOC
       TITLE "Command System Functionality"
  -->
  <s1>Command System Functionality

  This control panel directs the operation of the primary weapons systems.
  It provides command direction of phaser banks and photon torpedoes and
  allows disarming the weapon systems.

  <rsect> System Functions:

  <vex>
  <!-- LINK:weapons.phasers -->   Phasers <!-- \LINK -->
  <!-- LINK:weapons.torp -->   Photon Torpedoes <!-- \LINK -->
  <!-- LINK:weapons.off -->   Stand Down <!-- \LINK -->
  <\vex>

  <!-- END -->

  <!-- BEGIN TEXT weapons.phasers
       TITLE "Phasers"
  -->
  <s1> Phasers

  Phasers are short-range directed energy weapons.  If you activate this
  control, phaser fire is directed at the adversary currently in weapons
  lock.

  <!-- END -->

  <!-- BEGIN TEXT weapons.torp
       TITLE "Photon Torpedoes"
  -->
  <s1> Photon Torpedoes

  Photon torpedoes are long-range warp-powered weapons carrying warheads of
  high explosive yield.  If you activate this control, torpedoes will be
  fired from all loaded tubes at the adversary currently in weapons lock.

  <!-- END -->


  <!-- BEGIN TEXT weapons.off
       TITLE "Stand Down"
  -->
  <s1> Stand Down Weapons Systems

  If you activate this control, energy is drained from the phaser banks and
  returned to operational systems; photon torpedoes are disarmed and put in
  standby.  However, targeting systems still remain operational unless
  specifically disabled.

  <!-- END -->
```

Program Example: xtest4.src

The matching program is as follows:

```
   10    ! ******************************************************************
   20    !
   30    ! Context-Sensitive Help Test Program For BPP
   40    !
   110   ! ******************************************************************
   120   !
   150   DIM S$[256]                              ! GP string.
   160   INTEGER Nlines,D(1:4)                    ! Gets display size.
   170   INTEGER Dw,Dh,Px,Py,Pw,Ph,Iw,Ih,Gx,Gy    ! Main panel parms, etc.
   180   INTEGER Btnw,Btnh,Y,X1,X2,X3             ! Button parms.
   190   !
   200   ! Get display size.
   210   !
   220   STATUS CRT,13;Nlines
   230   GESCAPE CRT,3;D(*)
   240   Dw=D(3)-D(1)
   250   Dh=(D(4)-D(2))*((Nlines-7)/Nlines)
   260   !
   290   Pw=Dw*.95                                ! PANEL parms.
   300   Ph=Dh*.25
   310   Px=(Dw-Pw)/2
   320   Py=(Dh-Ph)/2
   340   S$="XHELPTST.HLP"                        ! Help file names.
   350   !
   360   ! ******************************************************************
   370   !
   380   ! Create the PANEL widget, add a toaster menu.
   390   !
   400   ASSIGN @Main TO WIDGET "PANEL";SET ("VISIBLE":0)
   410   CONTROL @Main;SET ("X":Px,"Y":Py,"WIDTH":Pw,"HEIGHT":Ph)
   420   CONTROL @Main;SET ("MAXIMIZABLE":0,"RESIZABLE":0)
```

```
430   CONTROL @Main;SET ("TITLE":"Weapons Control","SYSTEM MENU":"Quit")
440   CONTROL @Main;SET ("HELP FILE":S$,"HELP TOPIC":"weapons.TOC")
450   STATUS @Main;RETURN ("INSIDE WIDTH":Iw,"INSIDE HEIGHT":Ih)
460   !
470   Gx=Iw*.02
480   Gy=Gx
490   Btnw=(Iw-4*Gx)/3
500   Btnh=Ih-2*Gy
510   Y=Gy
520   X1=Gx
530   X2=X1+Btnw+Gx
540   X3=X2+Btnw+Gx
550   !
560   ! Set up buttons.
570   !
580   ASSIGN @Phasers TO WIDGET "PUSHBUTTON";PARENT @Main
590   CONTROL @Phasers;SET ("X":X1,"Y":Y,"WIDTH":Btnw,"HEIGHT":Btnh)
600   CONTROL @Phasers;SET ("LABEL":"Fire Phasers")
610   CONTROL @Phasers;SET ("HELP FILE":S$,"HELP TOPIC":"weapons.phasers")
620   !
630   ASSIGN @Torps TO WIDGET "PUSHBUTTON";PARENT @Main
640   CONTROL @Torps;SET ("X":X2,"Y":Y,"WIDTH":Btnw,"HEIGHT":Btnh)
650   CONTROL @Torps;SET ("LABEL":"Fire Photon Torpedoes")
660   CONTROL @Torps;SET ("HELP FILE":S$,"HELP TOPIC":"weapons.torp")
670   !
680   ASSIGN @Off TO WIDGET "PUSHBUTTON";PARENT @Main
690   CONTROL @Off;SET ("X":X3,"Y":Y,"WIDTH":Btnw,"HEIGHT":Btnh)
700   CONTROL @Off;SET ("LABEL":"Stand Down")
710   CONTROL @Off;SET ("HELP FILE":S$,"HELP TOPIC":"weapons.off")
720   !
730   ! Set up events and loop.
740   !
750   ON EVENT @Phasers,"ACTIVATED" GOSUB Phasers
760   ON EVENT @Torps,"ACTIVATED" GOSUB Torps
770   ON EVENT @Off,"ACTIVATED" GOSUB Off
780   ON EVENT @Main,"SYSTEM MENU" GOTO Finis
790   !
800   CONTROL @Main;SET ("VISIBLE":1)
820   LOOP
830   END LOOP
840   STOP
850   !
860   ! ******************** Subroutines Start Here **********************
870   !
880 Phasers: DIALOG "INFORMATION","Fired phasers!"
900   RETURN
910   !
920 Torps: DIALOG "INFORMATION","Fired photon torpedoes!"
940   RETURN
950   !
960 Off: DIALOG "INFORMATION","Weapons system standing down."
980   RETURN
990   !
1000  ! *********************** Go Here When Done ***********************
1010  !
1020 Finis: !
1030  ASSIGN @Main TO *                       ! Kill off main panel.
```

```
1040  END
1050  !
1060  ! *********************** That's All, Folks! ***********************
```

Program Example: xhelptst.rmb

As will be explained in the next chapter, there are features in the BPlus CONFIG file that allow you to control the behavior of context-sensitive Help:

* f1_is_help: Set or clear f1 key for help access.
* rht_ms_btn_is_help: Set or clear right mouse button for help access.
* provide_defaults: Set or clear providing default widget HELP TOPIC.

[18.0] BPLUS (18): Miscellaneous Topics

v2.4 / 01 feb 99 / greg goebel

* This chapter covers a number of issues that weren't convenient to discuss elsewhere.

[18.1] BPLUS CONFIG FILE

* The BPlus CONFIG file allows you to specify the following parameters for BPlus:

   * The disk volume where BPlus files are found.
   * RAM usage by BPlus.
   * The default BPlus system font.
   * Mouse, SLIDER widget, and context-sensitive help handling.
   * What fonts, widgets, and apps to preload.
   * What applications are loaded into Application Manager.
   * The name of a "tiling" bitmap file for the Applications environment.
   * The PEN settings for the PENS used by BPlus itself for its widgets.
   * What PENs are available to BPlus on different displays.
   * The PEN mappings for the user under BPlus.

The BPlus CONFIG file is a SAVEd program file with a program listing that consists entirely of comments; section headings are preceded by "@!"; configuration-file comments are preceded by "!#". Actual configuration information is only preceded by a "!".

This file MUST be SAVEd as an HPUX text file (same as DOS text file under DOS File System for the Measurement Coprocessor or for HBW), not an RMB ASCII file.

BPlus searches for CONFIG (after LOAD BIN "BPLUS" or SCRATCH A) in the following locations (listed in search order):

   * File CONFIG in current MSI.
   * Under RMB/UX, $HOME/CONFIG.
   * Under SRM, /SYSTEMS/CONFIGnn (<nn> = node number)
   * Under HFS, /WORKSTATIONS/PLUS/CONFIG.

* Under MCP, <blpdir>/PLUS/CONFIG, where <blpdir> is the MCP boot directory as designated by the BLPDIRn environment variable.

Excerpts from a sample CONFIG file follows, interspersed with descriptive comments:

1: BPlus files volume: This optional configuration command specifies what disk volume the BPlus auxiliary files -- widgets, apps, fonts, and so on -- are found on:

```
   320   !@ system MSI
   360   !# /BPLUS:,700,2
```

2: RAM usage (not applicable to HBW): You can use these three configuration commands to determine how much RAM RMB will set aside for BPlus at load time ("pre_allocate"), how big an increment will be used to get more memory for BPlus ("block_size"0), and the maximum amount of memory that BPlus will be allowed to take ("max_allocate").

```
   380   !@ RAM usage
   390   !# These parameters are only used on non-RMB/UX systems
   400   !  pre_allocate   50000          # 50000 minimum
   410   !  block_size     10000          # 5000 minimum
   420   !  max_allocate   2147483647     # 50000 minimum
```

3: Offscreen memory use on TOPCAT display (not applicable to HBW): If you don't know what this means, you don't need to worry about it.

```
   440   !@ use offscreen memory
   470   !  TRUE
   480   !
```

4: System font: You can specify a specific font in terms of width by height (say, 8X16) or specify the default font (which changes from display to display.

```
   490   !@ system font
   520   !  default
```

5: Mouse handling (not applicable to HBW): You can use these entries to determine the maximum delay (in milliseconds) that will be recognized for a mouse "double click", and to specify whether a BPlus window will be selected when clicked anywhere, or just on the title bar:

```
   540   !@ mouse
   550   !  doubleclick_interval  500
   560   !  raise_on_any_click  FALSE
```

6: SLIDER widget handling: You can specify the delay before auto-repeat starts and the interval between repeats:

```
   600   !@ scrollbar
   610   !  initial_delay  500
   620   !  repeat_delay   100
```

7: Context-sensitive Help handling: These entries allow you to specify whether by default to display the Help entry for a widget when you click on it with the secondary mouse button, or do nothing; to use the F1 function key to select help for a widget, or ignore the F1 key; and to use the secondary (normally the right) mouse button to select help for the widget, or to ignore it.

```
   640   !@ context help
   650   !  provide_defaults    true
   660   !  f1_is_help          true
   670   !  rht_ms_btn_is_help  true
```

8: Which fonts to load (not applicable to HBW): The full list of font options isn't shown here for the sake of brevity, but note that it also includes options for kanji and katakana character-handling on Japanese RMB.

```
   690   !@ load fonts
   940   !  FT6X12
   950   !  FT7X14
   960   !  FT8X11
   970   !  FT8X14
   980   !  FT8X16
   990   !  FT8X16B
   1000  !  FT10X16
   1010  !  FT10X20
   1020  !  FT10X20B
   1030  !  FT12X19
   1040  !  FT16X24
   ...
```

9: Which widgets to load (not applicable to HBW): Note that if you don't specify a widget in this list, it will only be loaded if the BPlus program tries to invoke it.

```
   1170  !@ preload widgets
   1200  !  WIBITMAP
   1210  !  WIBUTTON
   1220  !  WICBAR
   1230  !  WICLOCK
   1240  !  WICOMBO
   1250  !  WIFILE
   1260  !  WIGRAPH
   1270  !  WIHPGL
   1280  !  WIKEYPAD
   ...
```

10: Which applications to preload (not applicable to HBW): If an application isn't specified, it will be loaded on demand:

```
   1490  !@ preload applications
   1530  !  APHELP
   1540  !  APMGR
```

11: Which applications are loaded into the Application Manager (not applicable to HBW): These entries specify the file of the app, the icon and name that the Application Manager will display for it, and the appropriate Help file for that topic.

```
   1590  !@ APP management
   1620  !# name            parms   filename   help topic,file   icon file
   1630  !  "SCREEN BUILDER" ""     APSCRBLD   ,HLSCRBLD         BMSCRBLD
   1640  !  NOTEPAD          ""     APNOTEP    NOTEPAD,HLBASIC   BMNOTEP
   1650  !  CLOCK            ""     APCLOCK    CLOCK,HLBASIC     BMCLOCK
   1670  !  "HELP COMPILER"  ""     APHCOMP    ,HLHCOMP          BMHCOMP
   1680  !  HELP             ""     APHELP     HELP,HLBASIC      BMHELP
```

12: What background bitmap, if any, you want to use to "tile" the Application environment:

```
   1700  !@ background_bitmap
   1730  !# BMASTER   # asteroid
   1740  !# BMAZTEC   # aztec mosaic
```

13: What PENs will be used by the BPlus default colors:

```
   1800  !@ logical pens
   1850  !#                          physical pen for system containing
...
   1860  !# pen  pen name             2 pens  16 pens  64 pens  256 pens
   1870  !  0     ActiveBorder            0       14      14        30
   1880  !  1     ActiveTitleBar          1       14      14        30
   1890  !  2     ActiveTBarForeground    0       1       1         17
   1900  !  3     AppBackground           0       9       9         25
   1910  !  4     AppBorder               0       14      14        30
   ...
```

14: What PENs BPlus can set on various types of displays:

```
   2470  !@ settable pens
   2570  !  16_pen_system    0-15
   2580  !  64_pen_system    0-15, 54, 55, 62, 16-63
   2590  !  256_pen_system   16-31, 230, 231, 238, 0-15, 32-255
```

15: What color patterns are set in these PENs by BPlus:

```
   2610  !@ physical pens
   2700  !# PEN   COLOR           RED     GREEN   BLUE
   2710  !  0     black           0       0       0
   2720  !  1     white           255     255     255
   2730  !  2     red             255     0       0
   2740  !  3     yellow          255     255     0
   2750  !  4     green           0       255     0
   ...
```

[18.2] MONOCHROME DISPLAYS

* BPlus will work on a monochrome monitor, though it isn't extremely attractive on one. If you are using a monochrome grayscale display, you can get a reasonably pleasing user interface as long as you stay with the default colors; of course, that means that features of the user interface that depend on a change in colors are not particularly effective.

If you have an RMB system with a single-plane mono display, the situation is worse, since now there is only black and white, and the entire user interface becomes merely a set of line-draw boxes -- usable, but that's about it. Try to set colors and you end up with blocks of black and white.

[18.3] PRINTER HANDLING

* Dumping a BPlus user interface to a printer on RMB is a little complicated ... You can indeed use the RMB GDUMP_C CSUB to dump a display to a PaintJet printer. However, dumping to a standard printer is troublesome, since the various colors all map to black and all you'll get is illegible blots on the paper; RMB doesn't know how to dither.

The only way to get display dumps of BPlus user interfaces on an RMB system is use the BPlus CONFIG file to set the system up as the equivalent of a one-plane two-color mono system; you'll get a line-draw style display that will dump easily to a printer.

The number of colors can be set with an undocumented CONFIG option; you must add the following two lines to the file:

```
   2540 ! @ num_pens
   2550 ! 2
```

This gives a two-color line-draw display that can be then dumped to a printer legibly.

Another option is to use the BITMAP widget and save your display to a graphics file, and then use a separate graphics program that may be smarter than BPlus to dump the display to a printer.

Of course HP BASIC for Windows uses the Windows printer drivers, and in principle these should handle all the color mapping and grayscale dithering correctly.

[18.4] BPLUS FONT HANDLING

* Under RMB, if you don't specify a font to use, BPlus will pick a default font for you; this font will depend on the display size:

screen width	default font
< 620	"6X12"
620 -> 1024	"8BY16,BOLD"
> 1024	"10BY20,BOLD"

If you prefer to use a specific font that will be retained no matter what the size of the display, you can specify a font using an attribute:

```
   CONTROL @Widget; SET ("FONT":"10 BY 20")
```

You can select from the following fonts:

"6 BY 12"	
"8 BY 16"	
"8 BY 6"	Font rotated by 90 degrees.
"9 BY 15"	
"10 BY 20"	
"15 BY 9"	Font rotated by 90 degrees.
"18 BY 30"	
"6 BY 12,BOLD"	
"8 BY 16,BOLD"	
"8 BY 6,BOLD"	Font rotated by 90 degrees.

```
"9 BY 15,BOLD"
"10 BY 20,BOLD"
"15 BY 9,BOLD"    Font rotated by 90 degrees.
"18 BY 30,BOLD"
```

All these fonts are supplied with BPlus itself.

Program Example: xfonts.rmb

Under HP BASIC for Windows, you will get the best approximation to the desired font from the ones that are available.

[18.5] COMPILING BPLUS PROGRAMS UNDER RMB

* If you write RMB programs containing BPlus code, the program will work fine normally; but if you compile and run the program, it will lock up RMB with a SYSTEM ERROR.

The problem is that "pre-allocate" entry in the BPlus CONFIG file pre-allocates a fixed amount of memory for BPlus to use. If more room is needed, the interpreter allows this amount to be increased; the compiler, however, does not. This means that the compiled program runs out of RAM, resulting in the SYSTEM ERROR lockup.

The fix? Edit the "pre_allocate" entry of BPlus's CONFIG program and increase the amount of pre-allocated memory to cover the program's needs. (After changing the CONFIG file, you must execute SCRATCH A to re-initialize BASIC Plus.)

[18.6] COMMENTS & REVISION HISTORY

* This document started out in 1991 in the form of a set of notes I wrote to prepare for the BPLUS introduction. It was quite a project but paid off well, the materials were used as a basis for the BPLUS manual and were applicable to several later documents.

* Revision history:

```
   v1.0 / 01 sep 91 / gvg
   v1.1 / 01 jan 93 / gvg / Added IBW comments, chopped up, etc.
   v2.0 / 01 nov 93 / gvg / Deleted IBW comments, added stuff for B++.
   v2.1 / 01 apr 96 / gvg / Tidied up for Web, modified for HBW.
   v2.2 / 29 may 96 / gvg / Rearranged hyperlink scheme slightly.
   v2.3 / 26 dec 97 / gvg / Corrected minor typo error.
   v2.4 / 01 feb 99 / gvg / Minor cosmetic update, new title.
```

Agilent Technologies
Innovating the HP Way

Agilent Technologies
Innovating the HP Way
Channel Partner

```
  *************************************************************
Cie : BOURBAKY
      BP 53
      13, Rue des Alpes
      07302 TOURNON Cedex - France
Tel (Nat.) : 04 75 07 81 20    Tel (Int.) : +33 4 75 07 81 20
Fax (Nat.) : 04 75 07 29 74    Fax (Int.) : +33 4 75 07 29 74

web :    <http://www.bourbaky.com>
e-mail : <info@bourbaky.com>
  *************************************************************
```

RMB

A BASIC Plus Notebook

----

v2.4 /01 feb 99 / greg goebel