ASPECT™ Script User’s Guide

ASPECT Script User Guide

ASPECT Script User's Guide November, 1999.

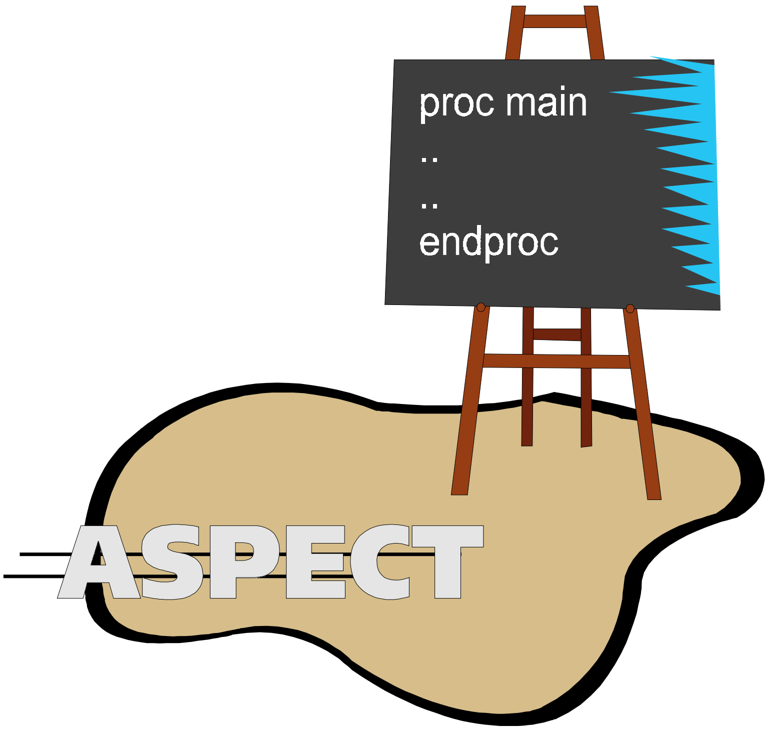
Symantec Corporation
10201 Torre Avenue
Cupertino, CA 95014-2132

10987654321

Table of Contents

Chapter 1

Understanding ASPECT



Introduction

This documentation set is designed as a resource for Procomm Plus ASPECT developers. As a reference guide, it provides:

- Detailed information on each script component.
- Procedures for creating, editing, and compiling scripts.
- Techniques for common script tasks.
- Advanced script techniques.
- Script-related technical information.

If you're new to ASPECT, or just looking for a few pointers, "*A Brief ASPECT Tutorial*" on page 453 is a great introduction to the scripting language and provides many examples. "*Script Basics*" on page 10 contains information about ASPECT in general and about writing ASPECT applications. "*The Elements of ASPECT*" on page 19 discusses the basic syntax and conventions of ASPECT. If you're not sure about scripts and their applications, examine"*What Are Scripts?*" on page 8.

Each command is detailed in "*The ASPECT Commands*" on page 65 and the parameters for the **set** and **fetch** commands are in "*Set and Fetch Statements*" on page 363.

"*The ASPECT Utilities*" on page 509 describes the tools provided to make ASPECT programming even easier. You can find out how to take advantage of tools like the *Dialog Editor* and the *User Window Editor.* The *ASPECT Editor* integrates your compiler and editor for even shorter development times. For more information about the *ASPECT Editor*, see the Procomm Plus help system.

Once you've written some scripts, odds are you'll need to debug them. "*Debugging Scripts*" on page 555 includes, among other things, a list of ASPECT "*Execution or Run-Time Errors*," "*Logic Errors*,"and "*Compile-Time or Syntax Errors*." There is also information regarding "*The ASPECT Compiler*" on page 567, detailed information about "*Using Remote ASPECT Commands*" on page 609, and a list of "*ASPECT Reserved Words*" on page 585.

What Are Scripts?

Scripts are a means of automating repetitive tasks. Procomm Plus uses scripts to quickly automate anything from minor tasks like logging onto an information system and retrieving messages to major tasks like creating and managing thousand-record databases. ASPECT can automate these tasks and more. Why not automate your tasks and let Procomm Plus do the work for you?

Now, in case you're not sure about what ASPECT can automate, let's outline a few more of the many applications for ASPECT:

- Automated fax operations.

  If you regularly send or receive faxes, an ASPECT script can easily handle the task for you. It can automatically forward your received faxes to you at another location, and advise you if there was a problem.

- Automated file transfers.

  If you need to transfer files at regular intervals, it's easy to automate the process as a "background" task. An ASPECT script can connect to a host system and transfer files unattended at the time you set. If an error occurs or Procomm Plus can't connect, your script can warn you with a dialog message or an alarm sound.

- Script applications.

  With ASPECT, you can create script modules that integrate seamlessly with Procomm Plus. Windows dialog controls, file I/O, printer output, *Dynamic Data Exchange* (DDE), and error-free packet transfers can all be created with ASPECT. For example, your script could retrieve its data from a Microsoft Excel spreadsheet before it connected to your host, or it could display a dialog box allowing you to select Web pages to retrieve while you are on-line. ASPECT offers the power and tools you'll need to build just about any Windows application!

- Running your own BBS.

  You can use the ASPECT *Host* script to run your own bulletin board system. The *Host* script features public and private messaging, file transfers, a *Chat Mode*, call-back and fax-on-demand capability. You can even modify the *Host* source file and re-compile it to add new features or customize the appearance of your system!

- "Packet mode" transfers.

  Two PCs running Procomm Plus can transfer binary data continuously to each other in "packets." This error-free method ensures that commands and characters you type are received accurately by the remote system.

- Customized features or commands.

  You can customize Procomm Plus to add extra commands to the menu system, or you can display your own User window graphics as a portion of the *Terminal window*, complete with "hot spots," icons and buttons.

If you are new to ASPECT, or to programming in general, we highly recommend that you review the "A Brief ASPECT Tutorial" on page 453. While it can't provide a complete programming education, it does introduce you to many of the basics involved in writing your own scripts. If you're familiar with other structured programming languages, picking up ASPECT should be no problem. Similarly, if you've used ASPECT in the past, this overview may be the only refresher course you need!

Script Basics

ASPECT is a complete, structured programming language. Because ASPECT files can only be run from within Procomm Plus, it is referred to as a scripting language and the files created are called scripts. ASPECT scripts are contained within two types of files: source files and compiled files.

Source files are standard text files written in the *ASPECT Editor*, or some other text editor. After writing the source code, simply save it as a **.was** (**W**indows **A**SPECT **S**ource) file. ASPECT, like most high-level programming languages, requires that your programs be "compiled." These compiled scripts are stored in **.wax** files (**W**indows **A**SPECT e**X**ecutable) which Procomm Plus can run.

Once you've written the source code, the ASPECT Compiler turns it into a **.wax** file that can be processed by Procomm Plus. During this process, the commands, data, and text contained in a **.was** file, or other source file, are checked for syntactical accuracy and reduced from their full-length "readable" format into code that Procomm Plus can read and process. After compiling, the resulting **.wax** file is smaller in size. Also, because string data is encrypted in the **.wax** file, it's much more secure! No one can read your sensitive passwords or other information.

A source script file must be compiled before it can be executed. Source files are not affected in any way by the compile process.

*A compiled **.wax** file cannot be "un-compiled." That is to say that it cannot be converted back to its **.was** source.*

Creating and Editing Scripts

Scripts are created and modified whenever you modify their source files. Within Procomm Plus, it is exceptionally easy to create a script "from scratch" or to change an existing script's source file. Just select ***Tools* | *Scripts* | *Compile/Edit*** from the menu, or click on the *Action Bar's* ***Compile/Edit*** icon. Procomm Plus displays the **Compile/Edit ASPECT File** dialog.

To create a new script, click on the ***New*** button. Procomm Plus invokes the script editor with an **untitled** window. Be sure to save your new file with a **.was** extension.

To modify an existing source file, select the file you'd like to edit and click on the ***Edit*** button. Procomm Plus invokes the script editor with the file you selected.

By default, Procomm Plus opens the *ASPECT Editor* as the script editor. For more information about the *ASPECT Editor*, see the Procomm Plus help system. If you prefer, you can use a different text editor by specifying it in the ***Editor Name and Path*** field in the ***System, System Options*** panel of the **Setup** dialog. For instance, if you wanted to use an editor in your **c:\util** directory with a filename of **zedit.exe**, you'd specify **c:\util\zedit.exe** in this field.

You can also delete script files you no longer need from within the **Compile/Edit** dialog. Select the name of the file to erase and click the ***Delete*** button. Procomm Plus will prompt you for confirmation before deleting the selected file.

Procomm Plus provides three powerful scripting utilities to help you create and modify ASPECT scripts:

The ASPECT Editor, a feature-rich text editor that you can use to edit your ASPECT source files and other text files. You can find more information about the ASPECT Editor in the Procomm Plus help system.

The Dialog Editor, a graphical tool that you can use to create and edit dialog boxes for your scripts. For more information, see "The Dialog Editor" on page 510.

The User Window Editor, a graphical tool that you can use to create and edit graphical displays for use within Procomm Plus's Terminal window. For more information, see "The User Window Editor" on page 535.

Compiling Scripts

Before Procomm Plus can execute an ASPECT script, it must be compiled. The *ASPECT Compiler* converts the script commands and statements within an ASPECT **.was** source file into symbols that Procomm Plus can interpret and execute, a **.wax** file. While not all programs with scripting capabilities require their scripts to be compiled, compiled scripts offer several advantages:

- Compiled scripts are smaller, and generally run faster than interpreted text scripts.
- Compiled scripts are more secure. The *ASPECT Compiler* automatically encrypts all string data, allowing you to share your compiled scripts with other users without revealing sensitive information.
- Compiled scripts can easily support multiple-file projects. By using compiled scripts, ASPECT can support **#include** files and **#define** macro statements, allowing you to build powerful libraries of your own ASPECT functions.

ASPECT script files are easily compiled from within Procomm Plus. You can select ***Tools | Scripts | Compile/Edit*** from the menu, or click on the *Action Bar's* ***Compile/Edit*** button to open the **Compile/Edit ASPECT File** dialog.

Once the **Compile/Edit ASPECT File** dialog is open, click the ***Compile*** button. Procomm Plus attempts to compile the source file you selected. If successful, the message "*Script file successfully compiled!*" is displayed in the compiler window, and a **.wax** file is generated. If you click the ***Compile and Run*** button, Procomm Plus will run the script immediately if the compilation is successful.

If the compilation is unsuccessful, specific warning and error messages are displayed in the ***Message List*** box of the **ASPECT Script Compiler** dialog. You can use these messages to help you "debug" your source file, then compile the script again.

For more information about debugging your scripts, you can read the "Debugging Scripts" on page 555. For additional information about compiling your scripts see"The ASPECT Compiler" on page 567.

Script Record Mode

Many script applications are repetitive tasks that can be duplicated using script commands. For example, scripts that automate logons or file transfers are usually simple sets of keystrokes that are repeated each time you go on-line. For procedures like these, you can use *Record* mode to save each prompt sent by the host system and each response you provide.

The *Script Recorder* converts these prompts and responses into script commands and allows you to save them in an ASPECT source file. Compile the source file, and you can use the resulting **.wax** file to automatically perform that procedure!

Generally, you will find this tool most useful when you are in the *Terminal window* or *Telnet window*. The Step-by-step "*Recording a Logon Script*" on page 17 provides detailed instructions for using the *Script Recorder*.

Running Scripts

You can run a script in many different ways:

- By selecting it from the *Action Bar*'s ***Script File*** drop-down list box.

  All of Procomm Plus's modes except *Internet Mail* and *News* include a ***Script File*** drop-down list box and a ***Script File*** icon on the default *Action Bar*s. The list box displays all compiled scripts in the default script directory. Click on the script name you want, and Procomm Plus will execute it.

  If the script you want to run is already displayed in the ***Script File*** list box, you can run it by clicking the *Action Bar*'s ***Start Script*** icon or by double-clicking on the list box.

- By selecting it from the **Run ASPECT File** dialog.

  From the *Terminal window*, the *Telnet window*, or the *FTP Client window*, you can select ***Tools* | *Scripts* | *Run*** to open the dialog, select your script, and click on ***OK***. Note that only successfully compiled **.wax** files will be available in this dialog.

- By selecting it from the **Compile/Edit ASPECT File** dialog.

  Within *Data Terminal*, *Telnet*, or *FTP Client*, select ***Tools* | *Scripts* | *Compile/ Edit***, or click on the Action Bar's ***Compile/Edit*** button to open the dialog, then select your script and click on ***Compile and Run***. Procomm Plus will attempt to compile your selection, and run it automatically if the compilation succeeds. If the script is already compiled, you can click on ***Run*** to execute it.

- By associating it with a *Connection Directory* entry.

  If you specify a script in a *Connection Directory* entry's ***Script*** field, it will be executed automatically each time you call that entry.

- By clicking on a ***Run Script*** *Action Bar* button.

  If you've configured a custom *Action Bar* button as a ***Run Script*** button, you can launch its associated script with a single click. You can customize an *Action Bar* with the *Action Bar Editor*. See the Procomm Plus User Guide for more information.

- By pressing a ***Run Script*** *Meta Key*.

  If you've configured a *Meta Key* as a ***Run Script*** button, you can launch its associated script with the touch of a key or a single mouse click. You can create custom *Meta Keys* with the *Meta Key Editor*. See the Procomm Plus User Guide for more information.

- By pressing a ***Run Script*** *Terminal* key.

If you've configured a mappable key as a ***Run Script*** key, you can launch its associated script with the press of a key. You can map keys with the *Keyboard Editor.* See the Procomm Plus User Guide for more information.

◆ By launching it from another script.

  To run a script from within another script, you can use either the **execute** or **chain** command. The launched script can even share information with the called scripts using the predefined global variables, opened files, and allocated memory!

  For more information, see the "*Script Spawning and Chaining*" on page 55. You may also want to see the **chain**and **execute** commands.

◆ By naming it **startup.wax**.

  Each time it launches, Procomm Plus searches your script directory for a compiled script with this special name. If it finds the file, Procomm Plus executes it automatically.

◆ By adding it to Procomm Plus's ***Target*** field in the ***Shortcut*** tab of its **Properties** dialog.

  The script argument can be used alone, or with up to 10 arguments which will be passed to the script as values for the predefined string variables S0 through S9. You must specify either a **.was** or **.wax** extension.

  After it has initialized, Procomm Plus executes the script immediately.

  The script can determine how it was started by testing the value of the $SCRIPTMODE system variable. It can also determine the number of command-line arguments by testing the value of the predefined I0 integer variable. As an example, if you used the default installation directories, you could specify the compiled ASPECT script **dialout.wax** to run automatically, initializing S0 to ***"DIALFAST***" with the following ***Target*** line:

  C:\Program Files\Prowin95\programs\pw4 dialout.wax "DIALFAST"

◆ By using a Dynamic Data Exchange (DDE) command from another application.

  Procomm Plus can run a script automatically when it's acting as a DDE server. It can also return values through its predefined global variables via DDE. For more information, see "*DDE server operation*" on page 498.

◆ By using a remote ASPECT command.

  If you're connected to a computer that is also running Procomm Plus, you can launch a script on that machine by transmitting a remote ASPECT command. The following sequence, received by Procomm Plus, will run the script **remote.wax**:

  <Ctrl><D> execute "REMOTE.WAX" <Ctrl><D>

"<Ctrl><D>" in the example above represents the decimal 4 value ASCII character. For more information, see "*Using Remote ASPECT Commands*" on page 609.

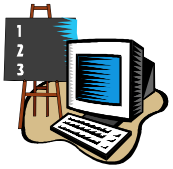
- ◆ By specifying it as a response to *Caller ID*.

  Procomm Plus can launch a script in response to *Caller ID*. See the Procomm Plus User Guide for more information.

- ◆ By specifying it as a task in the Procomm Plus *Scheduler*.

  For more information about the *Scheduler*, see the Procomm Plus User Guide.

While a script is running, the ***Tools | Scripts | Run...*** menu item changes to ***Stop Script***. You can select the ***Stop Script*** menu item, or click on the *Action Bar* ***Start Script*** icon to end script execution.



Recording a Logon Script

The Script Recorder makes script writing easy. Use the Script Recorder from within the Terminal window or Telnet window to record a script. Follow these steps to create a script for logging on to a favorite BBS:

1. **Start the *Script Recorder.***

   Select ***Tools* | *Scripts* | *Start Recorder*** from the menu, or click on the *Action Bar*'s ***Recorder*** button. After the *Script Recorder* is started, the ***Start Recorder*** menu command changes to ***Stop Recorder***.

2. **Connect to your favorite BBS.**

   Once you've connected you will be presented with a Welcome screen that prompts you for your full name

   In order to create scripts, the *Script Recorder* "watches" the characters you receive, and the characters you type in response. For this reason, it is important that the Script recorder be active *before* you receive the BBS's welcome screen.

3. **Log on to the BBS.**

   Type in the user name you have established on the BBS. Press <Y> to acknowledge that it has the correct user record. When you are prompted for your password, type it in as well.

   If the BBS prompts you with incorrect information, or if you incorrectly type your name or password, your script will duplicate those mistakes each time you use it. If you made a mistake, however, you don't have to quit, you can just edit the source file later.

4. **Finish your tasks and turn off the *Script Recorder.***

   If you only wanted the script to log you on, then you can turn the *Script Recorder* off now. If you want it to perform additional tasks each time it runs, perform those tasks as you normally would and turn the *Script Recorder* off when you're done.

   Select ***Tools* | *Scripts* | *Stop Recorder*** or click again on the *Action Bar's* ***Recorder*** button to stop recording.

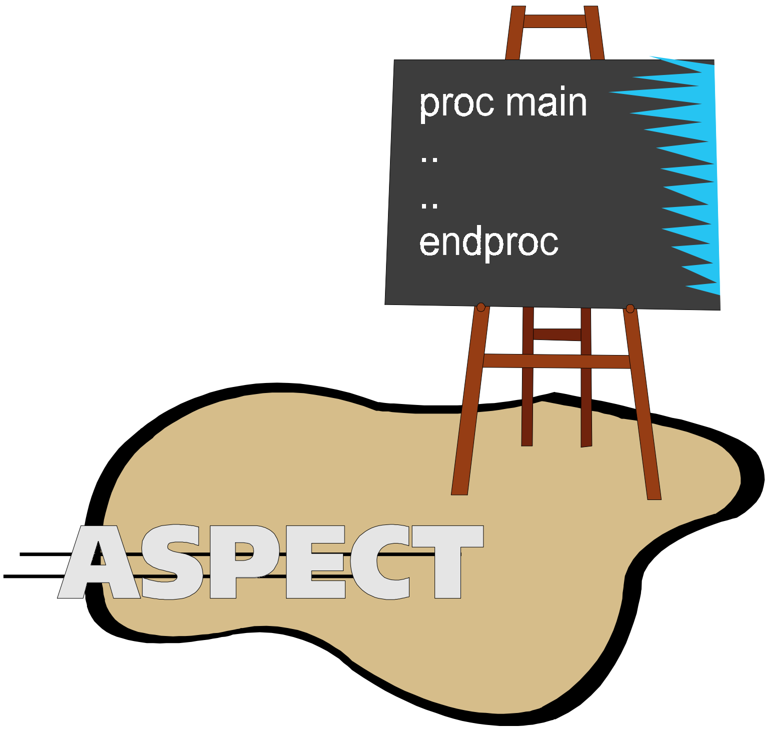
5. **Specify a filename for the new script.**

   Procomm Plus displays a dialog prompting you for a name for the newly recorded script file. Choose a file name that describes the file you're creating; let's name the script file "**myfavoritebbslogon.was**".

   At this point, it's a good idea to save the script and edit it, rather than attempting to compile it immediately. Scripts generated with *Record* mode sometimes require "fine-tuning" before they’ll run successfully. For example, Procomm Plus might have inadvertently captured line noise, or you may need to lengthen the script’s pause in order to allow the host system more time to respond.

Chapter 2

The Elements of ASPECT



Introduction

This section introduces the basics of the ASPECT programming language, including data and variable types, macro definitions, arrays and operators. We'll also cover the basic conventions used throughout this manual and the rules that govern all ASPECT scripts.

ASPECT Building Blocks

The basic building blocks of ASPECT are commands and expressions. Each line in a source or **.was** file begins with one of the following:

A command. An ASPECT command is usually followed by one or more parameters which control the function of the command in some manner, or provide constants or variables for the command to act upon. Within this documentation, items denoted as KEYWORDs or *arglist*s are examples of parameters. Some parameters are required, but most parameters are optional. More information is provided in “*ASPECT Command Form*” on page 54.

An expression. An expression consists of one or more operands acted upon by one or more operators. For example, in the expression:

a = b + c

a, **b**, and **c** are operands, while “=” and “+” are operators. Operands usually occur as numeric constants or variables, but they can also be the result of a sub-expression within the expression, like the “=” sign in our example above, which uses the result of the sub-expression **b** + **c**.

A label. A label denotes a target location within a local procedure which the script can jump to with the **goto** command. A label marks the location of a particular script command for later reference. The label may appear on the line preceding a command, or on the same line as the command. Labels in ASPECT are not case-sensitive; they take the form:

LABEL:

Additional details can be found in “*Labels*” on page 35.

A comment. A script source comment is denoted with a semicolon (;) or with the **#comment** command. Any characters to the right of a semicolon, or within a **#comment** block are ignored by the compiler and are not part of the compiled script. Comments allow you to document your source file in detail without losing speed or efficiency within the executing script.

Commands and expressions all conform to rules called ***syntax***. Like the syntax of written languages, programming syntax ensures that the function of each ASPECT command is clear and precise.

The Description Line

Procomm Plus uses the first line of a source file for special processing. If that line is a comment beginning with a semicolon, Procomm Plus displays the first 50 characters of the line as a description for that script in the **Compile/Edit ASPECT File** dialog, and in the **Run ASPECT File** dialog. For example, if the first line of a source file read:

; Logon to XYZ Technical Support BBS.

that text would also appear in the ***Notes*** field in the **Run ASPECT File** dialog when the compiled script was selected.

The description line can be of any length, but only the first 50 characters will be displayed. If the first line of the source file is not a comment beginning with a semicolon, it is processed by the compiler as a normal command line, and the ***Notes*** field remains blank.

Command Words and Parameters

Each ASPECT command word begins on a separate line, and is always followed by any parameters it requires. The command and its parameters, if any, must be completed on a single line. However, exceptionally long source lines can be extended over several lines if a backslash (\) is used as the last character of each intermediate line. For example, both of these commands are valid:

```
usermsg "Enter your name, please: " ; a one line command

usermsg "Enter your complete name \
including middle initial: "        ; a two line command
```

Note that if a quoted string constant is extended to two lines, any spaces occurring before the text on the second line will be included in the constant. No comments may follow the backslash. The maximum length of a command line in a source file is unlimited. In order to maintain readability, however, it is generally recommended that you use a maximum length of 80 characters.

Command words and parameters cannot be abbreviated, and they must be separated from each other by a space or a tab character. You can also use commas to separate argument list items in most cases; in some special cases, they may be required.

Commands and their parameters are not case-sensitive, except for string information between quotes, as in “Hello,” and format specifiers like those used in the **strfmt** command, as in “%d”.

You must enter the parameters shown in a command’s syntax listing unless the parameter is enclosed in brackets or otherwise identified as optional.

Data Types

ASPECT supports four data types:

Type	Description
integer	Values range from -2147483648 to 2147483647. An integer requires 4 bytes of storage.
float	Values range from a minimum positive value of 2.22507385072014e-308 to a maximum value of 1.7976931348623158e+308. Rounding is performed after 15 digits of decimal precision. A float requires 8 bytes of storage.

Type	Description
long	Values range from -2147483648 to 2147483647. A long requires 4 bytes of storage.
string	May contain from 0 to 256 characters.

For an explanation of how numeric and string constants are interpreted by the ASPECT *Compiler*, see “*User-defined Constants*” on page 23.

User-defined Constants

A numeric constant is assigned to a particular data type based upon its value and representation. A floating point constant is a decimal number representing a signed real number; it contains an integer part, a fractional part, and an exponent. Floating point constants have the following formats:

[-] digitstr. [digitstr] [e|E [+|-] digitstr]

or [-] .digitstr [e|E [+|-] digitstr]

or [-] digitstr e|E [+|-] digitstr

where *digitstr* is a sequence of one or more decimal digits.

No spaces may occur between the characters in a numeric constant. Spaces are shown in the format syntaxes above for readability.

Additionally, you may notice the “[]” notation around the plus and minus signs in the examples above. For an explanation of this and other syntax notations we use when describing ASPECT, see“ASPECT Conventions” on page 42.

Integer and long constants are signed values which can be represented in decimal (base 10), octal (base 8) or hexadecimal (base 16) notation. A minus sign can precede a constant to make it a negative value.

If the constant begins with the characters “0x” or “0X,” it represents a hexadecimal constant; digits ’0’ through ’9’ and ’A’ (or ’a’) through ’F’ (or ’f’) are allowed. If a constant begins with the digit 0, it represents an octal constant; digits ’0’ through ’7’ are allowed. If a constant does not begin with either “0x,” “0X,” or “0,” it is assumed to be a decimal value.

The letter 'l' or 'L' may be applied as a suffix to a integer or long constant to force it to have long type. The uppercase letter is recommended because the lowercase letter can easily be confused with the number 1. If a suffix is not used, the type assigned to a constant is determined from its value in the absence of a leading minus sign. If the constant is a decimal value, and its value is within the positive range allowed by integer values, it is assigned integer type. Otherwise it is assigned long type. If the constant is a hexadecimal or octal value between 0 and 0xFFFFFFFF or 037777777777, the constant is assigned integer type; otherwise it is assigned long type. If the value is decimal and is greater than the largest positive long value, a compile-time error will result.

For example, the smallest integer value -2147483648 can only be represented in hexadecimal or octal notation since, in the absence of the minus sign, the value of the constant is greater than the largest positive integer value. Thus 0x80000000, for example, can be used to represent this number without generating compile-time errors. Note that -0x80000000 or -020000000000 will also work.

Note that since integers and longs have the same range of values, the only way to denote a long constant is to use a long suffix.

ASPECT does not provide any unsigned data types. However certain commands will treat an integer value as an unsigned value. This means the integer value is always considered a positive value ranging from 0 to 4294967295, even if the integer is specified as a negative decimal value. Since a positive integer constant may only be specified in decimal up to 2147483647, another notation, hexadecimal or octal, is generally used to represent unsigned integer values larger than this. A negative integer value will work, but its meaning is not as clear as the hexadecimal or octal representation of that number. For instance -1 is the same as 0xFFFFFFFF or unsigned value 4294967295, -2 is the same as 0xFFFFFFFE or unsigned value 4294967294, and so forth down to -2147483648 which is 0x80000001 or unsigned value 2147483649, and 0x80000000 which represents the integer value -2147483648 or the unsigned integer value 2147483648.

A string constant is represented by a list of up to 256 characters enclosed within double quotes. Any ASCII character value may occur in a string variable, although some characters are not displayable. To use a quotation mark within a text string, precede it with the special escape character “`”, ASCII 96. For example:

```
usermsg "She said `"Hello.`""
```

The NULL character, ASCII 0, normally indicates the end of a text string and is not considered a part of the 256-character limit. Special escape sequences can be embedded in strings using the “back tick” character, as discussed in “*Escape Sequences*” on page 25.

For security, all string constants are encrypted during compilation so that they cannot be viewed in the compiled script.

Escape Sequences

ASPECT uses escape sequences in string and character constants to represent non-displayable characters and control codes which have special meanings to hardware devices. They consist of a backwards single quote character "`", followed by an alphabetic character or series of digits. The "`" character is ASCII 96, available on the unshifted tilde key; it is sometimes called the "back tick." The following table details the escape sequences supported by ASPECT:

Sequence	ASCII Value	Description
`a	7	Alert or bell
`b	8	Backspace
`f	12	Form feed
`n	10	New line/line feed
`r	13	Carriage return
`t	9	Horizontal tab
`v	11	Vertical tab
`'	39	Single quote
`"	34	Double quote
``	96	Backwards single quote
`000	ASCII character	octal notation
`xhhh	ASCII character	hex notation

If a backwards single quote precedes a character not listed above, the backwards single quote will be ignored.

When a "000" or "xhhh" sequence is used, it may consist of one to three octal or hexadecimal digits ranging in decimal values from 0 to 255. Values greater than 255 will generate an error. It is usually best to use all three digits, as in `004, since a letter or digit immediately following an escape sequence can represent either an octal or hexadecimal character which would then be interpreted as part of the escape sequence.

Character Constants

A character constant is an integer value representing an ASCII character within the range 0 through 255. It consists of a single quote, ASCII 39, followed by an ASCII character and a terminating single quote. An escape sequence may also be used to represent an ASCII character within a character constant. For example:

integer letter_a = ’a’   ; (integer constant 97)

integer escape = ’`033’ ; (integer constant 27)

Naming Elements in ASPECT

All of the named elements in ASPECT, such as command word names, procedure and function names, labels, variable names, and **#define** macros follow the same basic syntax rules:

Name Length. Names have a maximum length of 30 characters. The compiler treats all of these characters as unique, but they are not case sensitive. Therefore, the *ASPECT Compiler* would consider two separate variables with names identical for the first 29 characters to be different so long as the 30th characters were different. For example, the two following variables would be considered unique by the *ASPECT Compiler*:

string NotAVeryOriginallyNamedString1

string NotAVeryOriginallyNamedString2

If two *identical* local variables are defined within the same procedure or function block, the second such variable would be flagged as an error by the *ASPECT Compiler.*

Allowed Characters. Names can begin with any alphabetic character, A-Z or a-z, or an underscore “_.” Other characters in the name can be alphabetic, the underscore character or numeric, 0-9. Examples of valid names are “NUM”, “_FINAL_”, “Send_2_Where”, “e” and “Term5”.

Reserved Words. All pre-existing names, like command words and system variables, are considered reserved within ASPECT, and cannot be used as user-defined variable, function, or procedure names. Reserved words can be used in macro definitions, however. “*ASPECT Reserved Words*” on page 585 contains a complete list of reserved words.

Predefined Variables

For your convenience, ASPECT provides a total of 40 “predefined” global variables which you do not have to define at the beginning of a script. They are available for use by any command, procedure, or function within a script.

There are 10 predefined string variables, labeled S0 through S9. Each can contain a maximum of 256 characters. The predefined numeric variables are labeled I0 through I9 for integer values, L0 through L9 for long values, and F0 through F9 for float values.

Predefined variables are initialized to 0, or to null strings in the case of string variables, when Procomm Plus is started. If a script uses these variables, their values will remain unchanged after the script terminates, so they can be used to pass values between separate scripts.

Predefined variables have several uses. If Procomm Plus is executed with a command-line in which the first parameter is a script file name, with either a **.was** or **.wax** extension, the information following the filename will be parsed into individual options. Each option will be loaded into a predefined string variable, beginning with S0. Up to 10 options can be passed to the requested script in this manner. The predefined integer variable I0 will contain a count of the parsed options. The script can use the system variable $SCRIPTMODE to determine whether it was run from the command-line, and reference I0 for the number of options provided.

As mentioned before, predefined variables can be used to pass information between executed scripts. If a script launches or spawns another script using either the **chain** or **execute** command, the launched script automatically inherits all values stored within the predefined variables. Information is passed in both directions; changes made to predefined variables by a spawned script are maintained in the parent script when it resumes execution.

A script can determine whether it was executed by another script, but the convention for the information passed to the second script is left to the script author.

Predefined variables can also be used in Dynamic Data Exchange (DDE). In DDE, an application can change or retrieve the values of Procomm Plus's predefined variables, providing a mechanism for information transfer between the two applications.

System Variables

System variables are "read-only" variables to which ASPECT and Procomm Plus may assign specific values. They are available throughout a script. For example, although you cannot change the value of the variable $ROW, which is always equal to the current row position of the terminal cursor, you can read it from anywhere within a script and use it in any statement where an integer value is accepted.

A script can reference system variables to determine operating conditions, on-going processes, window and task information, user actions, and other information. The results of many ASPECT commands can also be tested with the SUCCESS and FAILURE system variables.

More information, and complete descriptions of all the ASPECT system variables, is available in "*System Variables*" on page 427.

User-defined Variables

ASPECT uses named elements called variables to store and manipulate values within a script. Variables are passed as arguments to commands, and can be used within expressions. User-defined variable names must adhere to the rules for valid ASPECT element names. We recommend that you use easily-recognized names that will help you to identify the purpose of particular variables within your script's source code.

A user-defined variable can be an **integer**, **float**, **long**, or **string**. For more information about the use of user-defined variables, see "*Global and Local Variables*" on page 28.

Global and Local Variables

ASPECT supports the use of both global and local user-defined variables. The terms "global" and "local" refer to the "scope" of a variable. That is, whether a variable is visible to the *ASPECT Compiler* throughout a script's source code, or only visible within a particular function or procedure block.

Global Variables

Global variables cannot be defined within a procedure or function block, and must be defined before they can be referenced. Typically, they are defined at the top of a source program, before any procedures or functions are declared.

A global variable can be referenced within any procedure or function throughout the script. If a procedure or function changes the value of a global variable, that value is maintained until the variable is acted upon by another command anywhere else in the script. User-defined globals cannot be passed to another script by chaining or spawning; only the predefined variables can pass information to another script. See"*Predefined Variables*" on page 26 for more information.

Global variable names must be unique. Attempting to define a global variable with the same name as a pre-existing global variable will result in a compile-time error message.

Local Variables

Local variables are defined with the same commands as globals. Unlike global variables, they can only be referenced within the procedure or function in which they're defined. The same local variable name can be used in more than one procedure or function, but each occurrence is a completely different case and has no effect on any other occurrences.

Unlike global variables, local variables are discarded by ASPECT when the function or procedure in which they're defined **return**s or terminates. Thus, local variable values are not retained between calls to the procedure or function in which they are defined.

Defining Variables

Whether global or local, all variables must be defined before they are referenced in a script. Multiple variable definitions of the same data type may occur on a single line if each definition is separated with a comma.

Variables may also be initialized when they're defined, as in:

```
integer number1, number2=13
```

Up to 256 global variables of each data type can be defined in a single script. In addition, you can define up to 256 global variable arrays for each of the data types. This allows you to define up to 2048 unique global variables in a single script!

The number of local variables allowed is limited by the run-time stack space available at the time the procedure or function is called. In general, this should not be considered a limitation; the stack can grow as large as 64K, which would allow for over 15,000 integer variables! If the run-time stack cannot accommodate the space required by a procedure's or function's local variables, a stack overflow error will occur, and the script will terminate.

Parameter Variables

Any procedure except **proc main** can be defined with up to 12 parameter variables. Parameter variables are similar to local variables in that they may only be referenced within the procedure in which they're defined. Unlike local variables, however, parameter variables are automatically initialized when the procedure or function is called, using the values supplied by the calling statement.

The **call** command allows values to be passed to the procedure or function being called. These values initialize the procedure's or function's parameters.

Parameter variables must be defined at the top of a procedure or function, before any commands or local variables. The order in which parameters are defined determines the order in which they're passed by the calling procedure or function.

Changes made to parameter variables can be passed back to the calling procedure. For more information, see the **call** command.

Additional checking on passed parameters is performed during compilation. Inconsistencies, such as mismatched data types, or too many or too few parameters will result in compiler error messages.

A parameter variable is defined much like a local or global variable, with the addition of the **param** keyword before the datatype statement. Several parameter variables of the same datatype may be defined on the same source line, if each definition is separated from the next by a comma. For more information on defining parameter variables, see the **param** command.

Arrays

An array is a set of data storage areas of like type grouped together, and accessed using a common name. Each data storage area is called an element of the array; an array element is accessed by combining the array name with a subscript expression.

An array is defined using the syntax:

type name[size]...

where type can be either **integer**, **long**, **float**, or **string**. The array name must follow the standard ASPECT naming conventions, and array definitions cannot include initialization values.

You may have noticed the "..." notation following the array syntax definition. For an explanation of this and other syntax notations we'll use to describe ASPECT, see"ASPECT Conventions" on page 42.

You can define both global and local array variables. A left square bracket begins the definition of the first dimension in the array, and a right square bracket ends that dimension. An expression which evaluates as a constant whole-number value is used within the square brackets to define the size of the dimension. For example,

integer myarray[10]

defines an **integer** array called ***myarray*** which has one dimension of ten elements. In other words, ***myarray*** is a single-dimensional array with ten elements of type **integer**.

Arrays can be defined with up to twelve dimensions. The number of elements in an array definition is determined by the product of the sizes of each dimension. For example, an array declared as:

integer mybigarray[20][21][22]

contains 9240 elements, as 9240 == 20 * 21 * 22. ASPECT limits the overall number of elements in an array to the maximum positive value of a **long** data type, which is 2147483647. However, if the amount of memory required for an array is not available at run-time, an error message will be given.

To access an element of an array, the array name is combined with a subscript expression with the same number of dimensions as the array definition. For example, the statement:

```
i9 = mybigarray[3][4][5]
```

would assign the value stored in ***mybigarray[3][4][5]*** to the predefined integer variable **i9**.

Array indices are zero-based. This means that an array declared as:

```
integer myarray[10]
```

would be referenced using index values from 0 through 9. The statement:

```
i1 = myarray[3]
```

assigns the value of the fourth element of ***myarray*** to the system variable **i1**. While the array definition requires a constant subscript expression, the subscript expression used to access an array element can be any valid expression using operators, constants, variables, and even function calls. An expression is valid for array access if it evaluates to an integer or long value at run-time. This small ASPECT script:

```
proc main
integer index, myarray[10]
  for index = 0 upto 9
    myarray[index] = index
  endfor
endproc
```

uses a simple **for**-loop to set each element in ***myarray*** equal to its index value.

If an array name is used in an expression, it must use the same number of dimensions or subscript expressions as there were in the definition of the array. An array name itself cannot be passed as an argument to a command, procedure, or function, but an element of an array can be passed.

ASPECT will detect out-of-bounds errors on array indices during run-time. However, an out-of-bounds condition will only occur if the overall resulting element offset into the array exceeds the number of elements in that array. For example, given the array:

```
integer mynextarray[4][3]
```

the statement:

```
mynextarray[2][5] = 1
```

would seem to be out of bounds. Instead, it accesses the 11th element of the array, as

```
11 == ((2 * 3) + 5)
```

It would not be reported as an error. To be readable and understandable, however, array accesses should conform to the definition of the array.

Now, let's look at "*Procedures and Functions*" which make up the containers for many of the elements of ASPECT.

Procedures and Functions

A procedure or function is a logical grouping of commands identified by a unique name. The name is provided as a parameter to a **proc** or **func** command, or as an argument to a **call** command. For example:

```
proc Update
```

defines the beginning of a procedure block named ***Update***. Your script could later branch to that procedure with the statement:

```
call Update
```

Unless explicitly stated otherwise, a function can be substituted for a procedure in any command description. Procedures and functions are very closely related; essentially, a function is a procedure that can return a value.

When a function is defined, its return type must be specified. Additionally, a variable of the appropriate type should be defined when a function is called to receive the function's return value. A numeric variable can receive a value from a function that returns a float, long, or integer value; a string variable can receive a value from a function that returns a string.

Every ASPECT program requires at least one procedure called ***main****, which identifies the entry point for command execution. For more information, see"ASPECT Program Flow" on page 53.*

If a procedure or function requires arguments to be passed, the procedure or function name is followed by the WITH keyword, and then a list of arguments. When calling a function, the argument list, if any, can optionally be followed by the INTO keyword, and a variable to receive the function's return value. For example:

```
call myFunc WITH arg0 arg1 arg2 INTO retval
```

calls the function ***myFunc*** with arguments whose names are *arg0*, *arg1*, and *arg2*, and receives the function's **return** value into a variable called *retval*. If a function or procedure receives no arguments, its format could be:

```
      call myProc
(or)  call mySpecialFunc
(or)  call myUniqueFunc INTO retval
```

A user-defined procedure or function can also be called using a shortened format, as in:

```
name ([arglist])
```

You may notice the "[]" notation around the arglist. For an explanation of this and other syntax notations we use when describing ASPECT, see"ASPECT Conventions" on page 42.

If an argument list is present, a comma must used to separate the arguments. If a function is called in this manner, its return value can be assigned to a variable using an assignment operator. A function call with the shortened format can also be used in an expression involving any number of operators and operands. Its return value is treated just like any other value in the expression.

With the shortened format, the examples above would read:

```
retval = myFunc(arg0, arg1, arg2)
myProc()
mySpecialFunc()
retval = myUniqueFunc()
```

This shortened form is thus more flexible, less verbose, and certainly as clear as the longer version. ***We highly recommend the shortened form, as future enhancements of ASPECT may require it.***

While the **call** format does not generally require commas to separate arguments within the argument list, commas are accepted if they appear there. In fact, in one special case a **call** actually requires a comma to separate arguments. This case is discussed below.

Passing by Value and Reference

When a procedure or function requires arguments, any arguments that are constants are passed by value. Arguments that are variables are also, by default, passed by value. This means that the called function or procedure receives a copy of the argument, and any changes made to that argument by the called procedure or function will not alter the value of the argument variable when the function or procedure returns.

If a procedure or function is intended to permanently alter the a variable argument's value, however, the variable can be "***passed by reference***." This means that the variable will return to the original procedure and its value will be whatever value its corresponding parameter in the **call**ed procedure or function last had.

To pass a variable by reference, an ampersand character (&) must be placed before the name of the variable. For example the statement:

myfunc(&arg0, arg1, &arg2)

calls the function ***myfunc*** with arguments *arg0* and *arg2* passed by reference. When ***myfunc*** returns, *arg0* and *arg2* may have new values assigned to them, but *arg1* will have whatever value it had when the **call** was executed. The **call** format would look like this:

call myfunc with &arg0 arg1, &arg2

Note that a comma was placed between *arg1* and *arg2*. Without this comma, the *ASPECT Compiler* would have interpreted the two arguments as a single argument resulting from the expression

arg1 & arg2

which performs a bit-wise-AND operation on the two operands. This would be flagged as an error by the *ASPECT Compiler*. In general, if the **call** format is used at all, the use of commas to separate arguments is always recommended and can sometimes be required.

A Technical Note

Technically, ASPECT's parameter passing convention is "***call by value reference***." This means that any variables passed by reference, as explained in "*Passing by Value and Reference*" on page 34, will not be updated until after the called procedure or function returns. The variable's value is passed to the called procedure, where the value is then used or modified by the procedure. Upon return, the value is copied back into the variable which was passed by reference.

“Call by value reference” has two important implications. First, if the **longjmp** command is executed within the called procedure, and the target is some other previously called procedure, variables that had been passed by reference after the **setjmp** command was processed in that target procedure will not be updated with the modified values. Second, if a global variable is passed by reference, its value will not change within the called procedure until it returns. Thus the value of the global itself, and the value of the variable argument corresponding to that global in the called procedure will not necessarily be the same!

While procedures and functions are the preferred method for navigating a program, occasionally you must use “*Labels*,” another element of ASPECT.

Labels

Labels use a special name format — they must end in a colon. A label may occur in a separate line preceding the destination, or it may precede a command on the same line. Labels are used only with the **goto** command word. Each **goto** statement specifies a label as a destination. Because labels are local to their procedure or function blocks, you may use the same name in another procedure for a different label. For example:

```
proc ask1
  Label1:
  sdlgmsgbox "Message" "This is displayed." INFORMATION OK
endproc
proc ask2
 label1: sdlgmsgbox "Message" "And so is this." INFORMATION OK
endproc
```

Certain ASPECT commands are invalid as label jump locations, and may not be used as targets for a **goto** command. These commands include **integer**, **float**, **long**, **string**, **param**, **case**, and **default**. Labels are also invalid within **dialogbox** command groups.

Let’s continue progressing through the elements of ASPECT with a look at “*Macros*.”

Macros

ASPECT supports macro definitions with or without parameters. When a macro is processed, the ASPECT *Compiler* replaces it with the body of the macro definition. Any argument list is also replaced. This is called macro expansion. The ASPECT *Compiler* then continues with the first token found in the macro body.

The name of a macro must be specified using the same rules followed for all identifiers. However, macro names are handled differently by the ASPECT *Compiler,* and there is no restriction on using the name of a reserved word or user-defined name, except for other macro names.

When a macro is defined without parameters, the macro body begins immediately after the macro name:

#define MACRO_NO_ARGS "This is the macro body."

A macro with parameters is defined by following the macro name with a left parentheses:

#define MACRO_ARGS(x,y,z) x = y * z

A macro can have between 0 and 12 parameters in a comma-separated list. The parameters must follow the standard ASPECT naming conventions, and the list must terminate with a right parentheses. A parameter name appearing within the macro body will be replaced with the argument given in the ASPECT source line using the macro during expansion, but the macro body does not have to contain every parameter in the macro definition.

For example, with user-defined integer variables *product*, *mult1*, and *mult2* declared either globally or locally, the statement:

MACRO_ARGS(product, mult1, mult2)

would become:

product = mult1 * mult2

when expanded by the ASPECT *Compiler*.

The body of a macro definition can be up to 254 characters in length, and a macro can expand into a multiple command line argument, using the “#” character as an end-of-line marker within a macro body. For example, a multi-line macro could be defined as:

#define XMIT(a,b,c) transmit a # transmit b # transmit c

or, using the line continuation character:

**#define XMIT(a,b,c) **

transmit a #

transmit b #

transmit c

Either of these forms expand into three separate ASPECT command lines when processed by the ASPECT *Compiler*:

transmit a

transmit b

transmit c

The ASPECT *Compiler* issues a warning message if a macro is invoked with too many or too few arguments. If too few arguments are specified and the missing arguments occur within the macro body, no argument substitution will be done. If too many arguments are specified, the overflow will be ignored in the macro expansion. The argument list must be comma-separated. Parentheses may be used within the argument list if they are properly matched and nested, and commas may be used within parentheses defining a single argument. Each macro argument can be up to 258 characters long.

After the ASPECT *Compiler* has substituted all of the arguments within the macro, the macro is said to be fully-expanded. The total number of characters of a fully-expanded macro cannot exceed 1024.

If any token in the macro expansion matches the name of a defined macro, it will also be expanded. However, macros are not expanded recursively. Thus if the name of a macro is found within its own body, or through a set of macro expansions, it will not be expanded again and the name will be treated as a normal identifier.

Macros are very useful for making code easier to read. For example, most programmers would find using:

```
#define MIN(x,y) (( x > y ) ? y : x)
  least = MIN(int1, int2); Least equals lesser of int1,int2
```

easier to understand than:

```
least = ( int1 > int2 ) ? int2 : int1
```

when reading the script source. Macros can also simplify global changes for constant values. It is much easier to change the value once within one **#define** statement than to scan through several ASPECT files searching for and changing many occurrences.

ASPECT Operators

Operators are used in mathematical expressions and string variable assignments within the ASPECT programming language. Each operator has a ***precedence*** level and ***associativity***.

Precedence defines the order of operator evaluation in the absence of parentheses. Operators of higher precedence are evaluated before those with lower precedence. Parentheses are used to override the default precedence. For example, the expression A + B * C would be evaluated by multiplying B and C together and then adding A because * has a higher precedence level than +. The expression (A + B) * C, however, would first evaluate A + B, and then would multiply the result by C.

Associativity determines the order of evaluation when examining two operators with equal precedence. All operators having the same precedence level always have the same associativity. Left-to-right associativity means that an operand is grouped with the operator on its left, whereas right-to-left associativity means that an operand is grouped with the operator on its right. For example, the expression A * B / C would be evaluated by multiplying A and B, and then dividing that result by C. Operand B groups with * rather than / because * and / are left-to-right associative. The expression A = B = C, however, is evaluated by assigning the value of C to B and then to A. B is assigned a value before A because = is right-to-left associative.

Two special unary operators, ++ and - -, may be applied to either side of an operand. In either case, the effect is to increment or decrement the operand. However, when the unary operator ++ or - - is applied to the right side of an operand, the value of the operand is used in the context of the expression before it is incremented or decremented. When applied to the left side of an operand, the value used in an expression is the result after the operand is incremented or decremented. For example, in the expression A + B- -, B is added to A, and then B is decremented. In the expression A + - -B, first B is decremented, and then it is added to A.

For clarity, spaces have been inserted between the character pairs in the post-fix decrement(- -) and prefix-decrement(- -) entries. Do not separate these character pairs in your script source, or the ASPECT Compiler will report them as errors!

Compound assignment operators. By using one of these, you can both transform and assign values in a single step. For example, the expression A += B is equivalent to the expression A = A + B.

Except for the left and right parentheses, the unary operators, and the ternary operator, all operators listed in the operator table are binary and act upon two operands.

Ternary operator. Also known as the conditional operator, uses the syntax:

expression ? expression : expression

During execution, the first expression is evaluated. If its value is true, then the expression following the ? character is evaluated. Otherwise, the expression following the : character is evaluated. Ternary operators are often used in assignment statements. For example, in the statement:

```
x = (y < 3) ? 100 : 500
```

x is assigned the value of 100 if *y* is less than 3. If *y* is not less than 3, *x* is assigned the value 500.

The comma. Depending on its context, the comma has different meanings. For instance, it can be used to separate items in a variable definition list. In such a list, the comma doesn't behave like an operator, but simply separates the items from one another.

The comma or sequential evaluation operator has left and right operands when used in an expression. Each operand can also be an expression itself. The operator causes the left operand to be fully evaluated first. The left operand's resulting value is discarded, and the right operand is fully evaluated. The result of the expression is the result of the right operand.

The comma operator occurs most often in **for** loops, where it allows several assignment expressions to be combined into a single expression. For example:

```
for x=0, y=1, z=2 upto 10
  .
  .
endfor
```

Note the "step" variable is the first variable encountered. In this example, the step variable is *x*.

sizeof operator. The **sizeof** operator can be used on any type of operand. It returns, in bytes, the amount of storage required for the operand's **datatype**. The operand can be a single value, the result of an expression, or it can be a **datatype** keyword enclosed within parentheses:

```
sizeof expression
sizeof(datatype)
memalloc 1 (100 * sizeof(integer))
```

The **sizeof** operator can also be applied to an array identifier without its subscript expression. In this case, the result is the size of the entire array, which is the number of elements multiplied by the size of an individual element.

Assignment operator. The simple assignment operator "=" can be used with string operands as well as numeric operands. For example,

```
S1 = "Hello, world!"
```

is a legal ASPECT expression, assigning the value “***Hello, world!***” to the predefined string variable S1.

In the ***Operator Table*** below, precedence level is indicated by bold horizontal lines. Operators that appear within the same pair of bold lines have the same level of precedence. For example, the multiplication (*), division (/) and modulus (%) operators all have the same level of precedence, and they are grouped to indicate this.

The table is also arranged by order of precedence. A group of operators has a higher precedence than those following it in the table. For example, the multiplication (*), division (/) and modulus (%) operators have a higher precedence than addition (+) and subtraction (-).

ASPECT OPERATOR TABLE

Operator	Description	Associativity
( )	Procedure or function call	Left to right
[ ]	Array subscript	
++	Postfix-increment	
- -	Postfix-decrement	
++	Prefix-increment	Right to left
- -	Prefix-decrement	
-	Arithmetic-negation	
!	Logical-negation (logical-NOT)	
~	Bitwise-complement (bitwise-NOT)	
sizeof	Storage amount (bytes) of datatype	
*	Multiplication	Left to right
/	Division	
%	Remainder (modulus)	
+	Addition	Left to right
-	Subtraction	
<<	Bit-wise leftward-shift	Left to right
>>	Bit-wise rightward-shift	
<	Less than	Left to right
<=	Less than or equal to	
>	Greater than	

ASPECT OPERATOR TABLE		
Operator	**Description**	**Associativity**
>=	Greater than or equal to	
==	Equal to	Left to right
!=	Not equal to	
&	Bit-wise AND	Left to right
^	Bit-wise exclusive-OR	Left to right
|	Bit-wise inclusive-OR	Left to right
&&	Logical-AND	Left to right
||	Logical-inclusive-OR	Left to right
? :	Conditional (ternary)	Right to left
=	Simple-assignment (versus compound)	Right to left
*=	Multiplication assignment	
/=	Division assignment	
%=	Remainder assignment	
+=	Addition assignment	
-=	Subtraction assignment	
<<=	Bit-wise leftward-shift assignment	
>>=	Bit-wise rightward-shift assignment	
&=	Bit-wise AND assignment	
^=	Bit-wise exclusive-OR assignment	
|=	Bit-wise OR assignment	
,	Sequential evaluation	Left to right

*Remember that you cannot use bit-wise operators on **float** type variables.*

ASPECT Conventions

Throughout our discussions of ASPECT, the following conventions are used to describe ASPECT commands and their parameters:

[ ]	Any parameter enclosed in square brackets ([ ]) is optional.
{ }	Curly braces indicate two or more parameters which are required when several choices are available.
|	If multiple parameters are separated from each other by a vertical bar (|), choose only a single parameter. The vertical bar indicates "or."
" "	Quotation marks should be entered wherever shown.
...	You may insert any number of parameters, usually up to 12, in the location marked by an ellipsis (...).
. . .	Three vertical dots indicate any number of command statements in a command block. Examples of commands that use command blocks are the **proc** and **while** statements.
arglist	A series of up to 12 variables, numbers, or strings separated by commas.
argument	A data item which is passed as input to an ASPECT command or a user-defined procedure or function.
baudrate	A **long** representing a valid communications baud rate.
character	An integer representing an ASCII character value between 0 and 255. Control characters range from 0 and 31, while displayable characters are usually considered to range from 32 to 126.
column	A vertical column on the *Terminal display*, specified either as an integer constant or variable. The maximum value of column is equal to the number of columns on the terminal display minus 1. Column 0 is always the left-most column.
condition	An expression with a numeric result, or any single ASPECT command which sets the system variables SUCCESS and FAILURE. The NOT keyword may precede an ASPECT command to perform a logical negation of the command result. The NOT keyword cannot precede a user-defined function call, a variable, or a numeric expression. Instead, the "!" ASPECT operator is used to logically negate the result of a user-defined function call or a numeric expression. A condition evaluates true for any nonzero result, and false if the result is zero.

data	A variable or constant of any data type.
datatype	A data type keyword. Either **string**, **integer**, **long,** or **float** may be specified.
datavar	A variable of any data type.
dialclass	A keyword representing a *Connection Directory* entry class. Valid keywords are: DATA, FAX, FTP, MAIL, NEWS, TELNET, VOICE, WWW.
disk	An integer specifying a disk drive within the current command. A value of 0 represents the current drive specified in the ASPECT User Path: 1 represents drive A, 2 represents drive B, up to 26 for drive Z.
expression	A list of one or more operands, each separated by an operator. The result of an expression is a single operand of either **string** or one of the numeric types. Expressions resulting in a string cannot be used within commands such as **if** or **while**.
filelength	A long value, representing the length of a file, or the length of some portion of a file.
filename	A string containing a single valid filename. The pathname is omitted.
fileoffset	A **long** value representing a location within a file.
filespec	A string containing a valid filename. The pathname is optional.
float	A floating point variable or constant.
floatcon	Any float constant.
floatvar	Any **float** variable.
formatstr	A string containing text and/or data format specifiers. For more information on the *formatstr* structure, please see "*Formatting Text and Data*" on page 49.
gdatavar	A global variable of any data type.
gfloatvar	A global **float** variable.
gintvar	A global **integer** variable.
glongvar	A global **long** variable.
gstrvar	A global **string** variable.

ID	A user-specified integer representing a unique identification value.
index	An integer representing a value from a predefined range, or a selection from an ordered list.
intcon	An integer constant, not a variable.
integer	An integer variable or constant.
intvar	Any **integer** variable.
itemlist	A string containing a comma-separated list of items.
itemtype	A keyword or keyword combination that represents an option set item type. Valid keywords are: PROTOCOL, TERMCOLORS, TERMFONT, TERMINAL, WINCOLORS, MODEM CONNECTION, DATA OPTIONS, FAX OPTIONS, FTP OPTIONS, MAIL OPTIONS, NEWS OPTIONS, and TELNET OPTIONS.
keyval	An integer value representing both a key and the current keyboard state. It uses the format 0xAABB, where AA is a two-digit hexadecimal number identifying the keyboard state, and BB is a virtual key value. The keyboard state is encoded by adding any combination of state values: NONE : 0x00, meaning no additional keys are pressed. SHIFT : 0x01, meaning the <Shift> key is also pressed. CTRL : 0x02, meaning the <Ctrl> key is also pressed. ALT : 0x04, meaning the <Alt> key is also pressed. EXTENDED : 0x08, meaning the value is also an extended key. CAPS LOCK : 0x10, meaning that <Caps Lock> is toggled on. For example, ALTSHIFT is 4 + 1 or 5, so the key press <Alt> <Shift><S> is identified as 0x0533 because the virtual key code for 3 is 0x33. The letter 'A' is identified as <Shift><A> or 0x0141, whereas 'a' is 0x0041. The EXTENDED value does not represent an actual key being pressed, but rather identifies the virtual key value as a special type of key such as a function key or a key on the numeric keypad. In order to make it easier to reference keystrokes within ASPECT, we've included **vkeys.inc**. This file is located, by default, within the **\aspect** directory of Procomm Plus. It contains macro definitions of standard virtual key code values.
keyword	A predefined name used by ASPECT to identify a command or command parameter. For a full list of ASPECT keywords, see "*ASPECT Reserved Words*" on page 585.
long	Any **long** constant or variable.

longcon	Any **long** constant.
longvar	Any **long** variable.
memindex	An integer representing a location within a memory block allocated by an ASPECT script. This is a zero based index.
memlength	An integer representing the number of characters to process in a memory block command such as **memread**.
name	An unquoted character string containing the name of a procedure, function macro, label or variable. For more information, see "*Naming Elements in ASPECT*" on page 26.
number	Any integer, long or floating point constant or variable. No spaces are allowed between the characters in a numeric constant.
numvar	Any **integer**, **long** or **float** variable.
operand	A data item which an operator can act upon, or which is the result of another operation.
operator	A symbol for an operation which is performed on one or more operands. For a complete list, see "*ASPECT Operators*" on page 37.
pathname	A string containing a full or partial path name. All file and pathnames returned by ASPECT are in lowercase.
protocol	A valid protocol name, for example, as used by the **sendfile** and **getfile** commands. A complete list of valid protocol names appears in "*Protocol Names and Indices*" on page 59.
row	A horizontal row on the *Terminal* display, specified as an integer constant or variable. The maximum value of *row* is equal to the number of rows on the terminal display minus 1. Row 0 is always the top row of the *Terminal* display.
strcon	A string constant; a group of characters surrounded by double quotes. A quoted string constant can be up to 256 characters in length, and is surrounded by quotation marks, as in "Hello."
strindex	A zero-based integer specifying a character position in a string. Possible values for *strindex* range from 0 to 255.
string	Any quoted string constant or variable.

strlength	An integer specifying a maximum string length. Possible values for *strlength* range from 0 to 256.
strvar	Any **string** variable.
sysvar	Any system variable. For a complete list of system variables, "*System Variables*" on page 427.
task	An integer value representing an application's task ID.
terminal	A valid terminal emulation name. A complete list of emulation names appears in "*Emulation Names and Indices*" on page 60.
text	Any valid source line text composed of zero or more tokens.
timeval	A long value containing the date and time stamp in system format — the number of seconds elapsed since midnight (00:00) on January 1, 1970. The time value is already adjusted for the local time zone and daylight savings time if used within that zone. A long value, it can be used in arithmetic operations to determine periods of time between events. The maximum time value, 0xFFFFFFFF, represents the date Sunday, February 7, 2106 06:28:15. Valid time values for file dates and times range from 0x12CEA600 to 0xF48656FF, representing dates from January 1, 1980 00:00:00 through December 31, 2099 23:59:59. Any values outside of the valid range will generate out-of-range errors.
vkey	A virtual key value. For a list of virtual keys, see **vkeys.inc**, an ASPECT source file you can **#include** in your script for easy access to virtual key codes. The file is installed by default in the **...\aspect** directory.
window	An integer value representing a window ID.

Task and Window IDs

ASPECT allows the manipulation of both tasks and windows. However, tasks and windows are completely different entities and must not be confused! Task IDs represent an entire application running under Windows. Each task can have many separate windows which it displays and controls. Window IDs, on the other hand, represent a single window within a specific task.

ASPECT commands which relate to task manipulation usually begin with the word **task** in the command name, while commands which use window IDs typically begin with the letters **win**.

Custom Interface Functions

ASPECT supports two important graphical interface formats: the *User window* and the dialog box.

The *User window* functions allow you to customize the appearance of Procomm Plus's *Data Terminal* or *Telnet* window for the needs of your script. You can:

- Display bitmaps, icons or metafiles.
- Display clickable buttons or bitmaps.
- Re-size or even hide the *Terminal* window, using a bitmap or metafile background.
- Create invisible "hotspots" in the *User window* that the user can click on.
- Create custom objects that are controlled by a custom Dynamic Link Library (DLL).

The dialog box functions allow you to create customized dialog boxes for any purpose. ASPECT dialog boxes support most of the standard Windows dialog controls, including:

- Buttons and icon buttons.
- List boxes and combo boxes, including a special file list box for lists of files within a directory.
- Edit boxes for strings or text files.
- Static and variable text for labels and information, including a special **dirpath** control for displaying the current drive and path.
- Check boxes and option buttons.

ASPECT also supports the use of graphical elements such as icons, bitmaps, metafiles, and group boxes within dialog boxes.

The User Window

A *User window* is defined with the **uwincreate** command, which requires arguments and keywords describing the size, position, color, and behavior of the window. Elements attached to the *User window* are known as *objects*, and are created with the appropriate ASPECT commands following the **uwincreate** statement.

There is a limit of 32 active objects for each object type. Only one DLL object file can be active at a time, but that DLL can handle multiple objects in the *User window*.

User Window Object IDs

Each *User window* object is assigned its own ID value. *User window* object IDs may have any integer value greater than zero. Duplicate IDs are not flagged by the compiler and should be avoided.

The object id is used with the object-related commands such as **objhide** and **objmove** to manipulate the object. The value is also assigned to the $OBJECT system variable when the object has been selected by the user. For more information, see the description of the **uwincreate** command.

User Window Units

User window objects are placed and sized based on *User Window Units*, or UWUs. An UWU is defined as 1/10000 of the horizontal width and 1/10000 of the vertical height of the background graphic. Units based on the size of the background graphic allow graphic displays designed on one screen resolution to be more accurately represented on different screen resolutions.

In most cases, you will not need to worry about calculations involving *User Window Units,* as the *User Window Editor* conveniently handles them for you!

Dialog Boxes

A dialog box is defined by the **dialogbox** command, which requires arguments and keywords specifying the size, placement, and behavior of the dialog. Elements within the dialog are known as *controls,* and are created using the appropriate ASPECT commands within the **dialogbox** statement group.

Up to 255 controls can be used within a single dialog box.

Dialog Box Control IDs

Each dialog box control is assigned a unique ID value. The dialog box control ID must be an integer value from 1 up to 999. The ASPECT *Compiler* will detect duplicate control IDs and report them as errors.

Control IDs are used to manipulate their associated controls. They are also used to detect when those controls have been selected or modified by the user. For more information, see the description of the **dialogbox** command.

Dialog Box Units

When a dialog box is created, its overall size and the position and size of the controls within it are determined in *Dialog Box Units*, or DBUs. The DBU is based on the size of the current system font. A vertical DBU is 1/8 of the height of the system font. A horizontal DBU is 1/4 of the average width of the system font.

In most cases, you will not need to worry about calculating *Dialog Box Units*, as the *Dialog Editor* does this for you automatically!

Caret Translation

Certain special ASPECT commands perform caret “^” translation on the contents of their string arguments. Unlike the escape sequence processing, which is performed by the ASPECT *Compiler* when the **.was** file is compiled into a **.wax** file, caret translation occurs at run-time as the string argument is passed to the ASPECT command.

If a caret, ASCII 94, is followed by a character whose value is from ASCII 64 (“@”) to 95 (“_”), a single character value is substituted within a value from 0 to 31. For example, **^@** is converted to a character value of 0, ***^A*** is converted to value 1, and so forth up to **^_** (caret underscore) which is converted to value 31.

If a caret is followed by a character whose value is within the range 97 (“a”) through 122 (“z”), a single character value is substituted with a value from 1 to 26. An ***^a*** is converted to value 1, a ***^b*** to value 2, and so forth up to ***^z***, which is converted to 26.

If a caret is followed by a vertical bar character, “|,” ASCII 124, then the two characters are converted to a single caret character, and the vertical bar is ignored.

If a caret is followed by any character other than those previously discussed, then no translation occurs, and the caret is treated like any other character.

ASPECT commands that perform caret translation are documented individually, but typically include those commands that write data to the terminal or provide strings to be tested against incoming data. For example, the **transmit** command writes the contents of a string to the currently active communication port. The command:

```
transmit "ATDT1-573-875-0503^M"
```

would write “***ATDT1-573-875-0503***” without quotes, to the active port, followed by a carriage return character, ASCII 13 or ^M.

Formatting Text and Data

The format string, *formatstr* consists of zero or more characters of literal text with zero or more format specifiers. For example, in the statement:

```
strfmt s0 "This is literal text."
```

the *formatstr* ***"This is literal text."*** performs no special formatting at all. It merely assigns the quoted string, "***This is literal text.***" to the predefined global variable S0.

However, the *formatstr* may also contain an assortment of format specifiers that correspond with optional variables or constants (parameters) following the format string. Format specifiers and their corresponding parameters are processed from left to right. A format specifier is constructed as follows:

%[flags][width][.precision]type

where:

Specifier	Description
%	Indicates that this is the beginning of a format specifier, rather than normal text. Characters following this special marker describe the format for displaying the corresponding param. If the % sign is followed by a character that has no special meaning as a format specifier, then that character and any subsequent characters until the next % are simply displayed as normal text. To display a percent sign, use "%%".
flags	Valid flags are "-," "+," " " (a space), and "#." These flags are optional; zero or more flags may be specified.
"-"	Left-justifies the converted parameter within the number of places indicated in the width field. Without this optional flag, the displayed text is right-justified.
"+"	Prefixes the converted parameter with a plus or minus sign, if it represents a signed type. Without this flag, the text will be displayed with a sign only if it represents a negative number.
" "	A space immediately after the % marker prefixes the converted parameter with a space if it represents a signed positive value; the space is ignored if both the blank and "+" flags appear.
"#"	Prefixes the converted parameter with the characters "0x" if it represents a hexadecimal value or with "0" if it represents an octal value. In addition, the # flag can be used to force a decimal point for all float conversions even if there is no fractional portion. This flag is ignored if used with any format types other than x, X, o, f, e, E, g, or G. For example:

Specifier	Description
	integer nm1 = 16 **float nm2 = 16** **strfmt s0 "%+d" nm1 ; formats as "+16"** **strfmt s0 "% d" nm1 ; formats as " 16"** **strfmt s0 "%#x" nm1 ; formats as "0x10"** **strfmt s0 "%#f" nm2 ; formats as "16."**
width	This optional value specifies the minimum number of characters that will be generated for the conversion from the corresponding parameter. If the width value is greater than the number of characters in the converted text, it will be padded on the left with spaces, unless this has been altered by one of the flags. A 0 immediately preceding the width field pads the converted text with leading zeros after any sign or prefix, as long as no other padding has been specified. Note that since the width field specifies the minimum number of generated characters, it will never cause a value to be truncated. **integer nm1 = 16** **strfmt s0 "%d" nm1 ; formats as "16"** **strfmt s0 "%05d" nm1 ; formats as "00016"** **strfmt s0 "%-5d" nm1 ; formats as "16 "**
precision	This optional value must always be preceded by a period (**.**). It specifies the minimum number of digits to display when converting an integer parameter to displayed text, the number of decimal places to display on a float conversion using the **e**, **E** , or **f** types, the maximum number of significant digits to display on a **g** or **G** conversion or the maximum number of characters to display on a string conversion using the **s** type. The default precision for floating point values may be specified using the **set aspect decimal** statement. Since precision specifies the maximum number of digits, it may cause truncation of strings or rounding of floats. For example: **string str1 = "This is a string"** **float nm1 = 1234.567** **strfmt s0 "%7.4s" str1 ; formats as " This"** **strfmt s0 "%-*.*f" 8 1 nm1** **; formats as "1234.5 "** The width and/or precision fields may be an asterisk (*), which means that the value is supplied from the parameter list following the format string. In this case, the extra parameter would be an integer type preceding the parameter to be converted.

Specifier	Description
type	This specifier determines the *datatype* required of the corresponding parameter, if any, and how the command will alter the parameter's value before converting it to displayed text. Please note that the following type specifiers are case sensitive; lower case types must be entered in lower case and upper case specifiers must be entered in upper case!
d or i	The parameter is converted to signed decimal notation.
u	The parameter is output as an unsigned decimal number.
o	The parameter is output in octal notation.
x	The parameter is output in hexadecimal notation using the characters 0-9 and a-f.
X	The parameter is output in hexadecimal notation using the characters 0-9 and A-F.
l	A lower case 'L' may be used before the d, i, o, x or X format specifiers to specify that the corresponding argument is a **long** value.
f	The parameter is output in the form [-]dddd.dddd, where dddd is one or more decimal digits. The number of digits displayed after the decimal point is determined by the precision specification. The default precision is 2. A minus sign is displayed if the value is less than zero, but a plus sign is displayed only if called for with the "+" flag.
e	The parameter is output in the form [-]d.dddd e sign ddd, where d is a decimal digit, dddd is one or more decimal digits, ddd is three decimal digits, and the sign is "+" or "-" (scientific notation).
E	Similar to "e," above, except that this type uses a capital E instead of lower case "e" to prefix the exponent.
g	The parameter is output in **f** or **e** format, as appropriate. If the **e** conversion would yield an exponent greater than -4 or less than the specified precision, the parameter value is output in **f** format instead. The generated text has no trailing zeros after the decimal point and includes a decimal point only if the result is not a whole number or if you specify the "#" flag.
G	Same as **g** above, except the **E** output format is used instead of the **e** format.
c	The integer parameter is output as a single ASCII character.

Specifier	Description
s	The **string** parameter is displayed up to the first null character or until the precision value is reached. For example: **integer num1 = 20, num2 = 30** **string string1 = "This is a string"** **float float1 = 1234.567** **strfmt "Print %d and %d" num1 num2** **; formats as "Print 20 and 30"** **strfmt "To %d and %#X" num1 num2** **; formats as "To 20 and 0X1E"** **strfmt "%7.4s" string1 \** **; formats as " This"** **strfmt "%-8.1f" num1 \** **; formats as "1234.5 "**

ASPECT Program Flow

Every ASPECT script program requires a special procedure that identifies the starting point for script execution. This procedure is always named **main**.

```
proc main
  ; ...
endproc
```

When a script is run, the first command line it executes is the first command line listed in the procedure named **main**. Execution flow continues until the end of the **main** procedure, at which time the script is terminated.

Generally, however, the **main** procedure directs execution to other parts of the script, using procedure or function calls. When a **call** command is issued, execution flow transfers to the start of that procedure or function and continues until that procedure or function block ends, or until a **return** command is encountered within the **call**ed procedure or function.

When a **return** command is encountered, or the end of the procedure or function is reached, execution returns to the point from which the procedure or function was originally called. Execution then proceeds with the command following the procedure or function **call**.

Any **call**ed function or procedure may in turn **call** any other function or procedure, even itself. In this way the program execution flow can branch out into all areas of the script.

The **main** procedure cannot define any parameters, since there is no initial **call** made where arguments would typically be passed. There are a number of ways to pass information to a script, however. Usually, this involves the predefined variables that are available to all scripts.

ASPECT Command Form

An ASPECT command is formed by listing the command name followed by a specific number of keywords and/or arguments of the appropriate data type. An argument can be the result of an expression or function **call**. Each keyword or argument must be separated by a comma, at least one space, or a tab character.

If an expression is used to represent a single argument, it is generally easier to read if the entire expression is enclosed in parentheses. In cases where an argument is an expression or constant value that is negated using a minus sign, a comma or parentheses must be used so that the negated value is not combined with any preceding arguments by the compiler. For example, the command **fseek 0 -1 2** is intended to position the file pointer one character before the end of the file. However, the 0 and -1 arguments are seen by the compiler as a single expression, **0-1**, with a value of -1, and the compiler reports syntax errors. To be evaluated correctly:

```
fseek 0 (-1) 2
```

or

```
fseek 0, -1, 2
```

should be used. Similar problems can occur when using the increment and decrement operators, ++ and - -.

The use of commas to separate adjacent arguments is encouraged. ASPECT is a growing language, and future versions may require a stricter syntax to provide enhanced features.

The ASPECT Script Environment

The ASPECT script environment defines how a script interacts with actions, events, and other features in Procomm Plus such as keystrokes typed at the *Terminal window* or data received through the current connection. It also determines how the script interprets its own commands and how it controls the operation of other scripts it may launch.

ASPECT allows a script to define much of its own environment including global variables, **when** events, and dialog boxes for user interaction. These are all owned solely by the executing script and cannot be controlled by another script. Other environmental changes such as custom menus, the *User window*, and packet transfer mode are not “tied” directly to the executing script, although interpreting these events from a second script would be difficult without some kind of understanding between the two scripts.

The critical environment issues for a script that is launched from another script are in regard to those settings that control how that script’s commands behave, and how the script interacts with external events. These "global" environment settings are referred to as the ASPECT environment.

The ASPECT environment consists of the User Path, the Aspect Path, and the settings affected by the **set aspect** command group.

An ASPECT script maintains its own current disk drive and current working directory. Together, these are referred to as the User Path. The User Path is maintained independent of any changes to the current drive and/or directory that are made outside of ASPECT.

The User Path is used as the default path for directory and file-related commands such as **diskfree**, **findfirst** , **getdir**, **getvolume**, **mkdir**, **rename**, and so forth. The User Path can be changed using the **chdir** command.

An ASPECT script also maintains its own default path, called the Aspect Path, which is separate and distinct from the User Path. The Aspect Path is referenced for dialog box and *User window* commands. When a filename without a path is specified in a *filespec* argument for one of these commands, ASPECT will look for the target file in the Aspect Path directory. The Aspect Path is also assumed for the **chain**, **compile**, **dllload**, **dllobjfile**, **execute**, and **set aspect helpfile** commands. It can be changed with the **set aspect path** command, or as an optional action in the **chdir** command.

Both the Aspect Path and the User Path are used in resolving partial path names. In general, a *filespec* with a partial path name will be evaluated based on the path normally associated with that command. No path resolution is performed for full path names or for path names that contain a drive designator. If a path name specifies only a drive designator, the current working directory for that drive is assumed.

When a script is executed, and no other script is currently running, it is assigned the default ASPECT environment settings. Included in this process are the normal script start-up functions: setting both the Aspect *Path and the User Path to the path containing the script's* ***.wax*** *file, and initializing the* ***set*** *ASPECT command values to their current defaults.*

Script Spawning and Chaining

One ASPECT script can be used to start another ASPECT script within the same instance of Procomm Plus in two different ways, referred to as ***spawning*** and ***chaining***. The calling script is referred to as the ***parent*** script, while the called script is referred to as the ***child***. The difference between spawning and chaining is quickly summarized by the fact that a spawned script, upon termination, returns control to the parent; a chained script does not.

When a child script is spawned using the **execute** command, ASPECT stores the current state of the parent script in a temporary task file. Upon return from the child script, the task information is restored from this file, and the parent script resumes execution from the point where the child was launched.

In addition to saving the current location within the parent script, the task file contains a record of all user-defined global variables, all active **when** commands, all files opened by ASPECT's **fopen** command, all active **setjmp**/**longjmp** commands, and all allocated memory blocks and their contents.

The task file does *not* contain information about any active ASPECT dialog boxes, any DDE client conversations which were in effect, the state of the predefined global variables, the current state of the **set** commands with the possible exception of those comprising the ASPECT environment, the current state of the *User window*, the state of the current menu items, or the state of packet transfer mode. However, certain system variables, such as those specific to the script's execution context, may be restored.

When a script passes execution to another script using the **chain** command, the parent script is terminated, and will not be restored when the child is finished. No task file is created for a **chain**ed script.

When a script spawns or chains to another script, all existing ASPECT dialog boxes in the current script will be destroyed, as will any DDE client conversations currently in effect. The child script will inherit the current values of the predefined variables and any system variables that are not script-specific, as well as the state of all **set** commands that are not part of the ASPECT environment. In addition, the child will inherit the current *User window* and all of its objects, any menus created by ASPECT, the state of all menu items, and the current state of packet transfer mode.

The script that performs the **chain** or **execute** command can determine whether the chained or spawned script will share the same current ASPECT environment settings, or whether it will be assigned the default ASPECT environment settings. This is controlled with the SHARED keyword in the **chain** or **execute** command.

If a child is spawned using **execute**, and the environment is SHARED, then when the child script terminates and control returns to the parent script, any changes to the ASPECT environment are carried back to the parent script's environment. If the environment was not SHARED with the child script, the parent will resume execution with its own ASPECT environment which had been saved in the task file. Note that, depending on how the ASPECT environment is SHARED, if either the child or the parent script was responsible for keyboard data or received data, and some of that data had not yet been processed, then it is possible that the unprocessed keyboard data will be lost, and the unprocessed received data will be processed by the *Terminal* rather than by the script that assumes execution.

If a script file is launched on top of another running script, it will be initialized with the default ASPECT environment settings. This situation can arise when a currently running script has **set aspect spawn** ON, which enables other scripts to interrupt it at any time. If it is interrupted, its current execution state will be saved in a task file.

Setting **aspect spawn** ON can be dangerous to a script’s proper execution if that script has dialogs or DDE commands that are active, since these are destroyed when the script is temporarily shut down. Typically, this flag should be set on only when a script can be freely interrupted and resumed without affecting its intended purpose.

The currently running script task may need to know if it was spawned or chained so that it can check for relevant data in the predefined variable set, or determine something about the ASPECT environment that was assigned to it. The system variable $SCRIPTMODE can be referenced to determine how any script was executed. If a script was spawned, the contents of the system variable $PARENTFILE will contain the name of the parent script. If the script was chained, the system variable $CHAINEDFILE will contain the name of the script which issued the **chain** command.

If the script was launched from a *Connection Directory* entry, the $DIALENTRY system variable will contain the entry name. By testing the values of $DIALCONNECT and $DIALING, the script can determine if a **dial** command should be issued to begin the dialing sequence for that entry. The $SCRIPTENV system variable allows a script to determine if the ASPECT environment it has is being shared with the script that launched it.

Preprocessor Commands

The ASPECT *Compiler* has a built-in preprocessor. A preprocessor command is an instruction to the compiler to change the way the ASPECT source file is being compiled. It is normally used to make source files simpler to adjust for special conditions, for the inclusion of other source files in the compiled result, or to prevent compilation of certain sections of the source file.

Preprocessor commands begin with a pound sign (#) as the first non-space character on a source line followed by the command word. Spaces or tabs are allowed between the pound sign and the command word. A pound sign appearing alone on an ASPECT source line is called a “null directive” and is treated as a blank line.

Preprocessor commands are recognized before macro expansion occurs. If a macro expands into something that appears as a preprocessor command, it will not be recognized as a preprocessor command, and it will generate a compile-time error.

The conditional compilation commands, **#ifdef**, **#ifndef**, **#elifdef** and **#elifndef** test for the existence of a defined macro name. The macro name should be specified without arguments, even if it was defined with a parameter list.

The **#if** and **#elif** preprocessor commands are conditional compilation commands as well. However, unlike the previous, these commands evaluate a constant expression (such as a numeric constant, or an expression composed solely of numeric constants) to determine whether or not the block of code which follows is compiled.

For more information on preprocessor commands, see the descriptions for the **#define** command and the **#ifdef** command.

Predefined Macro Definitions

The ASPECT *Compiler* automatically defines several macro definitions that are accessible during compilation: ASPVERSION, ASPDEBUG, ASPFILE, and ASPLINE. Predefined macros cannot be undefined using **#undef.**

ASPVERSION is defined as the current ASPECT *Compiler* version ID. A single number is defined where the hundreds digit represents the major version number, and the tens and ones digits represent the minor version number. For instance, version 4.00 is stored as the value 400 in ASPVERSION. This macro is useful for ASPECT scripts that can be compiled and run on several different versions of Procomm Plus.

ASPDEBUG is defined with a value of 1 whenever a script is being compiled in debug mode (or with the /Z command-line switch). This macro definition can help you write blocks of code that should be executed only when debugging their scripts. ASPDEBUG is not defined when not compiling in debug mode, thus the preprocessor command **#ifdef** would typically be used to test for its existence.

ASPFILE is defined with the current source file name that is being compiled. Its definition will change with each included source file.

ASPLINE is defined with the current source file line that is being compiled. It uses a **long** data type.

The combination of ASPFILE and ASPLINE are useful when writing debugging code within a script. For instance, you could write code to test a condition upon entry to a particular procedure:

```
;Assertion Debugging Technique
#ifdef ASPDEBUG
  #define ASSERT(condition) \
  if !(condition) #\
    errormsg "Assertion Failed: File: %s Line: %lu" \
    ASPFILE ASPLINE #\
  endif
#else
  #define ASSERT(condition)
#endif
proc Main
  usermsg "Test assertion with value 0"
  testassert(0)
  usermsg "Test assertion with value 100"
  testassert(100)
endproc
proc TestASSERT
param integer val
  ASSERT(val < 100)             ; test entry condition
  ; insert other code here
  ASSERT((val > 0) && (val < 1000))   ; test exit condition
endproc
```

Protocol Names and Indices

The **set** commands, **getfile**, **sendfile**, and the **item**-related commands use reserved names or indices to reference the protocols supported by Procomm Plus.

The protocol names and their corresponding indices are:

Protocol Name and Index Table			
Protocol Name	Index	Getfile Filespec	Sendfile Filespec
ZMODEM	0	Not required	Required
KERMIT	1	Not required	Required
XMODEM	2	Required	Required
1KXMODEM	3	Required	Required
1KXMODEMG	4	Required	Required
YMODEM	5	Not required	Required
YMODEMG	6	Not required	Required

Protocol Name and Index Table			
Protocol Name	Index	Getfile Filespec	Sendfile Filespec
CISB	7	Not required	Not required
ASCII	8	Required	Required
RAWASCII	9	Required	Required
IND$FILE	n/a[a]	Two required	Two required

a.The Ind$file protocol is shipped as a **.dlt**, meaning that it does not have an associated index value.

For further information on protocol use, please refer to the descriptions for the **getfile** and the **sendfile** commands**.**

*Emulations and protocols that are shipped with Procomm Plus as Dynamic Linked Libraries do not have associated index values. For convenience, ASPECT provides reserved keyword names for these **.dlt** (Dynamic Linked Terminal) and **.dlp** (Dynamic Linked Protocol) modules such as the RIPscrip terminal and the Ind$file file transfer protocol. For best results, you should use the module's keyword or string name whenever a protocol or emulation name is required.*

Data Terminals are specified in a similar fashion as we see in "*Emulation Names and Indices*."

Emulation Names and Indices

The **set** and **itemcreate** commands use keyword names, string names, or indices to reference the emulations supported by Procomm Plus. The emulation names and their corresponding indices are:

Emulation Name and Index Table			
Name	Index	Name	Index
ADDS60	0	TTY	17
ADDS90	1	TV1910	18
ADM31	2	TV1912	19
ADM3A	3	TV1920	20

Emulation Name and Index Table			
Name	Index	Name	Index
ADM5	4	TV1922	21
ANSIBBS	5	TV1925	22
ATT4410	6	TV1955	23
ATT605	7	TVI955	24
DGD100	8	VIDTEXT	25
DGD200	9	VT52	26
DGD210	10	VT100	27
ESPRIT3	11	VT102	28
HEATH19	12	VT220	29
IBM3101	13	VT320	30
IBM3161	14	WYSE 50	31
IBM3270	15	WYSE60	32
IBMPC	16	WYSE75	33
RIP	n/a[a]	WYSE100	34

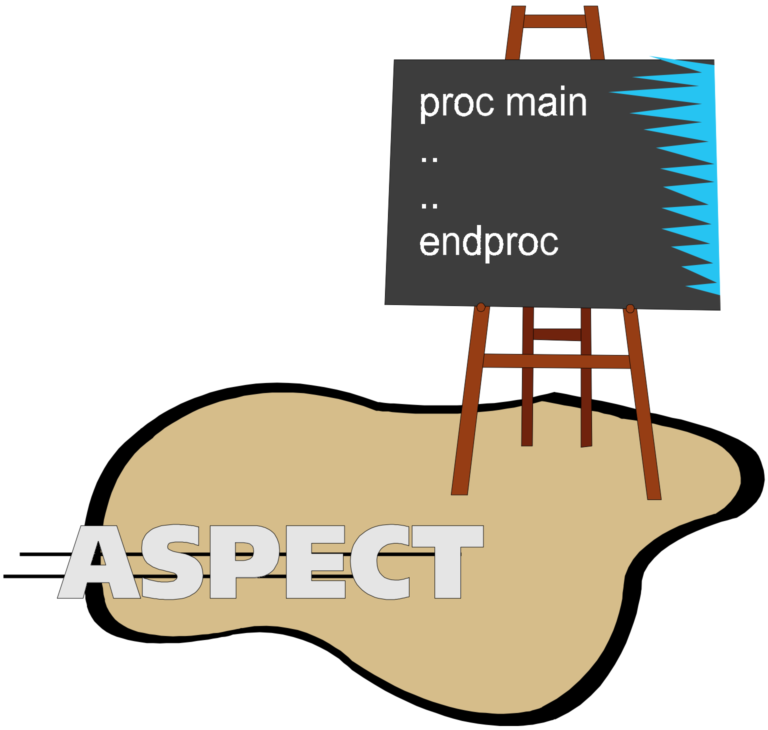
a.The RIPscrip emulation is shipped as a **.dlp**, meaning that it does not have an associated index value.

"*Set and Fetch Statements*" on page 363 contains additional information regarding the use of these indexes.

The following section, "*The ASPECT Commands*", details of each of the ASPECT commands.

Chapter 3

The ASPECT Commands



Introduction

To detail the *ASPECT* commands, this discussion is divided into the following topics:

- A list of the ASPECT commands grouped by function.
- "*Using Command Descriptions*" on page 88, which demonstrates the"*command*"description layout used throughout this discussion.
- A detailed explanation of each of the ASPECT commands, presented in alphabetical order.

Commands Listed by Function

ASPECT Primary Commands

call	Causes script execution to branch to a specified procedure or function and permits a return.
case | **endcase**	Declares a string or integer to be tested against a **switch** statement's target variable.
default	Provides for processing when no match is found among the **case** statements in a **switch** statement group.
else	Executes an alternate set of commands when the expressions specified in the associated **if** or **elseif** commands evaluate as false.
elseif	Enables multiple tests within an **if/endif** block.
exitfor	Transfers control from a **for** statement group to the line following the **endfor** command.
exitswitch	Transfers control from a **case** or **default** statement group to the line following the **endswitch** command.
exitwhile	Transfers control from a **while** statement group to the line following the **endwhile** command.
float	Defines a global or local float variable, or an array of float variables.

for | **endfor**	Repeats a command or series of commands a specified number of times.
func | **endfunc**	Denotes a function block.
goto	Performs an unconditional branch to the specified label within the current procedure or function block.
if | **endif**	Denotes a block of commands which are executed if the condition associated with the **if** command is true.
integer	Defines a global or local integer variable, or an array of integer variables.
long	Defines a global or local long variable, or an array of long variables.
loopfor	Skips the remaining commands in a **for** loop and branches to the top of the loop.
loopwhile	Skips the remaining commands in a **while** loop and branches to the top of the loop.
param	Defines a parameter variable in a procedure or function.
proc | **endproc**	Denotes a procedure block.
return	Exits the current procedure or function and resumes processing at the statement following the function or procedure call.
string	Defines a global or local string variable, or an array of string variables.
switch | **endswitch**	Denotes a block of multiple decision points.
while | **endwhile**	Denotes a block of repeated commands.

Clipboard Commands

cliptofile	Copies data from the Windows Clipboard to a file.
cliptostr	Copies text from the Windows Clipboard to a string.
filetoclip	Retrieves the contents of a file and places the data in the Windows Clipboard.

pastetext	Sends text in the Windows Clipboard to the active COM port.
strtoclip	Copies information from a string to the Windows Clipboard.

Com-Related Commands

break	Sends a break to a remote computer system.
clearxoff	Clears an XOFF state.
comgetc	Assigns the next character value in the receive data buffer to an integer variable.
computc	Sends the specified character value to the communications port.
comread	Retrieves data from the receive data buffer and stores it in the target variable.
comwrite	Sends the specified string value to the communications port.
hangup	Disconnects your computer from the telephone line.
rget	Receives and stores a text string sent by a remote system.
rxflush	Clears the receive buffer.
transmit	Sends a character string to the active COM port.
txflush	Clears the transmit data buffer.
waitfor	Pauses script execution until a target string is received from a remote system.
waitquiet	Pauses script execution until the receive data line has been inactive for a specified number of seconds.
when quiet	Forces a procedure or function call when the receive data line has been inactive for a specified number of seconds.
when target	Forces a procedure or function call when a specified string is received from a remote system.

Date and Time Commands

intsltime	Converts integer date and time values into a long system time format.
ltimeelapsed	Converts a *timeval* into a time string.
ltimeints	Converts a long time/date value in system time format into integer variables.
ltimemisc	Returns miscellaneous information about a given *timeval.*
ltimestring	Converts a long system time value into a string variable representing date and time.
ltimestrs	Converts a long system time value into a two string variables representing date and time.
monthstr	Returns the name of the specified month in a string.
strsltime	Converts a date string and a time string into a long variable.
weekdaystr	Returns the name of the day of the week.

Dialing Commands

dial	Calls one or more entries in the *Connection Directory.*
dialadd	Adds a new entry or group to the *Connection Directory.*
dialcancel	Cancels a dialing operation.
dialclass	Returns a value indicating the entry class(es) for the specified entry.
dialcount	Returns the number of entries, groups or entries within a group for a particular dialing class.
dialcreate	Creates a new *Connection Directory* file.
dialdelete	Removes a *Connection Directory* entry, group, or entry within a group.
dialfind	Finds the entry name and/or index associated with a specified name in the *Connection Directory.*

dialinsert	Inserts a *Connection Directory* entry into an existing group.
dialload	Loads a different *Connection Directory.*
dialname	Enumerates entries, groups, or entries within groups.
dialnumber	Calls a specified telephone number.
dialsave	Saves the current *Connection Directory* to disk.
dialstats	Retrieves information about a specified entry.

Dialog Box Commands

bitmap	Places a bitmap graphic in a dialog box.
checkbox	Adds a check box control to a dialog box.
combobox	Adds a combination edit/selection, drop-down list box control to a script dialog box.
dialogbox / **enddialog**	Denotes a dialog box statement group.
dirlistbox	Adds a file selection list box to a script dialog.
dirpath	Displays the current disk drive and directory for a **dirlistbox** control in a dialog box.
dlgctrlwin	Returns the window id of a dialog box control.
dlgdestroy	Destroys a dialog box.
dlgevent	Returns the ID of a user action within a dialog box, or flushes the event queue for the specified dialog box.
dlgexists	Sets SUCCESS if the dialog box with the associated ID exists.
dlglist	Adds, removes, counts, retrieves, or finds items from a list box, file list box, combo box, or file combo box.
dlgsave	Updates variables associated with dialog box controls and forces changes in file combo boxes, edit boxes, and list boxes to be written to disk.
dlgshow	Causes a hidden dialog box to be visible.

dlgupdate	Refreshes the display of a dialog box control.
dlgwin	Returns the window ID of the specified dialog box.
dlgwinctrl	Converts a window ID into the corresponding control ID within a given dialog.
editbox	Adds an editing box for text entry to a script dialog.
fcombobox	Adds a combination edit/selection or “drop-down” selection control to a script dialog box using a file as the list source.
feditbox	Adds a text editing box with the contents of a disk file in a script dialog box.
flistbox	Adds a list box to a script dialog, using the contents of a disk file for the list source.
ftext	Displays text from a file in a script dialog box.
groupbox	Displays a rectangle with an optional text label in a script dialog box.
icon	Displays an icon graphic in a script dialog box.
iconbutton	Displays an icon graphic as a clickable button in a script dialog box.
listbox	Adds a list box to a script dialog box.
metafile	Displays a metafile graphic in a script dialog box.
pushbutton	Places a standard button control in a script dialog box.
radiobutton	Creates or appends to an option button command group in a script dialog box.
radiogroup / endgroup	Denotes an option button command group.
text	Displays a text string in a script dialog box.

DOS- or Disk-Related Commands

addfilename	Adds a specified filename to a path.
chdir	Changes the User Path to the specified drive and/or directory.

copyfile	Copies a file to another file or path.
delfile	Deletes a specified file.
dir	Displays a standard file listing and optionally returns a selected filename.
diskfree	Returns the free disk space of the specified drive into a long variable.
dos	Executes a DOS command or another program within a separate DOS window.
fileget	Reports date, time, attributes, and size for a given *filespec*.
fileset	Sets date, time, attributes, and size for a given *filespec*.
fileview	Displays a file in a modal dialog box.
findfirst	Locates a disk file using a specification you provide.
findnext	Locates additional disk files with the specification provided in a previously-executed **findfirst** command.
fullpath	Returns the fully-qualified drive and path for a specified *filespec*.
getdir	Returns the current working directory and/or path of the specified drive into a string variable.
getenv	Returns the contents of a DOS environment variable definition.
getfilename	Extracts a filename from a *filespec*.
getpathname	Extracts a path name from a *filespec*.
getvolume	Returns the volume label for the specified drive.
isfile	Reports whether a file exists.
makepath	Creates a path and/or *filespec* from individual components.
mkdir	Creates a new directory using a path and/or directory name you provide.
putenv	Adds or changes a DOS environment variable definition.
rename	Renames an existing file.

rmdir	Removes an existing directory using a specified path.
run	Executes an external program in a separate window.
shell	Displays the DOS command prompt in a separate window or full screen.
splitpath	Splits a file and path name.

Dynamic Data Exchange Commands

ddeadvise	Allows Procomm Plus to request the value of a variable from a Windows DDE server.
ddeexecute	Executes a command in a Windows DDE server application.
ddeinit	Establishes a DDE data channel between Procomm Plus and a Windows DDE server application.
ddepoke	Assigns data from ASPECT to a Windows DDE server application.
dderequest	Receives the current value of a variable from a Windows DDE server application.
ddeterminate	Terminates an existing DDE data channel.
ddeunadvise	Terminates a current **ddeadvise** condition.

Fax Commands

faxcancel	Cancels the current fax operation.
faxlist	Enumerates the faxes displayed in the received or scheduled fax lists.
faxmodem	Determines whether a modem is fax-capable.
faxpoll	Dials a *Connection Directory* entry or group, or a specified number to receive faxes from a host.
faxprint	Prints a specified fax file.
faxremove	Removes a fax file from the received or scheduled fax list.

faxsend	Dials the specified number, *Connection Directory* entry, or group and transmits a specified fax file.
faxview	Displays a fax file.

File I/O Commands

fclear	Clears all end-of-file and error flags associated with the specified file ID.
fclose	Closes the file corresponding to the specified file ID.
fdelblock	Deletes a block of data from the specified file.
feof	Tests for end-of-file condition on the specified file ID.
ferror	Tests for any error condition on the specified file ID.
fflush	Writes the current I/O buffer contents to the specified file ID.
fgetc	Reads a character from the specified file ID into a variable.
fgets	Reads a string from the specified file ID into a variable.
finsblock	Inserts a block of space into the specified file ID.
flength	Returns the file size of an open file.
fopen	Opens a file in the indicated mode and assigns it to a file ID.
fputc	Writes a character value to the specified file ID.
fputs	Writes a string to the specified file ID.
fread	Reads a block of data from a file into a variable and returns the number of bytes read.
fseek	Repositions the file pointer for the specified file ID.
fstrfmt	Similar to **strfmt**, this command writes a formatted string to the specified file ID.
ftell	Returns the current file pointer position for the specified file ID.
ftruncate	Truncates or clips the file corresponding to the specified file ID at the current file position.

fwrite	Writes a block of data to a file.
rewind	Repositions the file pointer of the specified file ID to the beginning of the file.

File Transfer Commands

ftp	Allows file manipulation on either host or local FTP machines.
getfile	Receives or downloads a file from a remote computer using the specified transfer protocol.
kermserve	Issues a Kermit server command.
sendfile	Sends or uploads a file to a remote computer using the specified transfer protocol.
xfercancel	Cancels a current file transfer operation.

General Procomm Plus Commands

alarm	Sounds an alarm to signal an event.
beep	Sounds a beep tone.
capture	Controls session capture during script file execution.
capturestr	Writes a specified string to an open *Capture* file.
crc16	Generates a CRC integer value for a specified string.
decrypt	Decodes an encrypted string.
encrypt	Encodes a target string.
errormsg	Displays a message box with a graphic and specified text.
fetch	Returns the current value(s) of any **set** command parameter.
help	Displays on-line *Help.*
mapisend	Initiates a send mail action if the MAPI interface is loaded and accessible.

metakey	Executes the specified *Meta Key*.
mspause	Pauses script execution for the specified number of milliseconds.
pause	Halts script execution for the specified number of seconds.
playback	Plays back a *Capture* file or stops a **playback** currently in progress.
pwexit	Terminates the executing script file, disconnects if necessary, then exits Procomm Plus.
pwmode	Places the communication window into one of six execution modes.
pwtitlebar	Specifies the text to display in the Procomm Plus program title bar.
sdlgfopen	Displays a standard file open dialog.
sdlginput	Displays a standard dialog box, allowing the user to enter a single line of text.
sdlgmsgbox	Displays a standard message box with a specified icon and button.
sdlgsaveas	Displays a standard file save as dialog box.
set	Changes system parameters which control various operations within Procomm Plus and ASPECT.
setpointer	Moves the mouse pointer to the specified X/Y screen coordinates.
setup	Manipulates the Procomm Plus *Setup* utility.
statclear	Clears the status line in the *Terminal* window.
statmsg	Displays a formatted message on the status line in the *Terminal* window.
usermsg	Displays a message string in a dialog.
waitquiet	Pauses script processing until the specified date and/or time.
when	Forces a procedure or function call when a particular event occurs.
xlatin	Converts a character or string using the incoming *Translate Table* values.
xlatout	Converts a character or string using the outgoing *Translate Table* values.

xlatstr	Performs caret translation on a string.
wizard	Activates a Procomm Plus wizard.

General Windows Commands

disable	Disables controls, dialogs, menu selections or windows.
dllcall	Calls a routine or process within a DLL module that was created to interact with ASPECT.
dllfree	Frees unneeded DLL memory resources.
dllload	Loads the specified DLL module into memory.
enable	Enables controls, dialogs, menu selections or windows.
exitwindows	Terminates all executing tasks and exits Windows.
mciexec	Executes a high-level multimedia command.
mcisend	Issues an MCI command and can retrieve the error message or the result of the command.
profilerd	Reads a specified item value from any portion of a specified Windows **.ini** file.
profilewr	Writes a specified item value to any portion of a specified Windows **.ini** file.
screentowin	Converts X/Y coordinates into coordinates relative to the specified window ID.
sendkey	Processes a keyboard shift state and a virtual key value and sends it to the focus window of the active application.
sendkeystr	Sends a specified string to the active window or dialog as if the user had sent it.
sendvkey	Processes an encoded key value and sends it to the focus window of the active application.
wintoscreen	Converts X/Y coordinates into absolute screen coordinates.

Keyboard Commands

ansitokey	Converts an ANSI character to its corresponding virtual key code and shift state.
keyflush	Clears accumulated keystrokes from the keyboard buffer.
keyget	Receives a user-keypress and stores it in a integer variable if specified.
keystate	Determines whether a key corresponding to a virtual key code is pressed.
keytoansi	Converts a key value to its ANSI character value.
keytooem	Converts a key value to its OEM character value.
oemtokey	Converts an OEM character to its key value.

Memory Commands

memaddress	Returns the memory address associated with the specified memory ID.
memalloc	Allocates a block of contiguous memory for use by a script.
memavail	Returns a long value reflecting the amount of contiguous free memory available for allocation or executing other programs.
memchr	Searches for the first occurrence of the specified character in an allocated memory block.
memcmp	Performs a byte-by-byte comparison of two memory locations.
memfree	Releases a block of memory previously reserved by **memalloc** or **memrealloc**.
memgetc	Retrieves a single character from a block of memory.
memicmp	Performs a byte-by-byte, case-insensitive comparison of two memory locations.
memmove	Copies characters from one location to another, either between different blocks or within the same block.

memputc	Writes a single character to a location within a block of memory.
memread	Reads a block of data from an allocated block of memory.
memrealloc	Changes the size of a block of memory previously reserved by **memalloc**.
memset	Initializes a block of memory to a specified value.
memsize	Reports the size of a block of memory allocated by **memalloc.**
memwrite	Writes a block of data to an allocated memory block.

Menu Commands

menubar	Creates a new script menu bar.
menucheck	Places or removes a check mark on a menu item created by a script.
menuitem	Adds an item to a menu pop-up or menu bar.
menuitemcount	Returns the number of items on a menu bar or pop-up menu.
menupopup	Adds a pop-up menu to an existing pop-up menu or menu bar.
menupopupid	Returns the menu ID of a pop-up menu on a menu bar or other pop-up menu.
menuselect	Selects a Procomm Plus or ASPECT menu item.
menushow	Displays the menu bar corresponding to the specified menu ID.
menushowpopup	Displays a floating pop-up menu.
menustate	Reports the state of a Procomm Plus or ASPECT menu item.

Option Set Commands

itemcount	Returns the number of items within an option set list.
itemcreate	Creates a new item in an option set list.
itemfind	Returns the index associated with an option set item.

itemname	Returns the name of an option set item in a string.
itemremove	Removes a script- or user-created entry from an option set list.

Numeric, String Conversion Commands

atof	Converts the contents of an ASCII string to a floating point value.
atoi	Converts the contents of an ASCII string to an integer value.
atol	Converts the contents of an ASCII string to a long value.
ceil	Computes the smallest integral value greater than or equal to a floating point value.
floor	Computes the largest integral value less than or equal to a floating point number.
ftoa	Converts a float to an ASCII string and stores it in a string variable.
itoa	Converts an integer to an ASCII string and stores it in a string variable.
ltoa	Converts a long to an ASCII string and stores it in a string variable.
numtostr	Converts a numeric value to a string.
rand	Returns a random value, ranging from 0 to 32767.
strtonum	Converts a string to a numeric value.

OEM/ANSI Commands

ansitooem	Converts an ANSI character or string to its OEM equivalent.
oemtoansi	Converts an OEM character or string to its respective ANSI character equivalents.

Packet Mode Commands

pkmode	Toggles error-free packet mode on and off.

pkrecv	Receives a packet from another computer that has issued an ASPECT **pksend** command.
pksend	Sends a packet to another computer that is running an ASPECT script.

Preprocessor Commands

#comment | **#endcomment**	Denotes a comment block.
#define	Defines a macro name and the text to substitute during compilation.
#else	Causes the *ASPECT Compiler* to process an alternate set of commands when the result of an associated **#ifdef**, **#ifndef**, **#elifdef,** or **#elifndef** is false.
#elif	Allows the *ASPECT Compiler* to determine whether the block of code which follows should be evaluated.
#elifdef	Allows conditional compilation by testing for the existence of a defined macro name after an associated **#ifdef**, **#ifndef**, **#elifdef,** or **#elifndef** has failed.
#elifndef	Allows conditional compilation by testing for the non-existence of a defined macro name after an associated **#ifdef**, **#ifndef**, **#elifdef,** or another **#elifndef** has failed.
#if	Allows the *ASPECT Compiler* to determine whether the block of code which follows should be evaluated.
#ifdef | **#endif**	Allows conditional compilation by testing for the existence of a defined macro name.
#ifndef	Allows conditional compilation by testing for the non-existence of a defined macro name.
#include	Merges commands from another ASPECT source file during compilation.
#undef	Removes the current definition of a previously-defined macro.

Printer Commands

printalign	Specifies how text will be positioned when sent to the printer.
printattr	Specifies the character attributes for text sent to the printer.
printcapture	Routes characters received by Procomm Plus or entered from the keyboard to the printer.
printchar	Prints a character on the “open” printer at the current position.
printer	Opens the current printer in preparation for any of the **print-** command family.
printfit	Determines whether a string will print on the current page.
printfont	Selects a new printer font for subsequent print operations.
printmargin	Determines the page margins for data printed from Procomm Plus.
printstr	Sends a specified string to the open printer.
printtabs	Determines the tab positions for strings printed with the **printtabstr** command.
printtabstr	Prints a line of text, expanding tabs to the positions set with the **printtabs** command.

Script Control Commands

breakpoint	Interrupts execution of a script file and displays the **Debug** window.
chain	Executes another ASPECT script.
compile	Executes the *ASPECT Compiler* upon a target source file.
execute	Executes another ASPECT script.
exit	Terminates the executing script.
halt	Terminates the executing script and any parent scripts unconditionally.

longjmp	Returns to a previously marked **setjmp** location.
setjmp	Marks a location within a script that execution can immediately branch to using the **longjmp** command.
yield	Suspends script execution, allowing other tasks in Procomm Plus to continue execution until ASPECT is permitted to resume by the system.

String Commands

nullstr	Tests a string variable for the null condition.
rstrcmp	Compares the contents of two strings up to the specified length.
strcat	Concatenates one string to another string variable.
strchr	Searches for the first occurrence of a character in a given string.
strcmp	Compares two strings.
strcpy	Assigns a string to a string variable.
strcspn	Searches a string for a character contained in a second string.
strdelete	Removes characters from the contents of a string variable.
strextract	Returns a string from a list of elements.
strfind	Tests for the occurrence of the specified text within a variable.
strfmt	Creates a formatted string using a template and specified variables.
strgetc	Returns the ASCII value of a single character in a string.
stricmp	Performs a case-insensitive comparison of two strings.
strinsert	Inserts the contents of a string into another string.
strlen	Returns the length of a string or string variable’s contents.
strlwr	Converts the contents of a string variable to all lowercase.
strncmp	Performs a case-sensitive comparison of two strings for a specified length.

strnicmp	Performs a case-insensitive comparison of two strings for a specified length.
strputc	Modifies the value of a single character within a string.
strquote	Places the contents of a string within double quotes.
strrchr	Searches for the last occurrence of a character in a given string.
strreplace	Searches a string for a specific pattern of characters and replaces that pattern with another string.
strrev	Reverses the contents of a string.
strright	Copies a string of characters from the end of a string.
strsearch	Searches a string for a specific character or pattern of characters.
strset	Sets the characters in a string variable to a specified ASCII value.
strspn	Searches a string for the first character that does not occur within a second string.
strtok	Parses a delimited string into tokens.
strupdt	Overwrites a string with another string at a specified index.
strupr	Converts the contents of a string variable to all uppercase.
substr	Copies the indicated number of characters from a string, beginning at a specified position.

Task / Window Manipulation Commands

firsttask	Returns the first task encountered in the Windows Task List.
nexttask	Returns the next task encountered in the Windows Task List after the execution of a **firsttask** command.
taskactivate	Activates the task with the specified task ID.
taskexists	Tests a task ID value and determines whether the task still exists.
taskexit	Terminates a Windows task.
taskname	Returns the executable name associated with a specified task ID.

taskpath	Returns the executable directory used by a specified task ID.
taskwin	Returns the main or top-level window ID associated with a task ID.
winactivate	Activates the specified window.
winclose	Sends a “close” message to a window.
wincoord	Returns the location and dimensions of the specified window.
winenabled	Tests the target window to determine its availability to the user or the script.
winexists	Tests for the existence of a window.
winfocus	Specifies the window to receive keyboard input.
winhide	Conceals the target window from view.
winmaximize	Maximizes an application window to fill the screen.
winminimize	Minimizes an application window to an icon.
winmove	Positions an application window.
winowner	Returns the owner ID of a window ID.
winrestore	Restores an application window to its previous size from a minimized state.
winshow	Makes a hidden window visible.
winsize	Sizes the specified application window.
winstate	Returns an integer indicating the state of a window.
wintask	Returns the task ID of a window.
wintext	Returns the caption or titlebar text of a window.
winvisible	Tests whether a window is hidden or visible.

Terminal Commands

clear	Clears the *Terminal window* and the status line.

commandmode	Switches the modem on the current connection to command mode.
getcur	Returns the current cursor location on the *Terminal display.*
locate	Positions the terminal cursor to the location specified by row and column.
sbsave	Empties the *Scrollback Buffer* into a specified destination.
snapshot	Copies the contents of the current logical terminal screen to a specified destination.
termgetc	Returns a character value from the *Terminal window.*
termgets	Returns a string from the *Terminal window.*
termkey	Processes a virtual key value and shift state and sends it to the *Terminal window.*
termmsg	Writes a formatted string to the *Terminal window.*
termputc	Writes a character value to a specified location in the *Terminal window.*
termputs	Writes a string to a specified location in the *Terminal window.*
termreadc	Reads a character value from the *Terminal window* at the current cursor location.
termreads	Reads a string from the *Terminal window* at the current cursor location.
termreset	Resets the *Terminal window.*
termvkey	Processes an encoded key value and acts upon it as if it were typed in the *Terminal window.*
termwritec	Writes a character value to the *Terminal window.*
termwrites	Writes a string to the *Terminal window.*

User Window Commands

bitmap	Places a bitmap graphic in the User window.
bitmapbkg	Places a bitmap background in the User window.

dllobject	Identifies a graphic object within a User window to be updated by an associated DLL module.
dllobjfile	Specifies a file containing a graphic object to be accessed or updated with the **dllobject** command.
dllobjupdt	Allows a DLL module linked to a DLL object to update that object.
hotspot	Places a mouse-selectable rectangle in the User window.
icon	Displays an icon graphic in the User window.
iconbutton	Displays an icon graphic as a clickable button in the User window.
metafile	Displays a metafile graphic in the User window.
metafilebkg	Places a metafile background in the User window.
objcoord	Returns the location coordinates of an object in the User window.
objhide	Hides an object, or a range of objects in the User window.
objmove	Moves an object identified by an ID number to a new location in the User window.
objpaint	Updates the User window and some or all of the objects it contains.
objpointid	Returns the object ID, if any, of an object located at a specific point.
objremove	Removes one or more object from the User window
objshow	Displays an object previously hidden with **objhide**.
pushbutton	Places a standard button control in the User window.
uwincreate	Defines a User window which occupies all or part of the *Terminal* window.
uwinpaint	Paints the entire User window and all of its objects.
uwinremove	Removes the User window background or the entire User window.
uwutowin	Converts X/Y User Window Units to window coordinates.
wintouwu	Converts X/Y coordinates relative to a window into User Window Units.

Using Command Descriptions

This sample description, for an imaginary ASPECT command called “**command**”, demonstrates and explains the layout used throughout the command listings:

command

A The text following the command header provides a brief explanation of the **command**’s purpose. If the **command** returns SUCCESS and FAILURE, the explanation will be marked with an "**S/F**" bullet — just as this text is. Following the statement of purpose, the **command**’s full syntax is shown in bold face:

command parameters KEYWORDS

parameters	*Parameters* are shown in the syntax statement using the terms listed in“*ASPECT Conventions*” on page 42. *Parameters* are always shown in lowercase, and in italic face if they appear in body text.
KEYWORDS	ASPECT KEYWORDS are always shown in uppercase.

Comments

The “*Comments*” section provides more detailed information about the **command**. It often contains notes regarding the **command**’s interactions with other ASPECT commands or suggestions about using the **command** in special situations.

ASPECT commands appearing in body text are always shown in lowercase and boldface. For example, **addfilename**. ASPECT script examples present all **command** names in lowercase. However, the only case-sensitive elements within ASPECT are quoted strings and format specifiers used in commands like **fstrfmt** and **strfmt**. Feel free to use any combination of upper- and lowercase characters in your scripts to make your code easier to read.

If a command’s purpose is related to other ASPECT commands, **set**/**fetch** options or system variables, they will be listed in a “*See also*” section following any “*Comments.*”

Command Listings

The following pages detail the ASPECT commands. The commands are listed in alphabetical order, and the descriptions follow the format described in “*Using Command Descriptions*” on page 88.

The ASPECT Commands

addfilename

Adds a specified filename to a path.

addfilename pathname filename

pathname	A string variable containing the path to which the filename will be appended.
filename	A string containing the filename to append.

Comments

addfilename checks the supplied path for a terminating backslash or colon and automatically adds a backslash if appropriate. After the command executes, the completed path is stored in the *pathname* string variable.

See also

getdir and **fullpath**; $ASPECTPATH, $PWLOCALPATH and $USERPATH.

alarm

Sounds an alarm to alert you to an event.

alarm [integer]

integer	An optional integer value that specifies the number of seconds the alarm will sound.

Comments

alarm uses the Windows default beep sound and is not affected by the status of the **set** alarmtime integer. The beep will repeat for the duration of the specified alarm time, but a <Ctrl><Break> sequence will abort the alarm sound.

See also

beep and **mciexec**; **set** alarmtime integer.

ansitokey

A Converts an ANSI character to its corresponding virtual key value and keyboard shift state.

ansitokey character intvar

character	The ANSI character to convert.
intvar	The character’s converted keyvalue.

Comments

The shift state is contained in the high-order byte, and the key code in the low-order byte, as returned by **keyget** or accepted by **termvkey**. The numeric keypad keys will not be converted, since **ansitokey** translates only the main keyboard keys. If the character does not correspond to a key, a (-1) is returned.

See also

ansitooem, **keytoansi**, **keyget** and **termvkey**.

ansitooem

A Converts an ANSI character or string to OEM equivalent.

ansitooem {character intvar}{string strvar [strlength]}

character	The ANSI character to convert.
intvar	The character’s converted OEM value.
string	The string to convert to OEM.
strvar	Will contain the converted string.
strlength	The number of characters in the string to convert. If strlength is not specified, the entire contents of the string will be converted.

See also

ansitokey and **oemtoansi**; **set** aspect codepage integer.

atof

Converts the contents of an ASCII string to a floating point value.

atof string floatvar

string	String to convert to a float variable.
floatvar	Variable where the float value is stored.

Comments

atoi, **atof** and **atol** skip leading white space, including spaces and tabs. Conversion begins at the first numeric character and ends at the first character which is not a part of the number format. The format for floats is:

[sign] [digits] [.digits] [{d|D|e|E} [sign] digits]

If the string doesn't contain a float value, **atof** assigns a value of 0.0. **atof** will only convert numbers in base 10.

See also

atoi, **atol**, **ftoa** and **strtonum**.

atoi

Converts the contents of an ASCII string to an integer value.

atoi string intvar

string	String to convert to an integer.
intvar	Variable where the integer is stored.

Comments

Conversion begins at the first digit encountered in the string and stops at the first non-numeric character. If the string doesn't contain an integer value, **atoi** assigns a value of 0. **atoi** will only convert numbers in base 10.

See also

atof, **atol**, **ltoa** and **strtonum**.

atol

Converts the contents of an ASCII string to a long value.

atol string longvar

string	String to convert to a long value.
longvar	Variable where the long is stored.

Comments

Conversion begins at the first digit encountered in the string and stops at the first non-numeric character. If the string doesn't contain a long value, **atol** assigns a value of 0. **atol** will only convert numbers in base 10.

See also

atof, **atoi**, **ltoa** and **strtonum**.

beep

Sounds a beep tone, using the Windows default beep sound.

beep

See also

alarm and mciexec.

bitmap

A Places a bitmap graphic in a dialog box or in the User window. **bitmap** can only be tested for SUCCESS/FAILURE when it appears in a User window.

Dialog box format

bitmap id left top width height filespec

id	A unique integer constant value assigned to each bitmap.
left top	Integer constant values in Dialog Box Units (DBUs) which determine the position of the top left corner of the bitmap with respect to the dialog box. For more information on DBUs, see“*Dialog Box Units*” on page 48.
width height	Integer constant values in DBU’s which determine the width and height allowed for the bitmap. Care should be taken in setting the values for width and height. Unlike the format of the **uwincreate** or User window **bitmap** command, the dialog box form utilizes the width and height specified within the command. This could either clip the bitmap, or leave a graphical gap on the right-most or bottom edge of the bitmap.
filespec	A string constant or string variable describing the filename and path of the bitmap to display.

Comments

If no path is included in the *filespec* argument, the bitmap will be sought in the Aspect Path. **dlgupdate** can be used to refresh the bitmap, making it possible for a script to dynamically change the displayed graphic.

See also

dialogbox, dlgupdate, icon, iconbutton and metafile.

User window format

bitmap id left top label filespec BACKGROUND | USERWIN

id	A unique integer value assigned to each bitmap. The $OBJECT system variable will be updated to this id value when the bitmap is selected by the user with the left mouse button. It is also used with the **objpaint** command to update the bitmap, and the **objremove** command to remove the bitmap from the User window.
left top	Integer values which determine the position of the top left corner of the bitmap with respect to the User window or the background graphic. Each integer is expressed in User Window Units or UWUs. For more information on UWUs, see“*User Window Units*” on page 48.
label	Label displayed with this bitmap. The ampersand (&) can be used to indicate an accelerator for the bitmap. To display an ampersand in the label, use two ampersands. For example, the text “***&R&&D***” used as a label would display “***<u>R</u>&D***,” with the ***R*** displayed as an underscored accelerator.
filespec	A string constant or global string variable describing the filename and path of the bitmap to display. Local variable strings are not allowed. If a path is not specified, the current Aspect Path will be assumed.
USERWIN | BACKGROUND	Specify USERWIN to lock the top left corner of the bitmap in position relative to the upper left corner of the User window. If BACKGROUND is specified, the upper left corner of the bitmap stays over the same spot on a background graphic.

Comments

A User window must exist before a **bitmap** can be used. When the left mouse button is clicked on the **bitmap**, its label will invert. If no label is specified, the entire **bitmap** will invert.

See also

icon, iconbutton, pushbutton, dllobject, hotspot, bitmapbkg, metafile, uwincreate, objpaint, objhide, objremove, objshow, objmove, objcoord and objpointid; $OBJECT in “*System Variables*.”

bitmapbkg

A Places a bitmap background in the User window.

bitmapbkg CENTER | LEFT | RIGHT | TILED CENTER | TOP | BOTTOM MEMORY | DISK filespec

CENTER | LEFT | RIGHT | TILED	The horizontal position of the bitmap background with respect to the User window. If the bitmap is larger than the User window, it will be clipped along the bottom and right edges as necessary. TILED repeats the bitmap, beginning at the upper left corner of the User window, until the User window is filled with multiple copies of the bitmap. Some of the tiled images may be clipped, depending upon the size of the bitmap.
CENTER | TOP | BOTTOM	The vertical position of the bitmap background with respect to the User window.
MEMORY | DISK	Determines if the bitmap will be loaded into memory for re-display or reloaded from disk. MEMORY is faster, but DISK can be used to save memory space.
filespec	A string constant or global string variable describing the filename and path of the bitmap to display. If a path is not specified, the current Aspect Path is used by default.

Comments

The **uwinpaint** command is used to update the display of the User window.

See also

metafilebkg, bitmap, uwincreate, uwinremove and uwinpaint.

break

Sends a break signal to a remote computer system.

break

Comments

The length of the break can be specified in ***Setup, Data, Data Options, Advanced***, or with the **set port breaklen** command.

See also

set port breaklen in *“Set and Fetch Statements*.”

breakpoint

Interrupts execution of a script file and displays the Procomm Plus **Debug** window.

breakpoint [string | {formatstr arglist}]

string	A string of up to 256 characters.
formatstr arglist	A string of up to 256 characters containing up to 12 format specifiers followed by a list of arguments. The arguments must be listed in the order they are specified within *formatstr.*

Comment

The script must be compiled with the debug option ON for **breakpoint** to operate. Otherwise, the **breakpoint** command is ignored. The debug option is set by enabling the ***Compile for Debug*** check box in the **ASPECT Compiler Options** dialog.

For more information see,“*Compile-Time or Syntax Errors*” on page 556.

See also

compile.

call

Causes script execution to branch to a specified procedure or function and allows a return to the command following the call.

Function format

call name [WITH arglist] [INTO variable]

Procedure format

call name [WITH arglist]

name	The user-defined name of the procedure or function. *Name* must follow the standard ASPECT naming conventions.
WITH arglist	An optional list of up to 12 arguments to be passed to the procedure or function. Note that the items in *arglist* must match the order and number of parameters in the declaration of the called process.
INTO variable	Allowed for functions only. *Variable* will contain the function’s return value. The variable must be of the same data type as specified for the return value in the function declaration.

Comments

An argument to a function or procedure can be a constant or a variable. Variables are normally ***passed by value***. This means that only the variable’s ***value*** is given to the called procedure or function, and that any changes the called process makes to the value do not affect the actual variable.

Variables can also be ***passed by reference***. This means that a reference (or address) to the variable is given to the called procedure or function, and that any changes made to the corresponding argument by the called process will be recorded in the referenced variable after control returns to the calling procedure or function.

The "&" character, preceding a variable argument in a **call** statement, indicates that the argument is being ***passed by reference***.

Both functions and procedures can also be called in the shortened format:

name ([arglist])

value = myfunc ([arglist])

Functions called with the shortened format can also be used in general expressions. The shortened form of function and procedure calls is highly recommended. It is shorter, just as readable, and less verbose. Also, future versions of ASPECT may require it.

See also

return, proc, endproc, func, longjmp and goto.

capture

Controls session capture during script file execution. The *Capture* file is a continuous record of all characters received and transmitted.

capture OFF|ON

Comments

Capture file recording options determine how data is written to the capture file. To avoid capturing unwanted data, use the **clear** command first. Note that a **set aspect display off** command in a script will disable the *Capture* file.

See also

clear and snapshot; the set capture command in "*Set and Fetch Statements*."

capturestr

Writes a specified string to the open *Capture* file. **capturestr** is often used to insert comments or "landmarks" within a *Capture* file to record certain events.

capturestr string | [formatstr arglist]

string	A string of up to 256 characters.
formatstr arglist	A string of up to 256 characters containing up to 12 format specifiers followed by a list of arguments. The arguments must be listed in the order they are specified within *formatstr.*

Comments

A *Capture* file must be opened by the user, or by a script with the **capture** command for **capturestr** to work properly.

See also

capture; the set capture command in *“Set and Fetch Statements.”*

case

Declares a **string**, **integer**, or **long** value to be tested against the **switch** statement’s target value.

case integer | long | string

Comments

A successful match causes processing to continue with each command on subsequent lines until an **endcase** or **exitswitch** command is encountered. Successive **case** commands appearing before the **endcase** are ignored. A failed match results in the processing of any subsequent **case** or **default** command.

Several **case** commands can occur within a **case**/**endcase** block. These are ignored by any previous **case** match. Only the commands other than **case** or **default** are processed between the matched **case** and **endcase** command.

If a **case** target is matched, the statements following it are executed until an **endcase** or **exitswitch** is encountered. If another **case** is encountered, it’s skipped. Execution then falls through to the next commands following the matched **case** target. If a match is made with more than one **case** command, only the statements following the first matched **case** command will be executed.

➥ ***default** can occur anywhere within the **switch** statement. **case** statements occurring between **default** and its terminating **endcase** or **exitswitch** will be ignored.*

For more information on **case**, see the **switch** command.

See also

switch, default, exitswitch and endcase.

ceil

Computes the smallest integral value, with no fractional amount, greater than or equal to a floating point number.

ceil float numvar

float	A float variable or constant.
numvar	A numeric variable which receives the command result.

See also

floor.

chain

A Executes another Windows ASPECT script.

chain filespec [SHARED]

filespec	The name of the ASPECT file to **chain**.
SHARED	Causes the chained script to inherit the parent script's current ASPECT environment settings, rather than the default settings.

Comments

ASPECT will not return control to the original script, so all processing should be completed before a **chain** command is encountered.

The *filespec* doesn't require a path. If the path is not provided, the current Aspect Path is assumed. To change the current Aspect Path, see the **set aspect path pathname** command.

If an extension of **.wax** is specified but doesn't exist, ASPECT will search for a source file with a **.was** extension and compile it automatically. If an extension of **.was** is specified, the source is always automatically compiled before execution. If the extension is omitted and the source file is newer than an existing compiled script, the source will automatically be recompiled.

When a script **chain**s to another script, existing ASPECT dialog boxes are destroyed, as are any DDE client conversations currently in effect. However, the **chain**ed script inherits the current predefined and system variable values, as well as the state of all **set** commands, the current User window and its objects, and any menus and menu selections created by ASPECT.

The $CHAINEDFILE system variable will return the name of the script which issued the **chain** command. If the current script was not **chain**ed, it will return null.

The $SCRIPTMODE system variable can be used to determine whether a script process was spawned or **chain**ed. $SCRIPTMODE will return 2 if the current task was **chain**ed from a parent script. **chain**ed scripts can use data already stored in the pre-defined variable set, whether SHARED is specified.

For more information about chaining and the Aspect environment, see"*Script Spawning and Chaining*" on page 55 and "*The ASPECT Script Environment*" on page 54.

See also

call and execute; $SCRIPTMODE, $SCRIPTENV, $ASPECTPATH, and $CHAINEDFILE in "*System Variables*."

chdir

A Changes the User Path to the specified drive and/or directory.

chdir pathname [ASPECTPATH]

pathname	Identifies the DOS drive (for example, C:), pathname and directory. The drive identifier is only required if you change the current drive.
[ASPECTPATH]	An optional parameter that sets the Aspect Path to the new User Path.

Comments

The User Path is the default path used by many ASPECT commands. If a path isn't specified, the current User Path is used as the originating directory. For more information, see"*The ASPECT Script Environment*" on page 54.

See also

mkdir, rmdir, getdir and taskpath; set aspect path in "*Set and Fetch Statements.*" $WINPATH, $USERPATH and $ASPECTPATH in "*System Variables*."

checkbox

Adds a "checkmark" selection control to a script dialog box. The control may be either checked or unchecked.

checkbox id left top width height label intvar

id	A unique integer constant value assigned to the check box. The **dlgevent** command will assign this value to its target variable when the check box is accessed by the user. The value can also be used with **dlgupdate** to refresh the display of the check box.
left top	These integer constants determine the position of the top left corner of the check box control in Dialog Box Units, or DBUs. For more information on DBUs, see“*Dialog Box Units*” on page 48.
width height	Integer constants specifying the width and height of the check box in DBUs.
label	A string variable or constant, displayed on the right side of the check box. To specify a keyboard accelerator for the check box, simply include an ampersand (&) in front of the desired character. To display an ampersand in the label, use two ampersands. For example, the text “***&R&&D***” used as a label would display “***R&D***,” with the ***R*** displayed as an underscored accelerator.
intvar	Contains the value 0 if the box is unchecked and 1 if it is checked. The default check box state can be assigned by setting this value to 0 or 1 before the dialog box is displayed.

Comments

A **checkbox** command can only occur within a **dialogbox** command group. When a check box is selected, the **dlgevent** command will update its target variable with the id specified for the check box.

See also

dlgevent, dlgupdate and dialogbox.

clear

Clears the *Terminal display* and the status line.

clear

Comments

The **clear** command has no effect while the Procomm Plus *Terminal window* is in *Scrollback* mode. Script execution merely continues to the next command. If a **clear** command is executed while text is being marked in the *Terminal window*, however, script execution is suspended. After the marking operation is completed or cancelled, the **clear** command is executed and the script resumes normally.

clearxoff

Clears an XOFF character, usually **Ctrl-S,** sent by a remote computer. XON/XOFF flow control, commonly called "software" flow control, is often used to pace the transmission of ASCII data. If the flow of data is accidentally paused by line noise or an embedded XOFF character, this command will resume the transfer.

clearxoff

Comments

XON/XOFF flow control can often cause a communications session to "hang," since the transmitting computer may wait indefinitely to receive an XON character, **Ctrl-Q**. **clearxoff** has no effect if XON/XOFF flow control has been disabled.

See also

set port softflow in "*Set and Fetch Statements*" and $TXCOUNT, $RXCOUNT, $FLOWSTATE and $XOFFRECV in "*System Variables*."

cliptofile

A Copies data from the Windows Clipboard to a file.

cliptofile {BITMAP | METAFILE filespec} | {TEXT filespec [APPEND]}

BITMAP	Write the information as a bitmap.
METAFILE	Write the information as a metafile. Using a file extension of **.emf** causes the file to be saved in an enhanced metafile format; otherwise, the file is saved using the "placeable" standard metafile format.
filespec	The target filename; a path is optional. If a path is not specified, the User Path is used by default.

TEXT	Write the information as a text file.
filespec	The target filename; a path is optional. If a path is not specified, the User Path is used by default.
APPEND	Appends the contents to the end of an existing file. If the file does not exist, the file will be created.

Comments

The clipboard must contain valid data of the indicated type for a successful copy operation. The data is not erased from the Windows Clipboard by **cliptofile**.

See also

cliptostr, filetoclip and pastetext.

cliptostr

A Copies text from the Windows Clipboard to a string.

cliptostr strvar

strvar	The target string.

Comments

Up to 256 characters can be copied from the Windows Clipboard to the string. The Clipboard must contain textual data of some kind.

See also

cliptofile, filetoclip and strtoclip.

combobox

Adds a combination edit/selection or “drop-down” selection control to an ASPECT dialog box. Allows the selection of a single item from a list of items.

combobox id left top width height SIMPLE | DROPDOWN | DROPDOWNLIST itemlist strvar [strlength] [SORT]

id	A unique integer constant assigned to the combo box.

left top	Integer constants which determine the position of the top left corner of the combo box in dialog box units, or DBUs. For more information on DBUs, see“*Dialog Box Units*” on page 48.
width height	Integer constants which specify the width and height of the combo box in DBUs.
SIMPLE	Causes the list box to be displayed at all times. The current selection in the list box is displayed in the edit control.
DROPDOWN	Causes the list box to be displayed only if the user selects the down arrow icon to the right of the edit control. The current selection is displayed in the edit control.
DROPDOWNLIST	Similar to the DROPDOWN type, but the edit control is replaced by a static text item that displays the current selection in the list box.
itemlist	A string variable or constant containing a list of items to display within the combo box. Each item is separated by a comma.
strvar	Assigned the text of the item selected by the user. This variable can be pre-assigned a value from the list to be displayed as the default selection.
strlength	An optional integer variable or constant indicating the number of characters which can be input by the user into the edit control. The *strlength* argument is not valid with the keyword DROPDOWNLIST, since that type does not provide an edit field.
SORT	An optional parameter that sorts the contents of the combo box alphabetically.

Comments

The **dlgevent** command assigns the *id* value to its target variable when a selection is made in the combo box by the user. The *id* value can also be used with **dlgupdate** to refresh the display of the combo box, or to enable or disable the combo box.

If the combo box contains an edit field, and the user types information into the field, a dialog event will be generated when the combo box loses the input focus.

See also

fcombobox, flistbox, dialogbox, listbox, dlgevent and dlgupdate.

comgetc

A Assigns the next character value in the receive data buffer to an integer variable.

comgetc intvar [integer]

integer	An optional value which specifies the total number of seconds **comgetc** will wait for an available character.

Comments

If the receive buffer is empty, or if it remains empty during the specified timeout period, **comgetc** assigns a value of -1. If a timeout value is specified, the Ctrl-Break sequence can be used to terminate the command before the timeout period has completed.

A character retrieved by the **comgetc** command will not be displayed in the *Terminal window*, regardless of the **set aspect rxdata** state. Similarly, **comgetc** will not be interrupted by a **when** $RXDATA condition, since **comgetc** effectively “steals” data from the received data buffer.

comgetc will fail if the script does not have control of the communications port.

See also

comread and computc; set aspect rxdata and set aspect display in “*Set and Fetch Statements*.”

commandmode

A Switches the modem on the current connection to command mode.

commandmode OFF | ON

Comments

This is equivalent to the ***Data* | *Modem Command Modem*** menu item found in the *Terminal window*. As long as command mode is active, all answer options are disabled and no other applicatons can access the current connection.

commandmode fails if the current connection is a non-TAPI connection or if the connection could not be opened.

See also

$CONNECTOPEN.

#comment

Denotes the start of a comment block. Any commands or remarks following this command are ignored by the compiler until an **#endcomment** is encountered.

#comment

Comments

#comment blocks cannot be nested. Single-line comments are marked by a semicolon(;), and can be embedded within a **#comment**/**#endcomment** block.

See also

#endcomment.

compile

A Executes the *ASPECT Compiler* upon the target source file.

compile filespec [string]

filespec	The name of the source file to **compile**. This file must have an extension of **.was** on disk, but the extension is not required in the *filespec* argument. The Aspect Path is used by default if no path is specified.
string	An optional string containing arguments for the *ASPECT Compiler.* These arguments are in addition to those currently specified in the **pw4.ini** file. However, a "/N" argument tells the compiler to ignore the **pw4.ini** options.

Comments

Script execution is suspended until the compilation is finished. Ctrl-Break can be used to terminate the **compile** command, allowing script execution to continue.

For more information about command-line options, "*The ASPECT Compiler*" on page 568.

See also

chain, execute and run.

computc

A Sends the specified character value to the communications port.

computc character

Comments

Use **termkey** to send the current emulation's *Keyboard Mapping* code for special keys like the function and cursor control keys.

computc will fail if the script does not have control of the communications port.

Date will be transmitted at a rate determined by the current baudrate and transmit pacing values. For fastest possible throughput at the current baudrate, make sure that transmit pacing is set to 0.

See also

comgetc, comwrite, transmit and termkey.

comread

A Retrieves data from the receive data buffer and stores it in the target variable. The amount of data read depends on the type of data variable specified. For integers and longs 4 bytes, for floats 8, and for strings is user-specified.

comread numvar| {strvar strlength} [integer [intvar]]

numvar	The target data variable.
strvar strlength	A string target variable. *strlength* is an integer value which specifies the number of characters to be read into *strvar.* The maximum value is 256 characters.
integer	An optional value which specifies the total number of seconds **comread** will wait for data. **comread** will timeout if the requested data has not been received in the specified time. If no timeout value is specified, and the amount of available data is less than the requested amount, **comread** will fail immediately.
intvar	If specified, this variable will contain the actual count of data read.

Comments

comread is most commonly used to obtain “raw” data from the port. Because it can store retrieved values as any ASPECT data type, it is the most flexible read port command.

Data retrieved by the **comread** command will not be displayed in the *Terminal window*, regardless of the **set aspect rxdata** state. Similarly, **comread** will not be interrupted by a **when** $RXDATA condition, since **comread** effectively “steals” data from the received data buffer.

If the amount of data read is less than requested, **comread** has failed. **comread** will also fail if the script does not have control of the communications port. **Ctrl-Break** can be used to terminate a **comread** command.

See also

comgetc, computc, comwrite, rget, termkey and transmit.

comwrite

A Sends the specified information to the communications port.

comwrite number | {string strlength}

number	An integer, float, or long value to be written to the port.
string	A string value to be written to the port.
strlength	The number of characters to be written from the string. Maximum length of the string is 256 characters.

Comments

comwrite is often used to send “raw” binary data to a remote system. The amount of data written to the port depends on the data type specified. For more information, see the **comread** command.

Date will be transmitted at a rate determined by the current baudrate and transmit pacing values. For fastest possible throughput at the current baudrate, make sure that transmit pacing is set to 0.

comwrite will fail if the script does not have control of the communications port.

See also

comgetc, computc, comread, rget, termkey and transmit.

connect

A Calls one or more entries in your *Connection Directory.*

connect dialclass [GROUP] name [name...][CONNECTALL]

dialclass	A keyword representing a *Connection Directory* entry class. Valid keywords are: DATA, MAIL, FAX, FTP, NEWS, TELNET, VOICE, and WWW.
GROUP	Dial the group(s) specified.
name	A string identifying the entry or group name to dial.
name...	Optional additional entry or group names to dial. Up to 12 names in all can be specified.
CONNECTALL	Initiates a continuous attempt to connect to all selected *Connection Directory* entries. When a connection is made and then terminated, the dialing process continues to the next entry in the list and resumes dialing. If CONNECTALL is not used, any successful connection will terminate the dialing queue.

Comments

The **connect** command is a synonym for **dial**. For further details, please see the **dial** command.

See also

dialadd, dialcancel, dialclass, dialcount, dialcreate, dialdelete, dialfind, dialinsert, dialload, dialname, dialnumber, dialsave and dialstats; set aspect dialingbox, the set dialdir and set dialentry command families in “*Set and Fetch Statements*.” Also look at $DIALCONNECT, $DIALENTRY, $DIALSELECT, $DIALQUEUE, $SCRIPTMODE and $DIALING in “*System Variables*.”

connectmanual

A Connects to the specified telephone number.

connectmanual dialclass "phonenum" | string

dialclass	Specifies the class of call to process for this number.
string	A valid telephone number.

Comments

connectmanual is a synonym for the **dialnumber** command. **connectmanual** accepts only a single telephone number. For multiple *Connection Directory* entry numbers, use the **dial** or **connect** command. **connectmanual** will fail if a dialing operation or a file transfer is already in effect.

See also

connect, dial, dialcancel, dialnumber and dialload; $DIALING in "*System Variables*" and the set dialdir command family in "*Set and Fetch Statements*."

copyfile

A Copies a file to another file or path.

copyfile filespec filespec

filespec	The source filename. A fully-qualified path is recommended. If no path is specified, the User Path is used as the default.
filespec	The destination. Either a fully-qualified path, with or without the filename, or a new filename with a path may be specified. If a path is omitted, the User Path is used as the default.

Comments

copyfile works similar to the DOS COPY command. However, **copyfile** does not support wildcards. Script execution is suspended until the file is copied.

See also

chdir, delfile, isfile and rename; $USERPATH in "*System Variables*."

crc16

Generates a Cyclic Redundancy Check (CRC) integer value for a specified string. CRC values are often used to determine whether data has been changed by line noise.

crc16 intvar string [strlength]

intvar	Before the **crc16** command is executed, this integer holds the starting value of the CRC polynomial. After the **crc16** has executed, it holds the current CRC value.

string	The string to evaluate.
strlength	The string length. If omitted, a null-terminated string is assumed.

Comments

Prior to the first execution of this command, the script should set *intvar* to the desired initial CRC value. Typically, this initial CRC value is 0 or -1, and it must be used when the data is verified at some other time. In other words, two CRC values for the same data cannot be compared unless each CRC value was calculated using the same initial value.

crc16 allows the user to calculate CRC values on a single string or multiple strings. For more than one string, all subsequent executions of **crc16** should use the value last returned into *intvar* by the previous **crc16** command. If the optional length isn't included, the length will be calculated at run-time, assuming a null-terminated string.

crc16 uses CCITT polynomial 0x1021.

See also

comread and comwrite.

ddeadvise

A Allows Procomm Plus to request both the initial value of a variable and any subsequent changes to that variable from a Windows Dynamic Data Exchange (DDE) server. Each time the value of the specified variable changes in the server application, Procomm Plus is informed of the change.

ddeadvise long string gdatavar [integer]

long	Specifies the DDE channel number returned by the **ddeinit** command.
string	The name of the server variable to be monitored.
gdatavar	The name of the Procomm Plus global data variable that will be changed each time the server sends a new value for the specified variable.
integer	An optional identifier, indicating when a specific variable among multiple **ddeadvise** commands has been updated. The value is used in conjunction with the $DDEADVISE system variable to indicate which **ddeadvise** command has been updated. The value must be a positive, non-zero value. If not specified, events associated with the server variable will return -1 in $DDEADVISE.

Comments

A **ddeadvise** operation can be tested with the **if** SUCCESS statement, returning true if the operation was successful and false if it failed.

The DDE manager only maintains one advise loop per item per conversation. More than one **ddeadvise** can be established on the same item, and each can be uniquely indentified by using the optional ID. A **ddeunadvise**, however, will terminate all active advisements for that item.

See also

ddeinit, dderequest, ddepoke, ddeunadvise and when $DDEADVISE; $DDEADVISE "*System Variables*."

ddeexecute

A Executes a command in another Windows program acting as a Dynamic Data Exchange (DDE) server.

ddeexecute long string

long	Specifies the DDE channel number returned by the **ddeinit** command.
string	The server application DDE command. Server commands usually must be provided in a specific format; check the server application's documentation.

Comments

A **ddeexecute** operation can be tested with the **if** SUCCESS statement, returning true if the operation was successful and false if it was not. For more information on using Procomm Plus as a DDE server, refer to on-line help. A discussion of DDE client operations appears in"*A Brief ASPECT Tutorial*" on page 453.

See also

ddeinit.

ddeinit

A Establishes a Dynamic Data Exchange (DDE) data channel between Procomm Plus and a Windows DDE server application. This channel can be used to exchange data between the two programs, and the client can execute server commands.

ddeinit longvar string string [integer]

longvar	If **ddeinit** is successful, *longvar* is a DDE channel number. This channel number can be used in other DDE commands for data exchange and control.
string	The name recognized by the server application for DDE exchanges. This name is usually provided in the server application's documentation. Typically it is the name of the application itself.
string	The name of the document, file or topic recognized by the server application.

integer	An optional value that uniquely identifies the DDE server application. This value can be used to establish a DDE conversation with a specific instance of an application when more than one instance is running. ASPECT assumes that for a typical application, this value is appended to the DDE server name as the ASCII representation of an unsigned integer value (as if a DDE server name string were created with the format “MYAPP%u”). Consult the application documentation to determine what their DDE server naming conventions are. You may have to format your own DDE server name string, and avoid using the optional integer value. To identify a unique instance of Procomm Plus, the integer value used should be the main window ID of that instance. This value will be used to produce a DDE server name that uniquely identifies that specific instance of Procomm Plus.

Comments

A **ddeinit** operation can be tested with the **if** SUCCESS statement, returning true if the operation was successful and false if it was not. A discussion of DDE client operations appears in“*A Brief ASPECT Tutorial*” on page 453.

See also

ddeadvise, ddeterminate, ddeexecute, dderequest and ddepoke.

ddepoke

A Assigns data from ASPECT to a Windows Dynamic Data Exchange (DDE) server application variable.

ddepoke long string data

long	Specifies the DDE channel number returned by the **ddeinit** command.
string	The name of the variable or item which will receive the data in the server application.
data	The data to be sent to the server application. It can be any data type supported by ASPECT.

Comments

A **ddepoke** operation can be tested with the **if** SUCCESS statement, returning true if the operation was successful and false if it failed.

See also

dderequest, ddeinit and ddeadvise.

dderequest

A Receives the current value of a variable from a Dynamic Data Exchange (DDE) server application.

dderequest long string datavar

long	Specifies the DDE channel number returned by the **ddeinit** command.
string	The name of the server’s variable or item reference.
datavar	The name of the ASPECT variable which will receive the data.

Comments

A **dderequest** operation can be tested with the **if** SUCCESS statement, returning true if the operation was successful and false if it was not.

See also

ddepoke, ddeinit and ddeadvise.

ddeterminate

Terminates an existing Dynamic Data Exchange (DDE) data channel.

ddeterminate long

long	The DDE channel number returned from the **ddeinit** command to be terminated.

See also

ddeinit.

ddeunadvise

A Terminates a current **ddeadvise** condition.

ddeunadvise long string

long	Specifies the DDE channel number returned by the **ddeinit** command.
string	The name of the server variable currently being monitored. This is the same value originally referenced in a previous **ddeadvise** command.

Comments

A **ddeunadvise** operation can be tested with the **if** SUCCESS statement, returning true if the operation was successful and false if it failed.

See also

ddeadvise and ddeinit.

decrypt

Decodes an encrypted string.

decrypt strvar length

strvar	The string to decode. This variable will be updated with the decoded data.
length	The length of the string to **decrypt**.

Comments

The datakey value must be equal to the value used to **encrypt** the string to properly **decrypt** the string. It can be set with the **set aspect datakey** command.

See also

encrypt; set aspect datakey in "*Set and Fetch Statements*."

default

Provides for processing when no match is found among the **case** statements in a **switch** statement group.

default

Comments

If a **case** target is matched, the statements following it are executed until an **endcase** or **exitswitch** is encountered. If another **case** is encountered, it's skipped. Execution then falls through to the next commands following the matched **case** target. If a match is made with more than one **case** command, only the statements following the first matched **case** command will be executed.

The **default** command is optional, but there can be only one occurrence of it within a **switch** statement group. If a **default** isn't used and none of the **case** commands were matched, no action takes place at all.

➥ **default** *can occur anywhere within the* **switch** *statement.* **case** *statements occurring between* **default** *and its terminating* **endcase** *or* **exitswitch** *will be ignored.*

For further information on **default**, see the **switch** command.

See also

switch, case, endcase and exitswitch.

#define

Defines a macro name and the text to substitute for it during compilation. There are two forms of macros in ASPECT.

Simple text substitution:

#define name text

name	The macro name to be defined.
text	The text to be substituted for the macro *name* when processed by the *ASPECT Compiler.*

Argument/text substitution:

#define name([name[,name]...]) text

name	The macro name to be defined.
([name[,name] ...])	Zero or more arguments to be processed by the compiler with the substitution text.
text	The text to be substituted for the macro *name* when processed by the *ASPECT Compiler.*

Comments

Similar to macros in the "C" programming language, **#define** allows for textual substitution within a source program.

➥ *If the argument/text substitution macro form is used, any arguments specified must follow the standard ASPECT naming convention in the* **#define** *statement.*

Arguments must be comma-separated. No white space can appear between the last macro name and the opening parentheses, or it will be treated as part of the substitution text. For more information, refer to"*Macros*" on page 35.

See also

#elifdef, #elifndef, #else, #ifdef, #ifndef, #endif, and #undef.

delfile

A Deletes a specified file.

delfile filespec

Comments

The *filespec* may include a full directory path. If the path isn't specified, the current User Path is assumed. Wildcards are not allowed.

See also

copyfile, rename and isfile.

dial

A Calls one or more entries in your *Connection Directory.*

dial dialclass [GROUP] name [name...][CONNECTALL]

dialclass	A keyword representing a *Connection Directory* entry class. Valid keywords are: DATA, MAIL, FAX, FTP, NEWS, TELNET, VOICE, and WWW.
GROUP	Dial the group(s) specified.
name	A string identifying the entry or group name to dial.
name...	Optional additional entry or group names to dial. Up to 12 names in all can be specified.
CONNECTALL	Initiates a continuous attempt to connect to all selected *Connection Directory* entries. When a connection is made and then terminated, the dialing process continues to the next entry in the list and resumes dialing. If CONNECTALL is not used, any successful connection will terminate the dialing queue.

Comments

dial launches a dialing sequence and immediately returns control to the script. To force the script to pause for a connection create a loop testing $DIALING for 0. When $DIALING is set to 0, $DIALCONNECT will indicate the entry responsible for the connection.

dial searches for groups or entry names, but not both at once. The group or entry names must match exactly the information contained in the associated *Connection Directory* field, including upper- or lowercase. The **dialfind** command can be used to find either an entry or group name based on a partial name.

All entry names are placed in the dialing queue. If a group name is selected, then all entries in that group are placed in the dialing queue. The $DIALQUEUE system variable will return the number of entries remaining in the list to be dialed.

$DIALSELECT reflects the name of the entry currently being dialed, or the last-selected entry.

The $SCRIPTMODE system variable can be checked to determine if the script was executed from a *Connection Directory* entry. $DIALENTRY indicates the name of the entry that launched the script.

➥ *The* **dial** *command will fail if a dialing operation or a file transfer is already in effect.*

See also

dialadd, dialcancel, dialclass, dialcount, dialcreate, dialdelete, dialfind, dialinsert, dialload, dialname, dialnumber, dialsave and dialstats; set aspect dialingbox, the set dialdir and set dialentry command families in "*Set and Fetch Statements*." Also look at $DIALCONNECT, $DIALENTRY, $DIALSELECT, $DIALQUEUE, $SCRIPTMODE and $DIALING in "*System Variables*."

dialadd

A Adds a new entry or group name to the currently loaded Connection Directory.

dialadd dialclass [GROUP] name

dialclass	A keyword representing a *Connection Directory* entry class. Valid keywords are: DATA, MAIL, FAX, FTP, NEWS, TELNET, VOICE, and WWW.
GROUP	Create a new group. If this keyword is not specified, a new entry will be created.
name	A string which identifies the name of the new entry or group.

Comments

The *dialclass* keyword determines the initial dialing class in which the entry will appear. The entry will automatically appear within other classes when the appropriate connection information is entered.

➥ *When Connection Directory changes are complete, the* **dialsave** *command should be issued to save them.*

The **dialadd** command will fail if the *Connection Directory* is displayed at its execution time. It will also fail if the provided entry or group name is a null string or if it contains more than 40 characters including the null terminator.

See also

dialcreate, dialdelete, dialinsert and dialsave; the set dialentry command family in "*Set and Fetch Statements*."

dialcancel

Cancels a current dialing operation.

dialcancel [CURRENT]

CURRENT	Cancels the current dialing attempt only, and removes that entry from the queue. If CURRENT is not specified, the entire dialing operation is canceled.

Comments

Essentially, **dialcancel** has the same effect as the user pressing <Esc> or the ***Cancel*** button as the modem is dialing. **dialcancel** only affects a voice or data dialing attempt. Use the **faxcancel** command to cancel a fax dial.

See also

dial, dialnumber and faxcancel; set aspect dialingbox in “*Set and Fetch Statements*.”

dialclass

A Returns a value indicating the entry class(es) for the specified entry.

dialclass name intvar

name	The name of the entry to test. The group or entry names must exactly match the information contained in the associated *Connection Directory* field, including upper or lower case. The **dialfind** command can be used to find either an entry or group name based on a partial name.
intvar	Will be set to a value indicating entry class(es) if **dialclass** was successful.

Comments

The following values can be tested:

1	Data entry
2	Fax entry
4	Voice entry

8	Telnet entry
16	FTP entry
32	Web entry
64	Mail entry
128	News entry

If an entry contains information for more than one type of entry, the resulting integer will be the sum of those entries. For example, if you have an entry with data, fax and voice information, **dialclass** will return a value of 7.

When executed with $DIALSELECT, $DIALCONNECT, or $DIALENTRY, the returned *intvar* value will indicate a single class for the named entry, even though it may be a number of other classes as well.

See also

dialadd, dialdelete, dialfind and dialstats; the set dialentry command family in "*Set and Fetch Statements*."

dialcount

Returns the number of entries, groups, or entries within a group for a particular dialing class.

dialcount dialclass [GROUP [name]] intvar

dialclass	A keyword representing a *Connection Directory* entry class. Valid keywords are: DATA, MAIL, FAX, FTP, NEWS, VOICE, and WWW.
GROUP	Search a group for entries of the requested dialing class.
name	The name of a particular group to search. The group name must match exactly the information contained in the group name field, including upper- or lowercase. The **dialfind** command can be used to find a group name based on a partial name.
intvar	This variable will contain the number of entries, groups, or entries within a group that matched the dialing class specified.

Comments

If GROUP is not specified, the total number of entries will be returned. If GROUP is specified without a *name* argument, the total number of groups is returned. If a *name* argument is specified with GROUP, the total number of entries within that group is returned.

See also

dialfind, dialstats and dialclass; the set dialentry command family in "*Set and Fetch Statements*."

dialcreate

A Creates a new *Connection Directory* file.

dialcreate filespec

Comments

A *Connection Directory* file can be created with any filename. If a specific path is not supplied, **dialcreate** places the new file in the local task path. The **dialcreate** command will fail if the *Connection Directory* is displayed when it executes.

See also

dialadd, dialdelete, dialfind, dialinsert, dialload and dialsave.

dialdelete

Removes a *Connection Directory* entry, group, or an entry within a group.

dialdelete dialclass {GROUP name} | {ENTRY name} | {groupname entryname}

dialclass	A keyword representing a *Connection Directory* entry class. Valid keywords are: DATA, MAIL, FAX, FTP, NEWS, TELNET, VOICE, and WWW.
GROUP	Delete the specified group. The entries contained within the group are not deleted, but rather are made available outside of the deleted group if they are not a member of any other group.
name	The name of the group to delete.
ENTRY	Delete the specified entry.

name	The name of the entry to delete.
groupname entryname	Within the group specified, delete the specified entry. The entry will not be deleted, just removed from the group.

Comments

When an entry name is deleted, it is only deleted from the indicated class. If it exists in any of the other classes, it is not deleted from them.

The group or entry names must match exactly the information contained in the associated *Connection Directory* field, including upper- or lowercase. The **dialfind** command can be used to find either an entry or a group name based on a partial name.

➥ *When Connection Directory changes are complete, the* **dialsave** *command should be issued to save them.*

The **dialdelete** command will fail if the *Connection Directory* is displayed, or if a dialing operation is already in effect when it executes.

See also

dialadd, dialcreate, dialinsert, dialfind and dialsave; the set dialentry command family in “*Set and Fetch Statements*.”

dialfind

A Finds the entry name and/or index associated with a specified name in the specified class.

dialfind {dialclass [GROUP] name [EXACT]} | NEXT [strvar]

dialclass	A keyword representing a *Connection Directory* entry class.
GROUP	Search for a group defined with the specified name.
name	The full or partial text used to search the *Connection Directory.*
EXACT	Causes exact string matching instead of the default substring case-insensitive matching.
NEXT	Searches for a subsequent *Connection Directory* match.
strvar	An optional variable assigned the full name of the matching entry or group.

Comments

A partial name can also result in a match. **dialfind** will find the first entry or group name whose leading characters match the characters in the partial name without regard for character case.

See also

The set dialentry command family in “*Set and Fetch Statements*.”

dialinsert

A Inserts a *Connection Directory* entry into an existing group.

dialinsert dialclass groupname entryname

dialclass	A keyword representing a *Connection Directory* entry class. Valid keywords are: DATA, MAIL, FAX, FTP, NEWS, TELNET, VOICE, and WWW.
groupname	The name of the group to receive the new entry.
entryname	The name of the new entry.

Comments

The group specified must exist before an entry can be inserted. To create a group, use the **dialadd** command. The group or entry names must match exactly the information contained in the associated *Connection Directory* field, including upper- or lowercase. The **dialfind** command can be used to find an entry or group name based on a partial name.

➥ *When Connection Directory changes are complete, the* **dialsave** *command should be issued to save them.*

The **dialinsert** command will fail if the *Connection Directory* is displayed at its execution time.

See also

dialadd, dialdelete, dialfind and dialsave; the set dialentry command family in “*Set and Fetch Statements*.”

dialload

A Loads a different *Connection Directory.*

dialload filespec

Comments

If a full path is not specified, the local task path is used by default. The *filespec* can contain the name of any valid *Connection Directory* file. The **.dir** extension is not required, but if the filename has an extension other than **.dir** it must be specified.

The **dialload** command will fail if the *Connection Directory* is displayed at its execution time.

See also

dialcreate and dialsave.

dialname

A Enumerates entries, groups, or entries within a group.

dialname dialclass[GROUP [name]] index strvar

dialclass	A keyword representing a *Connection Directory* entry class. Valid keywords are: DATA, MAIL, FAX, FTP, NEWS, TELNET, VOICE, and WWW.
GROUP	Access groups containing the desired dialing class within the directory. If the *name* parameter is used, access entry names within the specified group.
name	The name of the group containing the desired entry. The group or entry names must match exactly the information contained in the associated *Connection Directory* field, including upper or lower case. The **dialfind** command can be used to find either an entry or group name based on a partial name.
index	The zero-based *index* of the entry, group, or entry within a group to search.
strvar	Will contain the name of the entry, group, or entry within a group if the search is successful. If **dialname** is not successful, *strvar* will be null.

Comments

If the GROUP keyword is omitted, then entry names in the *Connection Directory*, including those within groups, will be enumerated.

dialname returns FAILURE if no entry, group, or entry within a group was found for the given *index* value. To determine the number of entries, the number of groups, or the number of entries within a group, use the **dialcount** command.

See also

dialadd, dialcount, dialdelete, dialinsert, dialfind and dialload; the set dialentry command family in "*Set and Fetch Statements*."

dialnumber

A Calls a specified telephone number.

dialnumber dialclass string

dialclass	A keyword representing a *Connection Directory* entry class.
string	A valid telephone number. It can optionally contain embedded dialing codes.

Comments

dialnumber accepts only a single telephone number. For multiple *Connection Directory* entry numbers, use the **dial** command.

➥ ***dialnumber*** *will fail if a dialing operation or a file transfer is already in effect.*

See also

dial, dialcancel and dialload; $DIALING in "*System Variables*" and the set dial command family in "*Set and Fetch Statements.*"

dialogbox

Marks the beginning of a dialog box statement group. A dialog box is a customized data entry window that can be controlled by the ASPECT script.

dialogbox id left top width height style [title] [CALL name] [PARENT id]

id	An integer variable or constant, greater than or equal to zero, specifying the id for this dialog box. This value is used by other dialog box commands to control, update, or destroy the dialog box.

left top width height	Constant integer parameters specifying the initial position and size of the dialog box. Coordinates are relative to the parent's client window. If no parent is specified, the client window defaults to Procomm Plus's.
style	An integer constant, defining the behavior and appearance of the dialog box. Add any combination of the following values:
0	A normal modal dialog box. A modal dialog box prevents the user from accessing its owner, whether it is another dialog box or Procomm Plus's main window. Modal dialog boxes are typically used when input is required before program flow can be continued.
1	A dialog box centered with respect to its parent.
2	A moveable dialog box with caption.
4	A modeless dialog box, which allows the user to access the dialog box's owner, whether the owner is another dialog box or Procomm Plus's main window. Modeless dialog boxes will remain intact while the user performs other tasks in other windows. Typically, modeless dialogs are used when the input is independent of other tasks or actions in Procomm Plus.
8	Trap <Esc>, <Alt-F4> and **Close** events (event -1). This is used to prevent the user from destroying the dialog box without script intervention. If not trapped, the <Esc> and <Alt-F4> keys, and the ***Close*** command on the dialog box's system menu will automatically close the dialog box, and will not save any file-related control contents to disk. This includes the **flistbox**, **feditbox** and **dirlistbox** controls.
16	Remove ***Close*** command from system menu. This is often used in conjunction with style bit **8** to let the user know that closing the dialog must be done specifically through control actions. Removing the ***Close*** command from the system menu with value **16** does not prevent <Alt-F4> from closing the dialog box. The <Esc>, <Alt-F4>, and ***Close*** events must be specifically trapped using value **8** to prevent closure.
32	Hide dialog box until shown with the **dlgshow** command. Hiding the dialog box allows the script to modify, **enable** or **disable** controls before the user sees the dialog. This style is often used with the CALL *name* option, discussed below. A **dlgshow** command must be issued to display a dialog box created with this style option.

64	Suspend script execution until the dialog box is destroyed. This style is used to prevent the script from executing beyond the **enddialog** command until the dialog box is destroyed. In order to process dialog events, a **when** command should precede the **dialogbox** command, or the optional CALL *name* form should be used to create a polling loop, to handle dialog events. If no events are processed, either by a **when** or a CALL *name* procedure, the script will resume execution when the dialog box is destroyed, at which time the **dlgevent** command can be called to determine how the dialog was destroyed. In this case, **dlgevent** will return either a control *id* value or -1. Only one event will be available from the event queue, since the dialog box has already been destroyed.
128	Update variables when event is read from queue. This means that an ASPECT result variable associated with a dialog box control will not be updated immediately when the control is accessed and changed. Instead, the variable will be updated when the event is read from the event queue. Typically, this style option is used when the script intends for one control to interact with another control, yet needs to work with "old" information before fresh input is handled or variable contents are allowed to change. The event queue is limited in size. If an event occurs when the queue is full, the event that is “bumped” from the head of the queue will cause its associated variable to be updated automatically. A **dlgflush** command, when used with the optional SAVE argument, will also cause all variables associated with the flushed events to be updated. An **OK** button, when selected, closes a dialog box and updates all associated variables within the dialog, regardless of the status of the event queue. A ***Cancel*** button does not update associated variables. **dlgevent** *id flush* can be used to flush the dialog box event queue. If dialog box style **128** is specified, use the *save* argument to update variables whose control events remain in the queue.

256	Version 1.0 compatibility. This style allows a dialog box to retain certain behaviors that it would have had in Procomm Plus version 1.0. This style should not be used normally, as it is intended only for dialog boxes converted from Procomm Plus 1.0 to 2.0. It supports behaviors that could not be converted directly, and relies on *id* values as they were defined in version 1.0. These behaviors include saving all file-related controls whenever they lose input focus, letting partial paths resolve to the current working directory rather than the Aspect path, centering the dialog, if specified, horizontally with respect to Procomm Plus's main window, rather than both horizontally and vertically with respect to the parent window, not generating a dialog event when an <Esc>, <Alt><F4> or the system menu **Close** command is trapping these events, setting a single tab stop out of view in a **listbox**, and returning a sequence value for an **option button** rather than its *id*.
title	Text displayed in the title bar of the dialog box. This parameter is required for a style 2, Moveable with caption dialog box, but it can be a null string if no text is desired. A string constant or a global string variable can be used.
PARENT id	This parameter is a dialog box id that identifies the owner of the current dialog box. When an owner dialog box is destroyed, its child dialog boxes will be automatically destroyed as well. Either a constant or global integer can be used for the *id* value, but it must be greater than or equal to zero. If PARENT *id* is not specified, Procomm Plus's main window is assumed by default.
CALL name	This parameter identifies a procedure to be called when the dialog box is created. It is often used with style **32** dialogs in order to take control of the dialog box before it is displayed. It can also be used to set up a polling loop to process all dialog box events, achieving in effect the same results as style **64** dialog.

Comments

dialogbox groups are highly structured. All dialog box group commands in a script must appear between **dialogbox** and **enddialog** commands, and only dialog box commands may be used within a **dialogbox** group.

If another dialog box is created using the same *id* as an existing dialog box, the existing dialog will automatically be destroyed, just as if it were canceled by the user.

Local variables may be used within the **dialogbox** statement or in any dialog box control statement. However, the dialog box will be automatically destroyed when the procedure that defines the local variables returns.

Up to 255 total controls can be used within a **dialogbox**. Each control within a **dialogbox** must have a unique *id* value, ranging from 1 up to 999. The *ASPECT Compiler* will detect duplicate *id* values and return an error message.

The size of the dialog box and all controls placed within it are positioned and sized in Dialog Box Units, or DBUs. For more information on DBUs, see“*Dialog Box Units*” on page 48.

If a *strvar* is specified for the caption parameter, it can be changed and updated using **dlgupdate** with a starting id of 0.

The use of **#define**d macros within the **dialogbox** group is allowed. The *Dialog Editor* cannot interpret macros, however. To use macros within a **dialogbox** group, simply insert them after the **dialogbox** is complete.

Controls and graphics in the User window are painted in the order their commands appear in the **dialogbox** group, and not in the numeric order of their *id*s.

metafile and **bitmap** commands should be listed ahead of any controls you wish to place over them. If a **dlgupdt** command is used to repaint a control that has other controls displayed over it, those control *ids* should be updated as well.

Access to dialog boxes, and their controls, can be controlled with the **enable** or **disable** commands.

See also

dlglist, dlgctrlwin, dlgdestroy, dlgevent, dlgexists, dlgsave, dlgshow, dlgupdate, dlgwin, dlgwinctrl, enable, disable, bitmap, metafile, icon, iconbutton, listbox, flistbox, combobox, fcombobox, editbox, feditbox, text, ftext, dirpath, dirlistbox, groupbox, radiogroup, radiobutton, endgroup, pushbutton and enddialog.

dialsave

A Saves the current *Connection Directory* to disk.

dialsave [filespec]

filespec	Optional name or complete path for the directory file, if different from the *Connection Directory* path. Specifying a filename which is different from the currently loaded *Connection Directory* file is the same as using the *Connection Directory*'s ***Save As...*** menu item. Any filename and extension can be used. However, the use of a **.dir** file extension is highly recommended. No default extension is assigned if one is not specified. The local task path is assumed if no path is specified.

Comments

If the directory is saved under a different filename, that filename becomes the currently loaded *Connection Directory* file. The **dialsave** command will fail if the *Connection Directory* is displayed at its execution time.

See also

dialadd, dialcreate, dialdelete, dialinsert and dialload; the set dialentry command family in "*Set and Fetch Statements*."

dialstats

A Retrieves the last date and time called, total connect time, and total calls made for the specified *Connection Directory* entry.

dialstats name {longvar longvar longvar} | CLEAR

name	The name of the *Connection Directory* entry to access. The group or entry names must match exactly the information contained in the associated *Connection Directory* field, including upper- or lowercase. The **dialfind** command can be used to find either an entry or group name based on a partial name.
longvar	Will be set to the last date and time on which a connection was made with this entry, using the *timeval* format as is used by $LTIME. The **ltimestrs** or **ltimestring** command can be used to split this value into date and time strings.

longvar	Will be set to the total connect time for this *Connection Directory* entry, in seconds. It can be converted into a readable time string using the **ltimeelapsed** command.
longvar	Will be set to the total number of connections made to this entry.
CLEAR	Resets the statistics for this *Connection Directory* entry. This is the same as a user pressing the ***Clear*** button in the *Connection Directory* **Statistics** dialog.

See also

dialname, ltimestrs, ltimeelapsed and ltimestring.

dir

A Displays a standard file listing and optionally returns a filename selected by the user.

dir filespec [strvar]

strvar	An optional string variable containing the file selected by the user.

Comments

dir displays a standard file selection dialog with the title bar of **Directory File List**.

The *filespec* argument in the **dir** command can contain several file types to display by default. This is done by separating each file type from the previous type with a semicolon. For example, to display all **.txt** and **.doc** files, the command would be formed as:

```
dir "*.txt;*.doc"
```

An optional path may precede the first file type.

dirlistbox

Adds a file selection list box to a Windows dialog box. The list box displays the files, sub-directories and disk drives available.The parent directory of the current directory is also indicated by the usual double periods.

dirlistbox id left top width height filespec [filetype] {MULTIPLE filespec} | {SINGLE strvar} [dirpathid] [HSCROLL] [SORT]

id	A unique integer constant which identifies this control. This value will be stored in **dlgevent**’s target variable when a selection is made in the **dirlistbox**.
left top width height	Integer constants specifying the position and size of the **dirlistbox**. Coordinates are relative to the dialog box window, specified in Dialog Box Units or DBUs. For more information on DBUs, see“*Dialog Box Units*” on page 48.
filespec	A string variable or constant containing the drive and directory path of the files to be displayed in the list box. If a *filespec* is included, at least one DOS wildcard character “*” or “?” must be included in the *filespec*. If a path is not included in the *filespec*, the Aspect Path is assumed.
filetype	A optional string containing characters which represent file attributes.
MULTIPLE filespec	*Filespec* is a string variable or contstant naming the file created with the multiple files chosen from the displayed list. The file list is carriage return/linefeed-separated. Each entry in the list will be a fully-qualified filespec. Entries in a pre-existing selection file will be automatically selected within the **dirlistbox** on start-up, if the paths in the entries is identical to the current path specified in the file list *filespec*. If a path is not included in the *filespec*, the Aspect Path is assumed. When MULTIPLE *filespec* is used, a **dlgsave** or **dlgdestroy** command should be used to write the selected file names to the disk file. Otherwise, no file will be created.
SINGLE strvar	Will contain the filename selected by the user. *strvar* can be pre-initialized with a filename, allowing it to already be selected upon entry. Remember, this returns a fully qualified *filespec,* not just the filename.
dirpathid	An integer constant specifying an optional **dirpath** id value. The associated **dirpath** will be updated with the current directory path selected in the **dirlistbox**.
HSCROLL	If specified, places horizontal scrollbars on the **dirlistbox**.

SORT	If specified, causes the file and directory names to be sorted alphabetically.

Comments

dirlistbox can only occur within a **dialogbox** group. It allows the user to change directories or drives by double-clicking on the desired directory or drive name.

The **dirlistbox** command accepts an optional *filetype* string which can be used to limit the search pattern to files with specific attributes. The string may be composed of characters representing attribute types: ’***R***’ for read-only, ’***H***’ for hidden, ’***S***’ for system, ’***A***’ for archive, ’***D***’ for directory, and ’***I***’ for drive.

By default, the search always includes normal files, which are files without attributes or files with read-only and/or archive attributes. Directory names and drives are also normally included. For example, if you specify “***HS***” for search attributes, all files with no attributes, read-only or archive attributes will also appear in the resulting list, in addition to those files having hidden or system attributes.

The character ’***X***’ may be included to override the default searching method. This indicates that the search is limited exclusively to those files matching one or more of the specified attributes. A list of directory names can be created by specifying “***DX***” as the search string; only drives will be displayed if “***IX***” is used.

In Windows 95, the **dirlistbox** command will only display short file names. *ASPECT* will automatically convert a long filename to its short filename format in order to make a listbox selection during initialization or update of the **dirlistbox** control. To obtain the long filename format for a **dirlistbox** selection, simply issue a **findfirst** command with the selected item, and then read the $FILENAME system variable for the result.

In Windows NT, long filenames are displayed in a **dirlistbox** control, and selections must be made using the long filename format. To obtain a long filename from a short filename, issues a **findfirst** command, and read the $FILENAME result.

If either directories and/or drives are displayed exclusively in a **dirlistbox**, then the control *id* will generate dialog events whenever a drive or directory is selected. If no default button is present, then when a directory is double-clicked, the control will change directories and re-initialize its contents with the sub-directories within the directory that was double-clicked. Double-clicking a drive has no effect when a default button is not present in a drive-exclusive listing.

When the user clicks on a particular filename, **dirlistbox** places that filename into *strvar* if SINGLE is specified. If the user double-clicks on a particular filename, **dirlistbox** assumes that the selection(s) are complete, and will assign the *strvar* to the selected SINGLE file, or place only the double-clicked selection into the specified MULTIPLE *filespec*, and select the dialog box's **default** button if one is specified.

If a drive or directory is double-clicked, and there is a **default button** or **iconbutton** present, then that **default** button will be automatically selected.

See also

dialogbox, dirpath, dlgevent, dlgsave, dlgdestroy, dlgupdate, iconbutton and pushbutton.

dirpath

Displays the current disk drive and directory for a **dirlistbox** control within a dialog box.

dirpath id left top width height [strvar]

id	A unique integer constant which identifies this control.
left top width height	Integer constants specifying the position and size of the **dirpath**. Coordinates are relative to the dialog box window in DBUs. For more information on DBUs, see "*Dialog Box Units*" on page 48.
strvar	If specified, *strvar* will contain the **dirpath** contents.

Comments

If referenced in a **dirlistbox** command, **dirpath** automatically provides a display of the currently-selected path in that **dirlistbox**. An event will be generated whenever the contents of the **dirpath** are changed, but only if the optional string result variable is specified.

See also

dialogbox, dlgevent and dirlistbox.

disable

Disables dialog box controls, dialog boxes, menu selections, or windows. Items that are disabled are grayed and cannot be selected.

disable DIALOGBOX dlgid

dlgid	The unique id of the dialog box to **disable**, including its controls. If the dialog box has child dialog boxes, they will be disabled as well.

disable DLGCTRL dlgid startid [endid]

dlgid	The unique id of the dialog box containing the control to be **disable**d.
startid	The id of the first or only control to be **disable**d.
endid	If specified, allows a range of dialog box controls to be **disable**d, from *startid* to *endid*.

disable ASPMENU startid [endid]

ASPMENU	**disable**s a menu item generated by an ASPECT script.
startid	The id of the menu item to be **disable**d.
endid	If specified, allows a range of menu items to be **disable**d, from *startid* to *endid*.

disable PWMENU startid [endid] [PERMANENT]

PWMENU	**disable**s a menu item on the Procomm Plus menus.
startid	The id of the menu item to be **disable**d.
endid	If specified, allows a range of menu items to be **disable**d, from *startid* to *endid*.
PERMANENT	The menu item(s) will remain **disable**d after the script terminates.

disable WINDOW window

window	The id of the window to be **disable**d. This value can be obtained by accessing the $PWMAINWIN, $FOCUSWIN, $ACTIVEWIN, $MAINWIN, $POINTERWIN, and $USERWIN system variables. The **taskwin**, **dlgwin**, and **dlgctrlwin** commands also return a window id.

Comments

You may wish to **disable** certain items during the execution of a script to prevent the user from selecting them. **disable**d items can include any of the Procomm Plus menu items, any control in a **dialogbox** statement group, any menu item built by the script, or entire dialog boxes.

Use caution when disabling windows! If a window is **disable**d it no longer responds to keyboard or mouse input.

➥ *When disabling windows, dialog boxes, and dialog box controls, if the item you wish to disable has the focus, you should set the focus somewhere else prior to using the disable command. If you disable a window, dialog box, or dialog box control that has the focus, accelerators and other keystrokes will not be processed properly.*

The id values of Procomm Plus's menu items can be found in **pw4menu.inc**, installed by default in the ASPECT directory. This file can be **#include**d for reference in an ASPECT script.

A menu item that was generated by an ASPECT script can only be **enable**d or **disable**d when it is displayed. Procomm Plus's menu items, on the other hand, can always be affected, unless they were **disable**d by Procomm Plus itself.

➥ *The $ASPMENU system variable is used only for menu items created by an ASPECT script. It will not indicate when a Procomm Plus menu item has been selected. Pop-up menus and menubars cannot be directly enabled or disabled from within a script.*

See also

dialogbox, enable, menupopup, menuitem, winactivate, wintask, taskactivate and taskwin; $ACTIVEWIN, $PWTASK, $TASK and $ASPMENU in "*System Variables*."

disconnect

The operation of **disconnect** is dependent on the current mode. In *Data* mode, **disconnect** hangs up the phone. In *Web* mode, it discontinues loading of the current page, just like the ***Stop*** button on the *ActionBar*. In *FTP*, *Telnet*, *Mail*, and *News* modes, **disconnect** will terminate the connection with the server.

disconnect

Comments

disconnect is *not* a synonym for **hangup**.

See also

hangup.

diskfree

Returns the free disk space of the specified drive into a long variable.

diskfree disk longvar

disk	An integer number designating the drive; 0 indicates the current drive, 1 specifies drive A, 2 indicates drive B and so on up to 26 for drive Z.
longvar	Will contain the number of bytes free on the specified drive.

See also

dir, getdir and memavail; $USERPATH and $VOLUME in "*System Variables*."

dlgctrlwin

Returns the window id of a dialog box control.

dlgctrlwin id ctrlid intvar

id	The id of the dialog box which contains the specified control.
ctrlid	The id of the specified control.
intvar	Will contain the value of the window id of the specified control. This value may be used in any command that accepts a window id.

See also

dialogbox and dlgwinctrl; $FOCUSWIN in "*System Variables*."

dlgdestroy

Destroys a dialog box.

dlgdestroy id CANCEL|OK

id	The id of the dialog box to be destroyed.

CANCEL	If CANCEL is specified, the dialog box will be destroyed without updating any changes made to file-related controls.
OK	If OK is specified, the dialog box will be destroyed, but changes made to file-related controls will be saved to disk.

Comments

To save changes made to a file-related control before destroying the dialog, use the **dlgsave** command. File-related controls include **flistbox**, **feditbox**, and **fcombobox**.

Note that if the dialog box was created with style 128, Update variables when event is read, **dlgdestroy** *id* OK updates all variables whose events have not been retrieved with **dlgevent**.

See also

dialogbox, dirlistbox, dlgevent, dlgsave, fcombobox, feditbox and flistbox.

dlgevent

A Returns the id of a user action within a dialog box, or flushes the event queue for the specified dialog box.

dlgevent id [intvar | {FLUSH [SAVE]}]

id	The dialog box id whose EventID queue will be accessed.
intvar	If *intvar* is specified, it will return the value of the last dialog box EventID.
FLUSH	If FLUSH is specified, **dlgevent** will reset the EventID queue for the specified dialog box.
SAVE	Only valid with the FLUSH keyword. Causes variables associated with EventID values to be updated as the values are flushed from the queue.

Comments

dlgevent will set the *intvar* argument to a dialog box control ID if an event of that type has occured, to zero if no event is available, to -1 if the dialog box has been closed by <Alt><F4>, <Esc> or the menu **Close** command, or to -2 if the <F1> key was pressed for context-sensitive help. In addition, the *intvar* argument is set to -3 if the parent dialog box (if any) was closed via an OK button or destroyed with the OK flag in **dlgdestroy**.

➥ *A child dialog box is destroyed automatically when its parent is destroyed.*

The **dlgevent** *id* FLUSH SAVE is useful when the dialog box style value includes 128 (update variables when EventID is read). For more information, see the **dialogbox** command's style list.

dlgevent sets SUCCESS true when there are dialog events in the event queue. When the optional argument is not specified, an event is not read from the queue.

See also

dialogbox, dlgdestroy, dlgsave and dlgwin.

dlgexists

A Sets SUCCESS if the dialog box with the associated ID exists.

dlgexists id [intvar]

id	An integer vaiable or constant specifying the dialog box id. This value is used by other dialog box commands to control, update, or destroy the dialog box.
intvar	This optional integer variable is assigned the value 1 if the dialog exists, or 0 otherwise.

See also

dialogbox, dlgevent, and dlgdestroy.

dlglist

A Adds, removes, counts, retrieves, or finds items from a **listbox**, **flistbox**, **combobox**, or **fcombobox** control.

dlglist id ctrlid {INSERT string [integer]} | {REMOVE integer | string} | {SELECT integer | string [MULTIPLE]} | {UNSELECT integer | string } | {ITEMCOUNT intvar [intvar]} | {ITEMNAME integer strvar [intvar]} | {ITEMFIND string intvar [integer] [EXACT]}

id	The id of the dialog box containing the target list control.
ctrlid	The target list control id.
INSERT string	The specified *string* will be inserted into the target control's itemlist. If the optional *integer* is not provided, the item will be appended to the end of the list.

integer	If specified, *integer* is treated, as an index, determining where the string will be inserted in the itemlist. A value of -1 will append the string to the end, while a 0 will insert the string at the top of the list. If the index location is greater than the number of items already in the list, the item will be appended to the end of the list.
REMOVE string	The specified *string* will be removed from the target control's itemlist.
integer	If specified, *integer* is treated as an index, determining where the string will be removed from in the itemlist. A value of -1 will remove all of the item in the list. If the index location is greater than the number of items already in the list, the item will be removed from the end of the list.
SELECT	Specifies strings to be highlighted in the list box.
integer	If specified, *integer* is treated, as an index, determining which item is to be selected.
string	The string of the item to be selected.
MULTIPLE	If specified, this allows ASPECT to highlight multiple selections in a multi-select list box. It is ignored for combo boxes and single-select list boxes.
UNSELECT	Specifies strings to be unselected in the list box.
integer	If specified, *integer* is treated as an index, determining which item is to be unselected. If not specified, all selections are removed.
string	The string of the item to be unselected.
ITEMCOUNT intvar	Returns the total number of items within the specified list or combo box. This is useful if you're enumerating a list with an ITEMNAME operation.
intvar	If specified, *intvar* will be assigned the number of items selected in the list or combo box, although combo boxes can have at most one item selected.
ITEMNAME integer	Retrieves the displayed text associated with a list or combo box entry. *integer* is a zero-based index.

strvar intvar	*strvar* will be assigned the text displayed in the list or combo box entry associated with the specified index. *intvar* will be set to 1 if the entry is selected, 0 otherwise.
ITEMFIND string intvar	Searches a list or combo box for a particular item or entry matching the specified *string*. The search is case-sensitive. *intvar* returns the index of the entry satisfying the search string.
integer	Specifies the index after which the search will begin. This allows you to continue searches beyond the first item found. Otherwise, the list is searched from the top.
EXACT	If specified, only exact matches for the search *string* will be returned.

Comments

dlglist returns SUCCESS if the string was successfully added or removed from the target control's item list, FAILURE otherwise.

dlglist does not affect the contents of the associated control's list variable or list file contents, only the displayed itemlist. If **dlglist** is used to add or remove list items, these changes are not written to disk with **dlgsave**. **dlglist** does not affect **dlgsave**, since **dlgsave** only affects result variables and files.

See also

dialogbox, dlgsave and dlgupdate.

dlgsave

Updates variables associated with dialog box controls and forces changes in **fcombobox**, **flistbox,** and **feditbox** controls to be written to disk.

dlgsave id ctrlid [ctrlid]

id	The id of the dialog box containing the control to be saved.
ctrlid [ctrlid]	The control id, or, if the second [*ctrlid*] is specified, the beginning of a range of control id's to be saved.

Comments

dlgsave is useful for saving file-related controls, or for reading current selections when a dialog box is created using style 128. For more information on dialogbox style-bits see the **dialogbox**

command. Typically, dialog box controls update their associated data fields as changes are made to them.

See also

dlgdestroy, dlgevent and dlgupdate.

dlgshow

Causes a hidden dialog box to be made visible.

dlgshow id

id	The id of the dialog box to be made visible.

Comments

dlgshow is only useful for showing dialog boxes of style 32, Hide until shown. Use **winshow** to display a dialog box hidden with **winhide**.

See also

enable, disable, dialogbox, dlgwin, winshow and winhide.

dlgupdate

Refreshes the display of a dialog box control, applying any changes made to the control’s associated variables.

dlgupdate id ctrlid [ctrlid]

id	The id of the dialog box containing the controls to be updated.
ctrlid [ctrlid]	The control id, or, if the second [*ctrlid*] is specified, the beginning of a range of control id’s to be refreshed.

Comments

If a *gstrvar* is used for the dialog box title, it can be refreshed using a start id of 0. **dlgupdate** has no effect on files that need to be written to disk.

See also

dialogbox and dlgsave.

dlgwin

Returns the window id of the specified dialog box.

dlgwin id intvar

id	The id of the dialog box desired.
intvar	Returns the window id value of the dialog box.

Comments

The value returned by **dlgwin** can be used with **winhide**, **winshow**, and other window-related commands to manipulate the state of the dialog box.

See also

dialogbox, enable, disable, winhide, winmove and winshow.

dlgwinctrl

A Converts a window id into the corresponding control id, within a given dialog.

dlgwinctrl id window intvar

id	The target dialog box id.
window	The window id to convert.
intvar	Returns the corresponding control id, if found.

Comments

Used in conjunction with $FOCUSWIN, **dlgwinctrl** allows a script to determine which dialog box control has the input focus. This is the opposite function of **dlgctrlwin**. FAILURE is set if no control id within a dialog corresponds to the specified window id.

See also

dialogbox and dlgctrlwin; $FOCUSWIN in "*System Variables*."

dllcall

A Calls a routine within a **.dll** module created to interact with ASPECT.

dllcall integer string [arglist]

integer	The ID of the DLL previously loaded with **dllload**.
string	A string identifying the routine to be executed within the DLL. The routine name must be identical to the exported name in the DLL. Note that this name is case-sensitive.
arglist	A maximum of 12 variables can be passed as arguments.

Comments

For more information on DLL specifications, review the Aspect samples available on the Procomm Plus CD. Only DLLs that have been written with the ASPECT DLL interface specification can be used with this command. Calling an arbitrary **.dll** may cause a critical application error to occur!

See also

dllload and dllfree.

dllfree

A Frees unneeded DLL memory resources.

dllfree integer

integer	The DLL id number returned by the **dllload** command that initiated the process you wish to free.

Comments

Generally, a script should free the DLL modules it has loaded into memory when they’re no longer needed. This frees up application memory that would otherwise be claimed while the script is still being executed. However, Procomm Plus will ensure that all DLL processes loaded by a script are removed at script termination, unless they were loaded using the PERMANENT option, in which case the DLL will not be removed until Procomm Plus terminates.

See also

dllcall and dllload.

dllload

A Loads the specified DLL module into memory.

dllload filename intvar PERMANENT]

filename	The name of the DLL to load. If no path is specified, the Aspect Path is assumed.
intvar	An integer id used by the **dllcall** and **dllfree** commands to reference the DLL module.
PERMANENT	If this keyword is specified, the DLL will remain loaded after the script terminates. It will unload, however, when Procomm Plus terminates.

Comments

Typically, the only DLLs that should be loaded are those written specifically to interface with ASPECT and the **dllcall** command. For more information, see the**dllcall** command.

See also

dllcall and dllfree; set aspect path in "*Set and Fetch Statements.*"

dllobject

A Identifies a graphic object within a User window to be updated by an associated DLL module.

dllobject id left top width height BACKGROUND | USERWIN

id	The control id for the DLL object. $OBJECT will be set to this value when the object is selected by the user.
left top width height	Integer parameters specifying the initial position and size of the DLL object in User Window Units, or UWUs. For more information, see"*User Window Units*" on page 48. Coordinates are relative to Procomm Plus's client window.

BACKGROUND | USERWIN	If BACKGROUND is specified, the object stays over the same spot on a background graphic. Specify USERWIN to “fix” the top left corner of the object in position relative to the upper left corner of the User window.

Comment

The **dllobj**-command family allows Procomm Plus to animate a graphic display with real time data. **dllobject** assigns an id and defines a position in the User window where the animated object will be displayed. **dllobjfile** identifies the DLL. **dllobjupdt** calls the update entry point in the DLL with the data used to animate the object. These are very powerful statements, but they require Windows-compatible DLL modules that have been designed for Procomm Plus.

See also

uwincreate, dllobjfile, dllobjupdt, objpaint, objremove, hotspot, bitmap, icon, iconbutton, pushbutton and metafile; $OBJECT in “*System Variables*”.

dllobjfile

A Specifies a file containing a graphic object to be accessed or updated with the **dllobject** command.

dllobjfile filespec

filespec	If a path is not specified, *filespec* defaults to the current Aspect Path.

See also

uwincreate, dllobject and dllobjupdt; $OBJECT in “*System Variables*.”

dllobjupdt

Allows a DLL module linked to a DLL object to update that object.

dllobjupdt id [arglist]

id	The identifier assigned by the user in the **dllobject** command.
arglist	A maximum of 12 variables can be passed as arguments.

See also

uwincreate, dllobject and dllobjfile.

dos

A Executes a DOS command or another program within a separate DOS window.

dos string [MINIMIZED | MAXIMIZED | HIDDEN] [intvar]

string	Any executable command as it would appear on the DOS command line.
MINIMIZED	Forces Procomm Plus to execute the command or program as a minimized icon.
MAXIMIZED	Executes the command or program in a full-screen DOS window.
HIDDEN	Forces Procomm Plus to execute the command or program with no visible DOS window.
intvar	An optional integer variable assigned the task id number of the DOS window. This task id can be used in the **taskactivate**, **taskexit**, **ddeinit,** and other commands.

Comments

If one of the optional parameters MINIMIZED, MAXIMIZED or HIDDEN is not used, the application runs using a default window size and position. The **dos** command is equivalent to using the ***Run*** command on the ***Start*** menu, with the string "COMMAND /C cmndname" as an argument. The DOS session uses the settings in **_default.pif**, unless a **.pif** file already exists for **command.com**.

If the path to the executable command is not specified, the program or external DOS command must be either in the current directory or in the directory specified in your DOS path. ASPECT will, however, change to the directory specified in $USERPATH before attempting to execute the **dos** command. Also, **command.com** must be in the directory from which you started your computer, typically the root directory of drive C or in the location specified by the COMSPEC environment variable.

dos can be tested with the **if** SUCCESS statement, returning false if **command.com** is not found or true if it is found, even if the command to be executed is invalid or fails to execute. The **run** command provides a similar function and provides better testing for DOS external commands and other programs.

If you use the **dos** command to execute a program or command that requires user input, be sure that the user is aware of this, since processing will halt until the required input is provided. The **dos** command will execute both standard DOS batch files and internal DOS commands. The **run** command only accepts **.com** and **.exe** programs.

See also

run, shell and taskexists.

editbox

Adds an editing box for text entry in an ASPECT dialog box.

editbox id left top width height strvar [strlength] [MASKED][MULTILINE]

id	The control id for the **editbox**, specified as an integer constant. This value will be reported by **dlgevent** whenever the **editbox** contents are changed.
left top width height	Integer constants specifying the initial position and size of the **editbox**. Coordinates are relative to the dialog box, specified in Dialog Box Units or DBUs. For more information on DBUs, see "*Dialog Box Units*" on page 48.
strvar	This variable is set to the contents of the edit field when the **editbox** loses focus. It can be pre-set to reflect a value before the dialog box is shown, or updated using **dlgupdate**.
strlength	An integer constant or variable, specifying the maximum number of characters that can be typed into the edit field. If the user attempts to type more than the maximum number of characters, an error beep sounds.
MASKED	If specified, an asterisk is displayed for each character typed into the field.
MULTILINE	If specified, allows text to be word-wrapped to the next line in the edit box. Note that the text length is still limited to 256 characters.

Comments

editbox can only occur within a **dialogbox** statement group. If the user is entering sensitive information, the MASKED option can be used to protect it. A dialog event is generated when the edit control contents have been changed and the control loses the input focus.

See also

dlgupdate, dialogbox, dlgevent, ftext, text and feditbox.

#elif

Allows the *ASPECT Compiler* to evaluate whether the block of code which follows is evaluated. For more information see the **#define** command.

#elif expression

expression	A constant value expression; that is, an expression that results in a constant value throughout the script.

See also

#define, #endif, #if

#elifdef

Allows conditional compilation by testing for the existence of a defined macro name after an **#ifdef**, **#ifndef**, **#elifndef,** or another **#elifdef** statement has failed. For more information, see the **#define** command.

#elifdef name

name	Macro name to test for existence.

See also

#define, #ifdef, #ifndef, #else, #elifndef and #undef.

#elifndef

Allows conditional compilation by testing for the non-existence of a defined macro name after an **#ifdef**, **#ifndef**, **#elifdef,** or another **#elinfdef** statement has failed. For more information, see the **#define** command.

#elifndef name

name	Macro name to test for non-existence.

See also

#ifdef, #ifndef, #elifdef, #else and #define.

## *else*

Executes an alternate set of commands when the expression specified in the associated **if** or **elseif** commands evaluate as false.

else

See also

if, elseif and endif.

#else

Causes the *ASPECT Compiler* to process an alternate set of commands when the macro tested by an associated **#ifdef**, **#ifndef**, **#elifdef** or **#elifndef** is undefined. For more information, see the **#define** command.

#else

See also

#define, #ifdef, #ifndef, #elifdef, #elifndef and #undef.

elseif

Enables multiple tests within an **if**/**endif** command block. **elseif** can be used instead of **if** ... **endif** statement groups.

elseif condition

condition	*Conditions* can include any numeric expression or any command that sets the SUCCESS and FAILURE system variables. The **not** keyword can precede an ASPECT command to logically negate the result of an ASPECT command. The “!” operator can be used to logically negate a numeric expression or the result of a user-defined function. If the expression evaluates to a non-zero value, then the condition was satisfied and is considered true, while a value of 0 indicates that the test was false.

See also

if, else and endif.

enable

Enables dialog box controls, dialog boxes, menu selections or windows.

enable DIALOGBOX dlgid

dlgid	The unique id of the dialog box to **enable**.

enable DLGCTRL dlgid startid [endid]

dlgid	The unique id of the dialog box containing the control to be **enable**d.
startid	The id of the control to be **enable**d.
endid	If specified, allows a range of dialog box controls to be **enable**d, from *startid* to *endid*.

enable ASPMENU|PWMENU startid [endid]

ASPMENU | PWMENU	Determines whether the menu item to be **enable**d was generated by Procomm Plus or an ASPECT script.
startid	The id of the menu item to be **enable**d. This must be a valid menu item id not a menu pop-up id. Menu pop-ups cannot be disabled.
endid	If specified, allows a range of menu items to be **enable**d, from *startid* to *endid*.

enable WINDOW window

window	The id of the window to be **enable**d. This value can be obtained using the **winactivate** and **wintask** commands.

Comments

enable cannot be used to enable something that Procomm Plus has disabled. It applies only to Procomm Plus menu items and dialog box controls created by a script.

➥ *Pop-up menus and menubars cannot be directly enabled or disabled from within a script.*

You may wish to re-enable **disable**d items during the execution of a script to allow the user to select them. These items can include any of the Procomm Plus menu items, any control in a **dialogbox** statement group, any menu item built by the script, or entire dialog boxes. Script menu items can only be **enable**d on the active menu bar and its pop-up items. Procomm Plus's menu items can always be affected, unless they were **disable**d by Procomm Plus itself. The active menu is always returned by the $PWMENUBAR system variable.

➥ *Two useful files are provided with* Procomm Plus *that can be* ***#include****d into your scripts. These files are* ***pw4menu.inc****, which defines all of* Procomm Plus*'s menu values, and* ***vkeys.inc****, which defines the virtual key codes as shown in "Virtual Key Codes" on page 603. Both of these files are installed to the default ASPECT directory.*

See also

disable, dialogbox, menupopup and menuitem; $ASPMENU and $PWMENUBAR in "*System Variables*."

encrypt

encrypts or encodes the target string.

encrypt strvar strlength

strvar	The string to **encrypt**; it will be updated with the encoded data.
strlength	The number of characters to **encrypt**.

Comments

The **encrypt** and **decrypt** commands work on "raw" data. For this reason, you must specify the length of the string to be encoded. Use caution if writing the **encrypt**ed data to a text file. Since the data may contain control characters or even nulls, it is not suitable for a Windows **.ini** file.

See also

decrypt; the set ASPECT datakey command in "*Set and Fetch Statements*."

endcase

Concludes the **case** or **default** command within the **switch** command phrase, and transfers control to the line following the **endswitch** statement.

endcase

Comments

Regardless of the number of **case** statements within a **switch** structure, there must be at least one **endcase** or the *ASPECT Compiler* will generate an error.

See also

case, default, exitswitch and switch.

#endcomment

Marks the end of a comment block. Commands and text which follow **#endcomment** are processed normally.

#endcomment

See also

#comment.

enddialog

Marks the end of a **dialogbox** statement group. For more information, see the **dialogbox** command.

enddialog

See also

dialogbox.

endfor

Concludes the **for** statement group. At this point, the loop variable is increased or decreased, and execution jumps to the top of the **for** command block. The **for** conditions are tested to determine whether another loop iteration is required.

endfor

See also

exitfor, for and loopfor.

endfunc

Terminates a function **call** block. See the **func** command for further information.

endfunc

Comments

Similar to procedure **call** blocks, **endfunc** implies a **return.** However, since functions return values, at least one **return** statement is necessary within the function statement block. If the **endfunc** command is encountered before a **return** command is executed, the **return** value is undefined.

See also

func and return.

endgroup

Concludes a **radiogroup** block within a **dialogbox** definition.

endgroup

See also

radiogroup, radiobutton and dialogbox.

## *endif*

Concludes the **if** statement group.

endif

See also

if.

#endif

Concludes a **#if** statement group. The *ASPECT Compiler* will process statements following the **#endif** command normally.

#endif

See also

#define, #elif, #elifdef, #elifndef, #else, #if, #ifdef, #ifndef and #undef.

endproc

Terminates a procedure block. **endproc** implies a **return** to the calling procedure. The **return** statement isn't required. For more information, see the **proc** command.

endproc

See also

proc and return.

endswitch

Terminates the **switch** statement group. For more information, see the **switch** command.

endswitch

See also

switch.

endwhile

Terminates the **while** statement group. Execution jumps to the associated **while** command, where the condition is tested to determine whether another loop iteration is required. For further information, see the **while** command.

endwhile

See also

exitwhile, loopwhile and while.

errormsg

Displays an error message dialog box with an exclamation point graphic and accompanying text.

errormsg string | [formatstr arglist]

string	A string of up to 256 characters.
formatstr arglist	A string of up to 256 characters containing up to 12 format specifiers followed by a list of arguments. The arguments must be listed in the order they are specified within *formatstr.*

Comments

A default beep sound accompanies the message. While the dialog is displayed, the script suspends operation until the user clears the dialog by pressing <Enter>, <Esc> or the ***OK*** button.

See also

sdlgmsgbox, usermsg, termmsg and statmsg.

execute

A Executes another ASPECT script. Control returns to the parent script upon completion.

execute filespec [SHARED]

filespec	Any valid script filename. If a path is not included, the current Aspect Path is used by default.
SHARED	Causes the **execute**d script to inherit the parent script's current ASPECT environment settings, rather than the default settings.

Comments

If an extension of **.wax** is specified, the source file is re-compiled automatically if the compiled script doesn't exist. If the extension is omitted, **execute** will first search for a compiled script with the **.wax** extension. If found, **execute** will compare the file date and time stamp to the associated source file, if any. If the source file is newer, the script will be re-compiled automatically. If a compiled script matching the *filespec* can't be found, **execute** will attempt to compile a corresponding **.was** file. If a **.was** file is specified, the *ASPECT Compiler* will always attempt compilation. Only the predefined variables, S0-S9, I0-I9, L0-L9 and F0-F9 can be used to pass values to an executed script. User defined variables are not accessible between scripts.

The **execute** command will fail if the specified *filespec* doesn't exist or can't be compiled.

The $SCRIPTMODE system variable can be tested to determine whether the currently executing script was spawned from another script. $SCRIPTMODE will be set to 1 if the script was **execute**d. The $PARENTFILE system variable will contain the name of the script that will resume execution following the termination of the spawned script. For more information, see "*Script Spawning and Chaining*" on page 55.

See also

chain; set aspect path in "*Set and Fetch Statements*" and $PARENTFILE and $SCRIPTMODE in "*System Variables*."

exit

Terminates the executing script file. If the current script was spawned, or is the child of another script, control returns to the parent script.

exit [integer]

[integer]	An optional code returned to the parent script if the current script was spawned. The value can be read from the system variable $EXITCODE.

Comments

ASPECT is an excellent housekeeper, "tidying up" when a script terminates. However, a script should follow good programming practices, closing files and releasing memory and DLLs allocated by the script.

See also

halt and pwexit; $EXITCODE, $PARENTFILE and $SCRIPTMODE in "*System Variables*."

exitfor

Continues execution with the command following an **endfor** command. For more information see the **for** command.

exitfor

See also

endfor, loopfor and for.

exitswitch

Transfers control from within a **case** or **default** statement group to the line following the **endswitch** command. **exitswitch** is similar to the “C” language break command.

exitswitch

Comments

An **exitswitch** may only occur within a **case**/**default** block. If an **exitswitch** command is executed, then all commands between the **exitswitch** and **endcase** will not be processed.

See also

default, switch and case.

exitwhile

Terminates a **while** command block and continues execution with the command following the **endwhile**.

exitwhile

See also

endwhile, while and loopwhile.

exitwindows

A Terminates all executing tasks and exits Windows with the specified action.

exitwindows LOGOFF | REBOOT | SHUTDOWN

LOGOFF	Logs the user off, shuts down all applications except Procomm Plus, and displays the logon dialog.
REBOOT	Causes the computer to perform a cold boot on exiting Windows.
SHUTDOWN	Causes the computer to shutdown on exiting Windows.

Comments

SUCCESS and FAILURE are set to indicate whether all tasks actually terminated. Only testing for FAILURE is relevant. Under Windows 95, **exitwindows logoff** will shut down all applications running at the same “level” as Procomm Plus, but Procomm Plus will not be shut down. Upon successful completion of this command, you can issue a **pwexit** to shut down Procomm Plus. Under Windows NT, Procomm Plus will also shut down. In either case, the logon dialog box is displayed.

See also

taskactivate, taskexists, taskexit, taskname and taskwin.

faxcancel

Cancels the current fax operation.

faxcancel index | string | CURRENT

index	The option set index of a fax connection as listed in the ***Current Fax Connection*** list.
string	The name of the fax modem in the ***Current Fax Connection*** list.
CURRENT	The current modem/connection selected in *Setup*.

Comments

faxcancel only affects a fax dialing attempt. The **dialcancel** command can be used to cancel a data or voice dialing attempt.

See also

dialcancel and faxsend.

faxlist

A Enumerates the faxes displayed in the received or scheduled fax lists.

faxlist RECEIVED|SCHEDULED index strvar [intvar]

RECEIVED	If specified, the list of received faxes will be processed.
SCHEDULED	If specified, the list of scheduled faxes will be processed.
index	A zero-based index, specifying the fax list entry to process.

strvar	If the RECEIVED fax list is being processed, *strvar* will contain the name and extension, **.fax** or **.bft,** of the specified fax file. This file will always reside in the received fax directory, as specified in *Setup*. If the SCHEDULED fax list is being processed, and the scheduled fax is comprised of a single file, the name and extension, **.fax** or **.bft,** of the fax file will be returned. If the scheduled fax was comprised of more than one file, the string returned will be "***<<multiple>>***". If the scheduled fax was comprised only of a cover sheet, the string returned will be "***<<coversheet>>***". If the scheduled fax was a fax polling operation, the string returned will be "***<<remote poll>>***".
intvar	If ***RECEIVED*** faxes are being processed, *intvar* will contain either a 0, new **.fax** or **.bft,** 1, viewing was last action on a **.fax** file, 2, printing was last action on a **.fax** file, or 3, **.bft** file has had its contents extracted and saved. If SCHEDULED faxes are being processed, *intvar* will contain the number of attempts made to send the fax successfully.

See also

faxview and faxremove.

faxmodem

A Determines whether a modem is fax-capable.

faxmodem index | string [intvar]

index	The index number of the modem in the modem table.
string	The name of the modem in the modem table.
intvar	An integer variable that returns 1 or 0 (zero) if the modem is or is not fax capable.

faxpoll

A Dials a *Connection Directory* entry or group, or a specified number, to receive faxes from a host.

faxpoll {DIALDIR [GROUP]} | {index | string | CURRENT | FIRST} | string [timeval]

DIALDIR	An optional keyword, indicating that the string argument is the name of a *Connection Directory* entry or group, if GROUP is also specified.
GROUP	An optional keyword, indicating that the argument string contains a group name, rather than an entry name, to dial.
index	The option set index of a fax connection as listed in the ***Current Fax Connection*** list.
string	The name of the fax modem in the ***Current Fax Connection*** list.
CURRENT	The current modem/connection selected in *Setup*.
FIRST	The first available fax connection.
string	If DIALDIR is not specified, the string contains a fax number to be dialed. If DIALDIR is specified, the string contains the name of a *Connection Directory* entry, or a *Connection Directory* group if DIALDIR GROUP is specified.
timeval	A long value of type *timeval*, specifying the scheduled date and time for the polling to be performed. If the time specified by *timeval* has already past, the polling operation will be performed immediately, just as though no scheduled time had been specified.

Comments

The group or entry names must match exactly the information contained in the associated *Connection Directory* field, including upper- or lowercase. The **dialfind** command can be used to find either an entry or group name based on a partial name.

➥ *$FAXSTATUS behaves differently when a* **faxpoll** *command is executed. $FAXSTATUS will be set to 1 (Busy, sending), when a* **faxpoll** *command is being executed, but $FAXFILE will return null, since no file is being sent. $FAXSTATUS can then change from 1 to 2 (Busy, receiving), at which time $FAXFILE has the name of the file being received.*

See also

faxsend.

faxprint

Prints a specified fax file.

faxprint filespec

filespec	The name of the fax file to print.

Comments

faxprint will only accept **.fax** files, and the extension must be specified. If a fully-qualified path is not specified, the ***Received*** fax directory is assumed.

➥ *Any file type other than **.fax** will cause* **faxprint** *to fail. However, SUCCESS and FAILURE are not set by* **faxprint***.*

See also

faxlist, faxview.

faxremove

A Removes a fax from the received or scheduled fax list, returning FAILURE if the specified fax could not be removed.

faxremove RECEIVED | SCHEDULED index

RECEIVED	The target fax file is in the received fax list.
SCHEDULED	The target fax file is in the scheduled fax list.
index	A zero-based list index, corresponding to an entry in the ***Received*** or ***Scheduled Faxes*** list. If *index* is specified as -1, all entries will be removed from the targeted list.

See also

faxlist, faxsend and faxcancel.

faxsend

A Dials the specified number, *Connection Directory* entry or group, and transmits a specified fax file.

faxsend {index | string | CURRENT | FIRST string string} | {DIALDIR [GROUP] string} [COVERSHEET filespec | NONE] [NOTES string] [timeval] [SINGLE | MULTIPLE filespec [BINARY]] [MEMO filespec]

index	The option set index of a fax connection as listed in the ***Current Fax Connection*** list.
string	The name of the fax modem in the ***Current Fax Connection*** list.
CURRENT	The current modem/connection selected in *Setup*.
FIRST	The first available fax connection.
string	The information to insert in the "***To:***" field of the transmitted fax.
string	The fax number to dial, in order to send the fax.
DIALDIR string	The name of the *Connection Directory* entry to dial in order to send the fax. The group or entry name must match exactly the information contained in the associated *Connection Directory* field, including upper- or lowercase. The **dialfind** command can be used to find either an entry or group name based on a partial name.
GROUP	An optional keyword, indicating that the argument string contains a group name, rather than an entry name to dial.
COVERSHEET filespec	If specified, the name of a cover sheet file to be used in the fax transmittal. If a path is not included in the filespec, the cover sheet path in *Setup* is assumed. If a cover sheet is not specified, the default cover sheet will be used in the fax transmittal. To send a fax without using any cover sheet, use the NONE keyword.
NONE	If specified, the fax will be sent without a cover sheet.
NOTES string	If specified, *string* will be used in the "***Notes***" field of the cover sheet. If a cover sheet is not sent with the fax, or the specified cover sheet does not contain a "***Notes***" field, or the "***Notes***" field is too small to contain the complete *string*, *string* will be ignored.

timeval	A long value of type *timeval*, specifying the scheduled date and time for the fax to be sent. If the current time is already past the time specified by *timeval*, then the operation will be performed immediately, just as though no scheduled time had been provided.
SINGLE filespec	*Filespec* is a single file to transmit. If a path is not included in the *filespec*, the ***Scheduled Faxes*** path defined in ***Setup, Fax, Fax Paths*** is assumed.
MULTIPLE filespec	*Filespec* is a file containing one or more filenames to transmit by fax. Each filename should appear on a single line within the selection file. If a path is not included in *filespec*, the User Path is assumed. However, for filenames within the list of files, the ***Scheduled Faxes*** path in ***Setup, Fax, Fax Paths*** is assumed if no path is specified.
BINARY	If specified, the specified file(s) will be sent as a *Binary File Transfer*, and will not print as faxes on the receiving side.
MEMO filespec	The ASCII text file specified by *filespec* will be treated as a memo fax. If a path is not included in *filespec*, the User Path is assumed.

Comments

Under ASPECT, fax transmissions are asynchronous operations, meaning that they can run independently of script execution. A **faxsend** command only initiates a fax attempt. While the fax is processing, ASPECT executes the next commands in the script. If the script needs to wait for the fax attempt to complete, a **while** loop should be used to “stall” the script.

If the script executes a **while** loop to monitor the status of an on-going file transfer, you may find it useful to include the **yield** command within the loop. The **yield** command causes ASPECT to release its processing time to the system.

See also

faxpoll, faxlist and faxcancel.

faxstatus

Queries the status of a specific fax connection.

faxstatus index | string | CURRENT intvar [strvar [strvar [strvar]]]

index	The option set index of a fax connection as listed in the ***Current Fax Connection*** list.

string	The name of the fax modem in the ***Current Fax Connection*** list.
CURRENT	The current modem/connection selected in *Setup*.
intvar	An integer which stores the current value of $FAXSTATUS.
strvar	An optional string variable which receives the name of the file being sent or received. The string will be set to null if no fax transaction is in progress for the given connection index.
strvar	An optional string variable which receives the last message associated with the current fax transaction on the specified fax connection.
strvar	An optional string variable which receives the station ID of the location to which the fax is being sent. The string will be set to null if no fax transaction is in progress for the given connection index.

See also

faxmodem, faxsend, faxpoll, faxcancel; $FAXSTATUS in “*System Variables*.” .

faxview

Displays a fax file.

faxview filespec

filespec	The name of the fax file to view.

Comments

faxview will only accept **.fax** files, and the extension must be specified. If a fully-qualified path is not specified, the ***Received fax*** path in ***Setup, Fax, Fax Paths*** is assumed.

➥ *Any file type other than **.fax** will cause* **faxview** *to fail, but* **faxview** *does not set SUCCESS and FAILURE.*

See also

faxlist and faxprint.

fclear

Clears all end-of-file and error flags associated with the indicated file id.

fclear id

See also

fopen, feof and ferror.

fclose

A Closes the file corresponding to the indicated file id.

fclose id

Comments

The I/O buffer contents for a file will automatically be written before it is closed if it was opened in WRITE, READWRITE, or CREATE mode. **fclose** can be tested with the **if** SUCCESS statement, returning false if an error occurred while closing the file.

See also

fopen and fflush.

fcombobox

Adds a combination edit/selection or "drop-down" selection control to an ASPECT dialog box.

fcombobox id left top width height SIMPLE | DROPDOWN | DROPDOWNLIST filespec [OFFSET fileoffset [LENGTH filelength]] strvar [strlength] [SORT]

id	A unique integer constant value assigned to the **fcombobox.** When a selection is made from the list, this value will be reported by **dlgevent**.
left top width height	These integer constants determine the position and size of the **fcombobox** control in Dialog Box Units or DBUs, relative to the dialog box's dimensions. For more information on DBUs, see "*Dialog Box Units*" on page 48.
SIMPLE	The list box is displayed at all times. The current selection in the list box is displayed in the edit control.

DROPDOWN	The current selection is displayed in the edit control, but the list box is not displayed unless the user selects the down arrow button next to the edit control.
DROPDOWNLIST	Similar to the DROPDOWN type, but the edit control is replaced by a static text item that displays the current selection in the list box.
filespec	The file containing the items to be displayed. Each item must occur as a separate line in the file. *Filespec* may include a drive and path. If no path is specified, the Aspect Path is assumed.
OFFSET fileoffset	An optional offset within the file where the list to be displayed begins. A long variable, or a constant.
LENGTH filelength	If *filelength* is specified, only the allowed number of bytes will be displayed. A long variable, or a constant.
strvar	Contains a single list selection. *strvar* can be pre-initialized with an item, allowing it to already be selected upon entry.
strlength	If specified, defines the maximum number of characters which can be entered in the edit control. This parameter is not valid with the DROPDOWNLIST style. An integer variable or constant.
SORT	If specified, the contents of the **fcombobox** will be sorted lexicographically in ascending order.

Comments

Like **combobox**, this command allows a user to make a single selection from a list. With **fcombobox**, however, the list of items is taken from a disk file. **fcombobox** can only occur within a **dialogbox** group.

If the **fcombobox** contains an edit field, and the user types information into the field, a dialog event will be generated when the fcombobox loses the input focus.

See also

combobox, dialogbox, dlgsave, dlgupdate, dlgevent, flistbox and listbox.

fdelblock

A Deletes a block of data from the specified file.

fdelblock id integer

id	The target file index.
integer	The size in bytes to be deleted from the target file.

Comments

fdelblock begins deleting data at the current position of the file pointer. The file must be opened with either the READWRITE or CREATE parameters or **fdelblock** will return FAILURE. If the amount specified is greater than the data remaining in the file, **fdelblock** will return FAILURE. Use **ftell** and **flength** to determine pointer position and filesize.

See also

flength, finsblock, fseek, fstrfmt, ftell, fwrite, ftruncate, fflush and rewind.

feditbox

Adds a text editing box displaying the contents of a disk file in an ASPECT dialog box.

feditbox id left top width height filespec [HSCROLL]

id	A unique integer constant value assigned to the **feditbox** by the script. When the contents of the **feditbox** are changed and the control loses input focus, this value will be reported by **dlgevent**.
left top width height	These integer constants determine the position and size of the **feditbox** control in Dialog Box Units or DBUs, relative to the dialog box. For more information on DBUs, see "*Dialog Box Units*" on page 48.
filespec	A string variable or constant containing the name of the file to edit. If a path is not specified, the Aspect Path is assumed.
HSCROLL	An optional parameter allowing horizontal scrolling instead of line "wrapping" at the right margin of the text edit box.

Comments

If the specified text file doesn't exist when the **feditbox** command is executed, it's created automatically. The **feditbox** contents are not saved until the dialog box is exited with an OK status. The contents will not be saved if the dialog is terminated with a **Cancel**.

feditbox supports a maximum file size of 64K for editing, but the actual allowable file size may be less depending on available resources. An error beep will sound whenever a file cannot be loaded or saved. Standard Windows cursor and editing controls are available within an **feditbox** control.

feditbox can only occur within a **dialogbox** group.

See also

dialogbox, disable, dlgevent, dlgsave, dlgupdate, editbox, enable, ftext and text.

feof

A Tests for end-of-file condition on the specified file index id.

feof id [intvar]

id	The file id to test.
intvar	If specified, will be set to the result of the test; 0 for false and 1 for true.

Comments

When reading from a file, **feof** will not return true until an attempt has been made to read past the end of file. Reading just the last character of a file does not set end of file. End of file is generally never set when writing to a file.

See also

fclear, fclose, ferror, fopen, fseek and rewind.

ferror

A Tests for any error condition on the specified file index.

ferror id [intvar]

id	The file index id to test.
intvar	If specified, will be set to 0 if no error is detected. If an error is detected, *intvar* will be set to 1.

See also

fclear, fclose, feof, fopen, fseek and rewind.

fetch

A Returns the current value(s) of any **set** command parameter.

fetch param datavar

param	The desired **set** command parameter.
datavar	The current value of the specified **set** command parameter(s). Note that this value must be returned to the proper data type. The variable may be in *intvar, strvar* or *longvar* format, depending on the desired **set** command. The definitions for the **set** commands in "*Set and Fetch Statements*" on page 363 include all possible values for each command parameter.

Comments

ASPECT provides a **set**/**fetch** command for every item that can be changed in Procomm Plus *Setup*, except for items affecting individual window colors and terminal attributes. Note that changes made to the current *Setup* values will not be saved to disk until the **setup save** command is issued, or until the user explicitly saves the current settings through the user interface.

Although most **fetch** commands will always succeed, all **fetch** commands update the SUCCESS and FAILURE system variables based upon the results of the command. Certain **fetch** commands may fail if the information given to the command is invalid.

If a **fetch** command returns an integer value, its corresponding **set** command will accept either an integer or a keyword. This integer value is checked at run-time, but users should employ the keyword form when **set**ting a value not previously **fetch**ed.

The **set**/**fetch** commands are organized into groups that correspond to the parameters they affect. In addition, each protocol and each terminal emulation has its own **set**/**fetch** group. However, some of these options represent advanced settings which are not available for all emulation or protocols. For more information, see "*Set and Fetch Statements*" on page 363.

Commands without one of these group identifiers affect the overall operation of Procomm Plus. In general, the **fetch**ed value for parameters with OFF | ON syntax is 0 for OFF, 1 for ON. For parameters with NO | YES syntax, **fetch**ed values are typically 0 for NO, 1 for YES.

If you've **set dialentry access** to a specific *Connection Directory* entry, **fetch** will retrieve values from that entry instead of the current Procomm Plus *Setup* file. Only certain **set**/**fetch** commands have this dual functionality.

See also

setup and set; "*Set and Fetch Statements*."

fflush

A Writes the current I/O buffer contents to the specified file id.

fflush id

id	The file id to be flushed.

Comments

If the file was opened in READ mode, the buffer contents are cleared. In CREATE, WRITE, or READWRITE mode, buffers are automatically flushed when they're full or when the file is closed.

This command can be tested with the **if** SUCCESS statement, returning false if an error occurred while writing the buffer contents to the file.

See also

fclose, fclear and fopen.

fgetc

A Reads a character from the specified file id into a variable.

fgetc id intvar

id	The file id to access.
intvar	Will be set to a character value if **fgetc** was successful; -1 otherwise.

Comments

To make sure that a character is available, first check the file status using the **feof** command. Since a character contains a single byte, the file pointer will be incremented by 1 after performing **fgetc**.

See also

fgets, fopen, fputc, feof and fread.

fgets

A Reads a string from the file corresponding to a given id into a variable.

fgets id strvar

id	The file id to access.
strvar	The string receiving the data.

Comments

Reading will terminate when **fgets** encounters a line feed character, the end-of-file, or when 256 characters have been read.

The string will include the ending line feed if the file wasn't opened with the optional TEXT parameter. The **feof** command can test the file before reading it, and the **if** SUCCESS statement will return false if an error occurred while reading the file.

See also

fgetc, fputs, feof, fread and fopen.

fileget

A Reports date, time, attributes or size for a given *filespec*.

fileget filespec {ATTRIBUTE | DATE | TIME strvar} | {LTIME | SIZE longvar}

filespec	The desired file.
ATTRIBUTE strvar	*strvar* will contain the attributes of the specified file.
DATE strvar	*strvar* will contain a date string reflecting the specified file’s last update.
TIME strvar	*strvar* will contain a time string reflecting the specified file’s last update.
LTIME longvar	*longvar* will contain a *timeval*, which can be converted to time/date string(s) using **ltimestrs** or **ltimestring**.
SIZE longvar	Will contain the size in bytes of the specified file.

Comment

If no path is specified, the current User Path will be used. Wildcards are not allowed. **fileget** reports FAILURE if the file could not be found.

When getting attributes with the **fileget** command, the attribute string may consist of the following characters representing attribute types: ’***R***’ for read-only, ’***H***’ for hidden, ’***S***’ for system, and ’***A***’ for archive. ‘***N***’ for normal.

➥ **fileget** *SIZE should not be used to get the size of a file that is currently opened with WRITE or CREATE access! The* **flength** *command should be used in this case, since it accounts for data which may not have been written to disk when the command is executed.*

See also

fileset, findfirst, findnext, ltimestring and ltimestrs.

fileset

A Sets date, time, attributes and size for a given filespec.

fileset filespec {ATTRIBUTE | DATE | TIME string} | {LTIME | SIZE long}

filespec	The target filename.
ATTRIBUTE string	Will set the target file's attributes to the given value(s).
DATE string	Will set the target file's date stamp to the given value.
TIME string	Will set the target file's time stamp to the given value.
LTIME long	Will set the target file's *TIMEVAL* to the given value.
SIZE long	Will set the target file's size to the given value, truncating the file if applicable.

Comment

If no path is specified, the current User Path will be used. Wildcards are not allowed. **fileset** returns FAILURE if the file could not be found, or if the requested action could not be performed.

When setting attributes with the **fileset** command, the attribute string may consist of the following characters representing attribute types: '***N***' for normal, '***R***' for read-only, '***H***' for hidden, '***S***' for system, and '***A***' for archive. A null string will unset all attributes for the target file. The normal attribute is valid only when no other attributes are specified.

This command will set FAILURE if an invalid date string and/or time string is used as an argument to the command.

➥ *The time and date strings must reflect valid DOS times and dates. File times work with a 2 second resolution. As a result, the file may actually show a time of one second less than what you set with* **fileset***.*

See also

fileget, findfirst, findnext and strsltime.

filetoclip

A Retrieves the contents of a file and places the data in the clipboard.

filetoclip BITMAP | METAFILE | TEXT filespec

BITMAP | METAFILE | TEXT	Specifies the filetype for the copy.
filespec	The file which will be copied to the clipboard. If no path is specified, the User Path is assumed.

Comments

Note that the input file must be of the proper type for the retrieval to be successful. Bitmaps usually carry the extension **.bmp**. Metafiles are usually identified with a **.wmf** extension, and these files must contain the placeable metafile format that is typical within Windows. Enhanced metafile format files, usually identified with a **.emf** extension, are also supported.

See also

strtoclip, cliptofile and pastetext.

fileview

Displays a text file in a modal dialog box

fileview filespec

filespec	The file to display. If no path is supplied, ASPECT will search for the file in the User Path.

Comments

fileview should only be used with ASCII text files.

findfirst

A Locates a disk file using a specification you provide.

findfirst filespec [string]

filespec	The argument for the file search. Wildcard characters “?” and “*” are allowed, and a path may be included. **findfirst** defaults to the current User Path if a path isn’t provided.
string	An optional string specifying the attributes to be used for the search. If ‘***N***’ is used, only “normal” files with no attributes, or just a read-only and/or archive attribute set will be included. Specifying ’***R***’ for read-only, ’***H***’ for hidden, ’***S***’ for system, ’***A***’ for archive, ’***D***’ for directory and ’***V***’ for volume will search those files as well as the normal files. If the volume is being sought, and a volume label exists, it will always be found regardless of the *filespec* that was specified. If a volume label is found, the system variables $FILESPEC, $FILENAME and $FNAME will contain up to 11 characters of the volume name, and $FEXT will be set to null. To search exclusively for files with specific attributes, the character ’***X***’ may be included. For example, the string “***XD***” would search only for directories.

Comments

The file’s fully-qualified filespec, name, extension, name and extension, size, date stamp, time stamp and attributes are stored in the system variables $FILESPEC, $FNAME, $FEXT, $FILENAME, $FSIZE, $FDATE, $FTIME, and $FATTR respectively. The long value representing the file’s date and time is stored in $FLTIME.

See also

fileget, fileset, findnext, shortpath, and strsltime; $FLTIME, $FSIZE, $FATTR, $FDATE, $FEXT, $FILENAME, $FNAME, $FTIME and $FILESPEC in “*System Variables*.”

findnext

A Locates additional disk files with the specification provided in a previously-executed **findfirst** command.

findnext

Comments

The file specification for this command is taken from the previously-used **findfirst** command. **if** SUCCESS can be used to test the result of the operation.

As with **findfirst**, the file's fully-qualified filespec, name, extension, name and extension, size, date stamp, time stamp and attributes are stored in the system variables $FILESPEC, $FNAME, $FEXT, $FILENAME, $FSIZE, $FDATE, $FTIME, and $FATTR respectively. The long value representing the file's date and time stamp of the file is stored in $FLTIME.

See also

fileget, fileset, findfirst and strsltime; $FLTIME, $FSIZE, $FATTR, $FDATE, $FEXT, $FILENAME, $FNAME, $FTIME and $FILESPEC in "*System Variables*."

finsblock

A Inserts a block of space into the specified file id at the current position in the file.

finsblock id integer

id	The id of the target file.
integer	The size of block to insert.

Comments

The file must be opened with either the READWRITE or CREATE parameters or **finsblock** will return FAILURE. The current file position remains unchanged after inserting the block. Thus, new data can be written directly following **finsblock**. Use **if** SUCCESS to confirm that the block was inserted before attempting to write to the file, however.

See also

fdelblock.

firsttask

A Returns the first task encountered in the Windows Task List.

firsttask intvar

intvar	The task id.

Comments

ASPECT returns id's only for those tasks that have a top-level window with a caption containing text. Certain special tasks running in Windows may not meet these requirements, and will not be returned. The ordering of tasks is not necessarily preserved between two **firsttask** commands.

See also

nexttask, taskactivate, taskexit, taskexists, taskname and taskwin; $TASK and $PWTASK in "*System Variables*."

flength

Returns the file size of an open file.

flength id longvar

id	The id of the target file.
longvar	Will contain the length of the target file in bytes.

Comments

flength should be used to determine the length of a file which is currently opened with WRITE access, since it will account for data which has not yet been written to disk.

flistbox

Adds a list box to an ASPECT dialog box. Like **listbox**, this command allows a user to select a single item or multiple items from a list. With **flistbox**, however, the list of items is taken from a disk file, allowing for a larger list than can be contained in a string.

flistbox id left top width height filespec [OFFSET fileoffset [LENGTH filelength]] [tabstring]
{SINGLE strvar} | {MULTIPLE filespec} [HSCROLL] [SORT]

id	A unique integer constant value assigned to the file list box**.** This value will be reported by **dlgevent** when the contents of the file list box are changed, or a selection is made.
left top width height	These integer constants determine the position and size of the **flistbox** control in Dialog Box Units or DBUs, relative to the dialog box. For more information on DBUs, see "*Dialog Box Units*" on page 48.

filespec	The file containing the items to be displayed. Each item must occur as a separate line in the file. The *filespec* may include a drive and path. If no path is specified, the Aspect Path is assumed.
OFFSET fileoffset	An optional offset within the file where the list to be displayed begins. A long variable, or a constant.
LENGTH filelength	If LENGTH is specified, only the number of bytes specified in *filelength* will be displayed. *Filelength* is a long variable, or a constant.
tabstring	An optional, comma-separated string variable containing numbers in dialog units indicating column locations for tabs contained within the file. If only one tab stop is specified, the file list box will be initialized with multiple stops separated by the single tab stop value. If more than one tab stop is given, then only those tab stops will be initialized.
SINGLE strvar	If specified, *strvar* contains a single list selection. *strvar* can be pre-initialized with an item, allowing that item to be pre-selected upon entry to the dialog.
MULTIPLE filespec	*Filespec* is a string containing the name of the file to be created with the multiple selections chosen from the displayed list. The file described by *filespec* can be pre-initialized with items, allowing them to be pre-selected upon entry to the dialog.
HSCROLL	If specified, the **flistbox** provides scroll bars to allow the user to scroll the list items horizontally. Horizontal scrolling is appropriate only for list entries which contain no tab characters.
SORT	If specified, the contents of the **flistbox** will be sorted alphabetically.

Comments

flistbox allows multiple selections. Each selection is stored in the file specified by MULTIPLE *filespec,* with each selected item appearing on a separate line. An **flistbox** command can only occur within a **dialogbox** group.

See also

dialogbox, combobox, fcombobox, dlgupdate, listbox and dlgevent.

float

Defines a global or local **float** variable, or an array of **float** variables.

float name[=expression][,name[=expression]]...

name	The name of the new variable. This name must follow the standard ASPECT naming conventions. When defining an array, the subscript expression must evaluate to a constant value.
= expression	An initializing expression. It can specify operators, constants or previously-declared numeric variables, or a function which returns a numeric result. The initializing expression cannot be used when defining an array.

Comments

Up to 256 global **float** variables can be defined, as well as 256 global **float** arrays. Local **float** definitions are limited by the available stack space at the time the procedure or function is called.

See "*User Defined Variables,*" "*Global and Local Variables*" and "*Arrays,*" for more information.

See also

integer, long,param and string.

floor

Computes the largest integral value, with no fractional amount less than or equal to a floating point number.

floor float numvar

float	A **float** variable or constant.
numvar	A numeric variable to receive the resulting value.

See also

ceil.

fopen

A Opens a file in the indicated mode and assigns it to a file id.

fopen id filespec READ | WRITE | READWRITE | CREATE | APPEND | READAPPEND [TEXT] [SHARED]

id	The file id to assign to the file. This value must be greater than or equal to zero.
filespec	A valid file specification. If no path is specified, the current User Path is assumed.
READ	Data can only be read from the file.
WRITE	Data can only be written to the file. If a file described by *filespec* does not exist, it is created.
READWRITE	Data can be read from, or written to the file. If a file described by *filespec* does not exist, it is created.
CREATE	Will create a new file. If a file described by *filespec* already exists, CREATE mode will erase its contents. After the file is CREATEd, the filemode assumes READWRITE mode.
APPEND	Data can only be written to the file. Additionally, writes can only be made at the end of the file. If the file specified in *filespec* does not already exists, APPEND mode will create the file.
READAPPEND	Will allow you to read anywhere within the file, but all writes will be made at the end of the file. If the file specified in *filespec* does not already exist, READAPPEND mode will create the file.
TEXT	Affects only the **fgets** and **fputs** commands. This optional parameter forces ASPECT to strip line feeds and carriage return/ line feed combinations from the end of a string during an **fgets** operation, and forces ASPECT to append a carriage return/line feed combination to a string written with the **fputs** command. Additionally, if the last string read with an **fgets** command consists only of an EOF character, a null string is returned.
SHARED	Allows the opened file to be shared with *child* scripts. Child scripts, as long as they are aware of the file ID, can perform any file I/O operations on SHARED files without having to open the file. Note, however, that any changes to the file or file position are in effect when the parent script resumes control.

Comments

The file pointer position is always initially set at the start of the file, no matter what mode is used to open it. An unlimited number of files may be opened at once.

See also

fclose, fdelblock, finsblock, fflush, fputc, fputs, fseek, ftell, fread, feof, ferror, flength, fgets, fgetc, fstrfmt, ftruncate, rewind and fwrite.

for

Repeats a command or series of commands a specific number of times.

for numvar[=expression] UPTO | DOWNTO number [BY number]

.

.

.

endfor

numvar	The loop variable used as a counter. If an array element is specified, and the subscript expression contains a variable, changes to that variable within the **for** command block do not affect which loop variable is examined; the tested variable is determined when initialization is performed.
=expression	Optional initialization expression for the loop variable. Typically this is an assignment expression to initialize the variable.
UPTO | DOWNTO	Indicates whether the variable is to be increased (UPTO) or decreased (DOWNTO) after each iteration.
number	The boundary value for the loop counter. It is inclusive. For example, if UPTO is set to 25 and the counter variable is initialized to 1, the loop will be performed 25 times.

BY number	Determines the amount to be added to (UPTO) or subtracted from (DOWNTO) the target amount after each iteration. This value is typically a positive number, since a negative number in conjunction with the UPTO keyword will result in a step down from the initial value. A negative number in conjunction with the DOWNTO keyword will result in a step up from the initial value. If BY *number* is not specified, the default step value is 1. *Number* must be a constant or simple variable, or an array variable with a constant sub-expression. It cannot be the result of a general expression involving more than 1 variable.
endfor	Required terminator for the **for** statement group.

Comments

The **loopfor** and **exitfor** commands allow branching to the next iteration test or command following the **endfor**.

See also

while, loopfor and exitfor.

fputc

A Writes a character value to a file.

fputc id character

id	The target file id.
character	The character can be any ASCII value. **fputc** writes a single byte to the file.

See also

fgetc, fopen, fputs and fwrite.

fputs

A Writes a string to the file corresponding to the specified index.

fputs id string

id	The target file id.

string	Any valid ASPECT constant or variable string.

Comments

fputs should only be used on null-terminated strings. The **fwrite** command can be used to write "raw" data.

If the **fopen** command used to open the file included the TEXT parameter, **fputs** will append a CR/LF after writing the *string* to the file. **fputs** can be tested with the **if** SUCCESS statement, returning false if the data cannot be written to the file.

See also

fopen, fputc, fwrite, fstrfmt and fgets.

fread

A Reads a block of data from a file into a string variable and returns the actual number of bytes read in an integer variable.

fread id numvar | {strvar strlength} [intvar]

id	The source file index.
numvar	An integer, float or long variable. The number of bytes read is determined by the number of bytes of storage contained by *numvar*'s data type.
strvar	String variable that will hold the results of the read.
strlength	The size in bytes of the block to be read. An integer value between 0 and 256.
intvar	Return value is the actual number of bytes read, which may be less than requested if an error occurs or end-of-file is reached.

Comments

ASPECT automatically determines the number of bytes to be read if a numeric variable is specified. An integer argument will cause 4 bytes to be read from the file. A long will read 4 bytes, and a float 8 bytes. The file pointer will be updated accordingly.

The status of this command can be checked by comparing the number of bytes read with the number requested. You can also test for the end-of-file condition. To test for errors while reading a file, use the **if** SUCCESS or **if** FAILURE statements.

See also

fgetc, fwrite, fgets and fopen.

fseek

A Repositions the file pointer for the file corresponding to the specified id.

fseek id fileoffset integer

id	The id of the target file.
fileoffset	Offset of the new position relative to the origin of the **fseek.** *fileoffset*, a **long** variable or constant, can be positive or negative. If *fileoffset* is a negative constant, enclose the parameter within parenthesis to prevent a “Missing token” error from the ASPECT compiler.
integer	The origin of the **fseek** operation. Possible values are 0 for beginning of the file, 1 for current pointer position or 2 for end-of-file.

Comments

Attempting to reposition the file pointer before the beginning of the file will result in an error. Use **ftell** to determine the current position of the file pointer.

See also

ftell, fopen, ferror and rewind.

fstrfmt

A Writes a formatted string to the file referenced by the specified id.

fstrfmt id formatstr [arglist]

id	Target file id.
formatstr	A string containing text and/or data format specifiers.
arglist	Up to 12 arguments to be loaded into the *formatstr.*

Comments

The processed *formatstr* cannot exceed 256 characters. For more information on the *formatstr* structure, see "*Formatting Text and Data*" on page 49.

See also

fopen, fputs, fwrite and strfmt.

ftell

Returns the current file pointer position of the file corresponding to the specified file id.

ftell id longvar

id	The id of the target file.
longvar	Will contain the number of bytes past the beginning of the file.

Comments

ftell can be used together with **fseek** to store and return to a particular file location.

See also

fopen, fseek and rewind.

ftext

Displays a text file in an ASPECT dialog box.

ftext id left top width height filespec [{OFFSET fileoffset [LENGTH filelength]} |DYNAMIC] [HSCROLL]

id	A unique integer constant value assigned to the **ftext** control by the script.
left top width height	Integer constants which determine the position and size of the **ftext** control in Dialog Box Units or DBUs, relative to the dialog box's dimensions. For more information on DBUs, see "*Dialog Box Units*" on page 48.
filespec	A constant or string variable containing the name of the file to be displayed. A drive and path may be included. If no path is specified, the Aspect Path is assumed.

fileoffset	An optional offset within the file where the text to be displayed begins. A long variable or constant.
LENGTH filelength	If LENGTH *filelength* is used, only the number of bytes specified in *filelength* will be displayed. *Filelength* is a long variable, or a constant.
DYNAMIC	Allows the file to be updated using **dlgupdate** in the edit control as it increases in size.
HSCROLL	An optional parameter allowing horizontal scrolling instead of lines “wrapping” at the right margin of the text edit box.

Comments

The **ftext** command supports a maximum file size of 64K. The actual maximum file size may be less depending on available resources, however. An error beep will sound whenever a file cannot be loaded.

An **ftext** control with dynamic file updating can be reset by changing the filespec to another file or to null and issuing the **dlgupdate** command.

Standard Windows cursor controls are available within file based text controls. **ftext** can only occur within a **dialogbox** group.

See also

dialogbox, feditbox, dlgupdate and text.

ftoa

Converts a **float** to an ASCII string and stores it in a **string** variable.

ftoa float strvar

float	The **float** value to convert.
strvar	Will contain the converted value.

See also

atof, itoa, ltoa, numtostr and strfmt.

ftp

A Allows file manipulation on either host or local FTP machines.

ftp LOCAL | REMOTE {CHDIR | MKDIR | RMDIR pathname} | {COPYFILE | DELFILE | FILELIST filespec} | {RENAME filespec filespec} | {GETDIR strvar}

LOCAL	Tells the script that the following parameters affect the LOCAL machine.
REMOTE	Tells the script that the following parameters affect the host FTP site.
CHDIR	Changes the User Path to the drive and/or directory specified in the *pathname* parameter.
MKDIR	Creates a new directory using the path and/or directory specified by the *pathname* parameter.
RMDIR	Removes an empty directory using the path specified by the *pathname* parameter.
pathname	Identifies the DOS drive (for example, C:), pathname and directory. The drive identifier is only required if you change the current drive.
COPYFILE	Copies the file specified by the *filespec* parameter.
DELFILE	Deletes the file specified by the *filespec* parameter.
FILELIST	Enumerates a list of files from the current directory of the specified machine to the file specified by the *filespec* parameter.
filespec	The name of the file to be affected.
RENAME	Renames an existing file, reporting its results in SUCCESS and FAILURE.
filespec	The source filename.
filespec	The destination filename.
GETDIR	Returns the current working directory path of the current directory into a string variable.
strvar	Contains the current path as returned by **getdir.**

See also

www and pwmode; $FTPCONNECT, $FTOPTIONS, and $FTPSTATUS in "*System Variables*."

ftruncate

A Truncates or clips the specified file at the current file position.

ftruncate id

id	The target file id.

Comments

The targeted file is truncated at the current file pointer location. **ftruncate** with the file pointer at zero essentially overwrites an existing file. **fseek** can be used to position the file pointer to the desired location.

The target file must have been opened in CREATE, READWRITE, or WRITE mode.

See also

fdelblock, fseek, ftell and rewind.

fullpath

A Returns the fully-qualified drive and path for a specified *filespec*.

fullpath strvar filespec [pathname]

strvar	Will contain the resultant drive and path.
filespec	The filename to be tested.
pathname	An optional default pathname used to qualify the *filespec*. If omitted, the User Path is assumed.

Comments

fullpath does not verify that the file exists. Use **isfile** to test for the existence of a file.

See also

addfilename, getdir, getfilename, getpathname, isfile and splitpath; $USERPATH in “*System Variables*”.

func

Marks the beginning of a function block.

func name : datatype

datatype	Specifies the type of data returned by the function. Valid types are **float**, **integer**, **long** and **string**.

Comments

A function may define a parameter list. The parameters are variables of any type, and are local to the function. The parameters are initialized with arguments passed via the **call** which invoked the function. Functions return constant values which cannot be modified directly.

Functions can also be called in “expression” format, which allows functions to be used in general expressions. For example, a function could be used as part of an assignment. The syntax for the expression format is:

var = fname([arglist])

If a function returns a numeric value, it can be used as an expression in **if** or **while** statements. For example,

if ! fname([arglist])

would test for a return value of zero from function *fname*.

➥ *The expression form of function calls is highly recommended. It is shorter, just as readable, and less verbose. Also, future enhancements to ASPECT may require it.*

Each form constitutes a function call which causes execution to transfer to the start of the function block. When the function is exited, execution returns to the point where the **call** was made.

See also

call, proc, endfunc, return, setjmp, longjmp and param.

fwrite

A Writes a block of data to a file.

fwrite id number | {string strlength} [intvar]

id	The target file id.
number	An integer, float, or long value. The number of bytes to write is determined by the storage requirements of number’s data type.
string	A string or string variable that contains the block to be written.
strlength	The size in bytes, in an integer value between 0 and 256, of the block to be written.
intvar	If specified, will return the number of bytes actually written to the file.

Comments

fwrite can be tested with the **if** SUCCESS statement, which returns false if the data cannot be written.

See also

fopen, fputc, fputs, fread and fstrfmt.

getcur

Returns the current cursor location on the *Terminal display.*

getcur intvar intvar

intvar	Will contain the current row.
intvar	Will contain the current column.

getdir

A Returns the current working directory path of the specified drive into a string variable.

getdir disk strvar

disk	An integer number designating the drive. 0 indicates the current drive, 1 specifies drive A, 2 for drive B and so on, up to 26 for Z.
strvar	Will contain the requested path.

Comments

When retrieving the current working directory path using a disk value of 0, the current User Path will be returned. If a disk value other than 0 is specified, the current working directory for that drive will be returned.

➥ *The User Path and current working directory for the drive contained within the User Path may be different!*

You can test for possible drive access errors with **if** SUCCESS or **if** FAILURE statements.

See also

chdir and taskpath; $USERPATH, $PWTASKPATH and $WINPATH.

getenv

A Returns the contents of an environment variable definition.

getenv string strvar

string	An environment variable currently defined in DOS.
strvar	Will contain the requested environment definition, if it exists. Otherwise, *strvar* will be a null string. **if** SUCCESS can also be used to test the command results.

See also

putenv.

getfile

A Receives or downloads a file from a remote computer using the specified transfer protocol, returning SUCCESS if the transfer initiated successfully.

getfile protocol | index | string | DEFAULT [filespec[filespec]]

protocol	A keyword that represents a file transfer protocol. For a complete list of protocol keywords and indices, see “*Protocol Names and Indices*” on page 59.
index	An integer value corresponding to the desired protocol.
string	A string constant or variable which represents a file transfer protocol, or a named-item version of a protocol.
DEFAULT	If DEFAULT is specified, the current default protocol will be used to download the file.
filespec	The name of the file used to store the downloaded file. If the *filespec* contains a path, the file will be created and saved in the specified path. If no path is included, the file will be saved to the directory specified in the ***Download path*** field in ***Setup, Options, Paths***.
filespec	Valid only with the Ind$file protocol, the second *filespec* must contain the name of the file as it exists on the mainframe.

Comments

Under ASPECT, file transfers are asynchronous operations, meaning that they can run independently of script execution. A **getfile** or **sendfile** command only initiates a file transfer attempt. While the transfer is processing, ASPECT executes the next commands in the script. If the script needs to wait for the transfer to complete, a **while** loop should be used to “stall” the script.

If the script executes a **while** loop to monitor the status of an on-going file transfer, you may find it useful to include the **yield** command within the loop. The **yield** command causes ASPECT to release its processing time to the system, which can improve file transfer performance.

The **when** $XFERFILE command can be used to call a separate procedure if the value of $XFERFILE changes.

If a protocol requires a *filespec*, a path may be included. Otherwise, Procomm Plus uses the directory specified in the ***Download path*** field in the ***Data, Data Options, Paths*** dialog in *Setup*. Protocols that do not require a *filespec* receive the filename from the remote system.

The Zmodem, Kermit, Ymodem, Ymodem-G, and CIS-B+ protocols do not require a *filespec* for downloads. The Xmodem, 1K-Xmodem, 1K-Xmodem-G, ASCII, and Raw ASCII protocols do. The Ind$file protocol is unique in that it requires two: one *filespec* for local storage of the received file, and one *filespec* specifying the name of the file as it exists on the host.

➥ *If index, string, or DEFAULT is specified, the compiler will treat the filespecs as optional. It is up to the user to provide the proper filespec information based on the protocol.*

See also

sendfile, yield and kermserve; $XFERSTATUS and $XFERFILE in "*System Variables*" and the set protocol, set aspect filexferbox and protocol-specific **set** commands in "*Set and Fetch Statements*."

getfilename

Extracts a filename from a file specification.

getfilename strvar filespec

strvar	Will contain the extracted filename. If a filename could not be extracted, *strvar* will be null.
filespec	The file and pathname to search.

See also

addfilename, getpathname and splitpath.

getpathname

Extracts a pathname from a file specification.

getpathname strvar filespec

strvar	Will contain the extracted pathname. If a path could not be extracted, *strvar* will be null.
filespec	The file and pathname to search.

See also

addfilename, getfilename and splitpath.

getvolume

Returns the volume label for the specified drive.

getvolume disk strvar

disk	An integer number designating the drive. 0 indicates the current drive, 1 specifies drive A, 2 for drive B and so on up to 26 for drive Z.
strvar	Will contain the volume label. If the drive has no label, *strvar* will be null.

See also

chdir and getdir; $VOLUME in “*System Variables*.”

goto

Performs an unconditional branch to the specified label within the current procedure or function.

goto name

name	A label within the current procedure. Labels provide a point to which the **goto** command directs program control.

Comments

When ASPECT encounters a **goto** command, it jumps to the location where the label is defined, resuming execution with the command immediately following the label. A *name* used for a label in one procedure can be reused in another procedure, but the label can only be referenced within the procedure where it’s defined. For non-local branching, use the **setjmp** and **longjmp** commands. For more information on using labels, see “*Labels*” on page 35.

See also

setjmp, longjmp and call.

groupbox

Displays a rectangle with optional text label in an ASPECT dialog box.

groupbox id left top width height [label]

id	The control id for the **groupbox**. An integer constant.
left top width height	Integer constants specifying the position and size of the **groupbox** in Dialog Box Units or DBUs, relative to the dialog box. For more information on DBUs, see "*Dialog Box Units*" on page 48.
label	An optional string constant or string variable that appears in the upper left corner of the rectangle. To include a keyboard accelerator for the groupbox, simply include an ampersand in front of the desired character. To display an ampersand in the label, use two ampersands. For example, the text "***&R&&D***" used as a label would display "***R&D***," with the ***R*** displayed as an underscored accelerator. The accelerator will direct the input focus to the control whose statement follows the **groupbox** statement in the **dialogbox** command group.

Comments

groupbox is used only within a **dialogbox** statement group. It is used to group dialog box controls graphically, but has no effect on the controls otherwise.

See also

dialogbox.

halt

Terminates the executing script file and any parent scripts unconditionally.

halt

See also

pwexit and exit; $PARENTFILE in "*System Variables*."

hangup

The operation of **hangup** is dependent on the current mode. In *Terminal* mode, **hangup** hangs up the phone. In *Web* mode, it discontinues loading of the current page, just like the default *ActionBar*'s ***Stop*** button. In *FTP*, *Telnet*, *Mail*, and *News* modes, **hangup** will terminate the connection with the server.

Additionally, if Procomm Plus is not running in *Terminal mode*, and the ***Internet Connection*** in the ***Setup, Internet, Internet Connection*** panel is not set to ***External Network***, this command attempts to hang up the specified ***Internet Connection***.

hangup

See also

disconnect.

help

Opens on-line *Help*.

help CONTENTS | HELP | {SEARCH[string]} | EXIT | {TOPIC index [POPUP]}

CONTENTS	Displays the help file contents topic.
HELP	Displays the topic that explains how to use help.
SEARCH[string]	Displays the topic containing the optional search key *string*, if an exact match was found. Otherwise, the **Search** dialog is displayed.
EXIT	Causes the help window to close if no other applications are currently using help.

TOPIC index	Displays help for a particular topic, specified by *index*. TOPIC is only useful for custom help files. Special software tools are required to build custom help files.
POPUP	If specified, the help topic will be displayed in a pop-up window, rather than a full window.

Comments

To load an alternate on-line Help file, use the **set aspect helpfile** command.

See also

dialogbox; set aspect helpfile in “*Set and Fetch Statements*.”

hotspot

A Places a mouse-selectable rectangle in the User window.

hotspot id left top width height BACKGROUND | USERWIN

id	A unique integer value assigned to each **hotspot**. This id value is used with the **objremove** command to remove the **hotspot** from the User window. It is also reported to the $OBJECT system variable when the **hotspot** is selected by the left mouse button.
left top	Integer values, expressed in User Window Units or UWUs, which determine the position of the top left corner of the **hotspot** with respect to the User window or the background graphic. For more information on UWUs, refer to “*User Window Units*” on page 48.
width height	The size of the **hotspot** in UWUs.
USERWIN | BACKGROUND	Specify USERWIN to “fix” the top left corner of the **hotspot** in position relative to the upper left corner of the User window. If BACKGROUND is specified, the **hotspot** stays over the same spot on a background graphic.

Comments

When the left mouse button is clicked, the **hotspot** inverts. When the right mouse button is clicked, and not over a non-**hotspot** object, all **hotspots** are revealed. If a **hotspot** has been hidden with **objhide**, however, it will not be revealed when the right mouse button is clicked.

A User window must exist before **hotspot**s can be placed. A **uwinpaint** command must be issued to display user window changes.

See also

uwincreate, icon, iconbutton, bitmap, dllobject, pushbutton, objmove, uwinpaint, objcoord, objpointid, objremove and metafile; $OBJECT in *"System Variables*."

icon

A Loads an icon from a disk file and displays it within an ASPECT dialog box or the User window. **icon** has two forms depending on where the icon is displayed. **icon** can only be tested for SUCCESS/FAILURE when it appears in a User window.

Dialog box format

icon id left top filespec index

id	The control id of the **icon.** This value can be used by **dlgupdate** to refresh the **icon** display. An integer constant.
left top	Integer constants, specifying the location of the icon in Dialog Box Units or DBUs, from the top left corner of the dialog box. For more information on DBUs, see "*Dialog Box Units*" on page 48.
filespec	A constant or a string variable, containing the name of the icon file. Icons are generally found in files with extensions of **.exe, .dll, .nil, .icl, .icn,** or **.ico**.
index	Some types of files can contain multiple icons. This integer constant or variable identifies the icon to be loaded. If the file only contains one icon, *index* should be 0.

See also

dialogbox, iconbutton, dlgupdate, bitmap and metafile.

User window format

icon id left top filespec index BACKGROUND | USERWIN

id	A unique integer constant or global variable identifying this **icon**. It can be used by the **objremove** command to remove the icon.

left top	The location of the **icon** in User Window Units (UWUs) from the top left corner of the User window. For more information about UWUs, see “*User Window Units*” on page 48.
filespec	A constant or global string variable which specifies the file containing the **icon**. If a path is not included in *filespec*, the current Aspect Path is assumed. Icons are generally found in files with extensions of **.exe, .dll, .nil, .icl, .icn,** or **.ico**.
index	Some types of files can contain multiple icons. This integer constant identifies the icon to be loaded. If the file only contains one icon, *index* should be 0.
BACKGROUND | USERWIN	Specify USERWIN to fix the top left corner of the icon in position relative to the upper left corner of the User window. Specify BACKGROUND to fix the icon over the same spot on a background graphic.

Comments

icons are for graphical display only. They do not generate $OBJECT events when clicked upon. A User window must exist before **icon**s can be placed. A **uwinpaint** command must be issued to display user window changes.

See also

iconbutton, uwincreate, uwinpaint, bitmap, metafile, dllobject, pushbutton, objcoord, objpointid, objpaint, objmove, objremove and hotspot.

iconbutton

A Loads an icon from a disk file and displays it within an ASPECT dialog box or a User window. The user may click on this icon to indicate a choice. The **iconbutton** command has two forms depending on where the icon is displayed. The **iconbutton** command can only be tested with **if** SUCCESS when it appears in a User window.

Dialog box format

iconbutton id left top label filespec index [OK | CANCEL] [DEFAULT]

id	A unique integer constant value given to the **iconbutton**. **dlgevent** will return this value when the **iconbutton** is selected.
left top	Integer constants, specifying the location of the **iconbutton** in Dialog Box Units (DBUs) from the top left corner of the dialog box. For more information about DBUs, see "*Dialog Box Units*" on page 48.
label	A constant or string variable displayed as a label, centered just below the **iconbutton**. To specify a keyboard accelerator for the **iconbutton**, simply include an ampersand (&) before the desired character. To display an ampersand in the label, use two ampersands. For example, the text "***&R&&D***" used as a label would display "***R&D***," with the '***R'*** displayed as an underscored accelerator.
filespec	A constant or string variable which specifies the file containing the icon to be displayed on the button. Icons are generally found in files with extensions of **.exe, .dll, .nil, .icl, .icn,** or **.ico**.
index	Some types of files can contain multiple icons. This integer constant or variable identifies the icon to be loaded. If the file only contains one icon, *index* should be 0.
OK	The dialog box is removed when an OK **iconbutton** is pressed. Changes made to file-related controls will be saved to disk.
CANCEL	The dialog box is removed when a CANCEL **iconbutton** is pressed. Changes made to file-related controls will not be saved.

DEFAULT	Specifies that the **iconbutton** is to be selected when the <Enter> key is pressed and no other button or **iconbutton** has the input focus, or when a **listbox**, **flistbox**, **combobox**, **fcombobox** or **dirlistbox** item is double-clicked. Note that only a single button or **iconbutton** can be named as a DEFAULT in a dialog box.

See also

dialogbox, icon, dlgupdate and dlgevent.

User window format

iconbutton id left top label filespec index BACKGROUND | USERWIN

id	A unique integer value given to this **iconbutton**. The value is assigned to $OBJECT when the **iconbutton** has been selected by the user with the left mouse button. It is also used by the **objremove** command to remove the icon button.
left top	If the USERWIN option is specified, these values are the location of the **iconbutton** in User Window Units, or UWUs, from the top left corner of the User window. For more information about UWUs, see "*User Window Units*" on page 48. If the BACKGROUND option is specified, the **iconbutton** is fixed over a certain area of the background graphic.
label	A constant or global string variable displayed as a label, centered just below the **iconbutton**. To specify a keyboard accelerator for the **iconbutton**, simply include an ampersand (&) before the desired character. To display an ampersand in the label, use two ampersands. For example, the text "***&R&&D***" used as a label would display "***R&D***," with the '***R***' displayed as an underscored accelerator.
filespec	A constant or global string variable which specifies the file containing the **icon**. Icons are generally found in files with extensions of **.exe, .dll, .nil, .icl, .icn,** or **.ico**. If a path is not specified in the *filespec*, the Aspect Path is assumed.
index	Some types of files can contain multiple icons. This integer constant identifies the icon to be loaded. If the file only contains one icon, *index* should be 0.

BACKGROUND | USERWIN	Specify USERWIN to fix the top left corner of the icon button in position relative to the upper left corner of the User window. Specify BACKGROUND to fix the button over the same spot on a background graphic.

Comments

A User window must exist before **iconbutton**s can be placed. An **uwinpaint** command must be issued to display User window changes.

See also

icon, uwincreate, uwinpaint, pushbutton, hotspot, metafile, bitmap, dllobject, objcoord, objpointid, objpaint and objremove; $OBJECT in "*System Variables*."

if

Executes the commands on the following line if the *condition* is true.

if condition

..

[elseif condition]

..

[else]

..

..

endif

Commands on the lines between the **if** statement and the next corresponding **elseif**, **else**, or **endif** command are executed when the condition tested by the **if** command is true.

[**elseif**]	A branch point evaluated when the previous **if** or **elseif** has evaluated as false.
[**else**]	A branch point that is executed when the previous **if** or **elseif** commands have evaluated as false.
endif	Concludes the **if** command statement group.

condition	A *condition* is true if it evaluates as a non-zero value. It is false if it evaluates as zero. A *condition* can consist of certain ASPECT commands and their parameters, or a numeric expression: **if** { [**not**] command } | { [**!**] expression }
command	Any ASPECT command which sets the SUCCESS/FAILURE flags. SUCCESS evaluates as true; FAILURE false. The **not** keyword can be used to reverse or negate the result of the ASPECT command. For example, when **isfile** is true, **not isfile** is false.
expression	A numeric expression, which may include user-defined functions returning numeric results. Numeric system variables such as $CARRIER may also be used, as well as the SUCCESS/FAILURE settings. The "!" ASPECT operator can be used to negate the results of an user-defined function, or a numeric expression.

Comments

Each **if** command must end with an **endif** command. An **else** or **elseif** command can be used to provide alternative branches based on the results of the **if** *condition*. There can be any number of **elseif** commands but all must precede the **else** command if it is used. The command following the **else** is executed if the condition(s) specified for the preceding **if** and **elseif** commands evaluate as false. Some common uses of the **if** command are end-of-file testing with **feof**, file error-checking with **ferror** and empty string checking with **nullstr**. See the definitions of these commands for more information.

See also

else, elseif, endif and switch; SUCCESS and FAILURE in "*System Variables*."

#if

Allows the *ASPECT Compiler* to determine whether the block of code which follows should be evaluated.

#if expression

expression	A constant value expression, which is an expression that results in a constant value throughout the script.

Comments

For more information, see the **#define** command.

See also

#define, #elif, and #endif

#ifdef

Allows conditional compilation by testing for the existence of a defined macro name.

#ifdef name

Comments

For more information, see the **#define** command.

See also

#endif, #ifndef, #elifdef, #elifndef, #else and #define.

#ifndef

Provides for conditional compilation by testing for the non-existence of a defined macro name.

#ifndef name

Comments

For more information, see the **#define** command.

See also

#ifdef, #elifdef, #elifndef, #else, #endif and #define.

#include

Merges commands from ASPECT source files during compilation.

#include “filespec”

“filespec”	The *filespec* must be a quoted string, but the described filename may have any valid extension. Typically, source files written for inclusion in other scripts are given extensions such as **.inc** or **.h**, however.

Comments

A **#include**d file may contain other **#include** commands. There is no limit to the number of nested levels. **#include** commands can be placed within or outside of a procedure block. However, if it is to be used within a procedure or function block, the **#include**d file cannot contain other procedures or functions. **#include** can be used to set up source "header" files containing a set of **#define** commands, much like those used in the "C" programming language. Such files allow you to share macros and definitions between several scripts simply by **#include**-ing the appropriate file.

You can also use **#include**d files to create a library of procedures and functions. For example, if your scripts often require a set of particular functions or procedures, you can place them in an **#include** file, then reference that file in any script that needs them. Library files also offer an advantage if the procedures or functions need to be updated. If the source files reference the **#include** file for a particular function or procedure, you can update all references to the function or procedure simply by re-compiling them. The *ASPECT Compiler* will not generate output for procedures and functions that aren't called. Your libraries can be as large as you like without adding unnecessary code to the compiled **.wax** script!

➥ *Two useful files are provided with Procomm Plus that can be* ***#include****d into your scripts. These files are* ***pw4menu.inc****, which defines all of Procomm Plus's menu values, and* ***vkeys.inc****, which defines the virtual key codes as shown in "Virtual Key Codes" on page 603. Both of these files are installed to the default ASPECT directory.*

integer

Defines a global or local integer variable or an array of integer variables.

integer name[=expression][,name[=expression]]...

name	The name of the new variable. This name must follow the standard ASPECT naming conventions. When defining an array, the subscript expression must result in a constant value.
= expression	An initializing expression. It can specify operators, constants or previously-declared numeric variables, or a function which returns a numeric result. The initializing expression cannot be used when defining an array.

Comments

Up to 256 global **integer** variables can be defined, as well as 256 global **integer** arrays. Local **integer** definitions are limited by the available stack space at the time the procedure or function is called. See “*User-defined Variables*” on page 28, “*Global and Local Variables*” on page 28, and “*Arrays*” on page 30.

See also

float, long, string and param.

intsltime

Converts integer date and time values into a long system time format.

intsltime integer integer integer integer integer integer longvar

integer	An integer between 1970 and 2106 representing the year.
integer	An integer between 1 and 12 representing the month.
integer	An integer between 1 and 31 representing the day.
integer	An integer between 0 and 23 representing the hour.
integer	An integer between 0 and 59 representing the minute.
integer	An integer between 0 and 59 representing the second.
longvar	Assigned a value in *timeval* format.

See also

strsltime, ltimeints and monthstr; $LTIME in “*System Variables*.”

isfile

A Determines if a specified file exists.

isfile filespec [intvar]

filespec	Any valid DOS path and/or filename, including extension. No wildcards are allowed. If no directory is specified, the User Path is assumed.
intvar	If specified, will contain the result of the test. If the file exists, *intvar* will be 1; otherwise 0.

Comments

The **isfile** command functions by attempting to open a file in read-only mode. If the file is currently in use by another process, the command may fail even though the file exists. The **findfirst** command is more reliable.

See also

copyfile, delfile, findfirst and fopen.

itemcount

Returns the number of items within an option set list.

itemcount itemtype intvar

itemtype	A keyword or keyword combination that represents an option set item type. Note that only one of these selections may be specified per **itemcount** command. For example: **itemcount data options "my modem"** i0
intvar	Will receive the number of items in the specified set.

Comments

It is important to note that **itemcount** returns a non-zero-based value. The index values for the items being counted, however, *are* zero-based.

See also

itemfind, itemname, itemcreate and itemremove; $DATAOPTIONS, $DIALOPTIONS, $FAXOPTIONS, $FTPOPTIONS, $TCPIPOPTIONS, $TELNETOPTIONS, $WWWOPTIONS, $PROTOCOL, $TERMINAL, $DATACONNECT, $FAXCONNECT, $MODEMCONNECT, $TERMFONT, $PHONECARD, $TERMCOLORS and $WINCOLORS in "*System Variables*."

itemcreate

A Creates a new item in an option set list.

itemcreate {PROTOCOL name protocol | index | string [intvar]} |
{TERMINAL name terminal | index | string [intvar]} |
{itemtype name index | string [intvar]}

itemtype	A keyword or keyword combination that represents an option set item type. Note that only one of these selections may be specified per **itemcreate** command. For example: **itemcreate data options "my modem" i0**
name	A string variable or constant, specifying the name of the item to create. For valid protocol keywords and indices, see "*Protocol Names and Indices*" on page 59. Terminal keywords and indices are listed in "*Emulation Names and Indices*" on page 60.
intvar	Will receive the index value of the new item, if specified.

Comments

itemcreate returns FAILURE if the item could not be created. **itemcreate** will fail if either the *Connection Directory* or *Setup* is open at the time it executes. **itemcreate** always returns FAILURE for the MODEM CONNECTION itemtype.

See also

itemfind, itemname, itemcreate and itemremove; $DATAOPTIONS, $DIALOPTIONS, $FAXOPTIONS, $FTPOPTIONS, $MODEMCONNECT, $TCPIPOPTIONS, $TELNETOPTIONS, $WWWOPTIONS, $PROTOCOL, $TERMINAL, $DATACONNECT, $FAXCONNECT, $TERMFONT, $PHONECARD, $TERMCOLORS and $WINCOLORS in "*System Variables*."

itemfind

A Searches an option set for a specifically named item.

itemfind {itemtype name [EXACT]} | NEXT [strvar] [intvar]

itemtype	A keyword or keyword combination that represents an option set item type. Note that only one of these selections may be specified per **itemfind** command. For example: **itemfind data options "my options" i8**
name	A string variable or constant containing a full or partial item name.
EXACT	An optional parameter that causes **itemfind** to find an exact match rather than a case-insensitive substring match.
NEXT	Continues searching the option set for another occurrence of the previously specified item name.
strvar	If specified, will return the complete name of an option set item if the search is successful.
intvar	Will receive the index value of the item, if specified.

Comments

itemfind reports FAILURE if the item could not be found. Otherwise, it returns the index of the first matching item.

See also

itemfind, itemname, itemcreate and itemremove; $DATAOPTIONS, $DIALOPTIONS, $FAXOPTIONS, $FTPOPTIONS, $TCPIPOPTIONS, $TELNETOPTIONS, $WWWOPTIONS, $PROTOCOL, $TERMINAL, $DATACONNECT, $FAXCONNECT, $MODEMCONNECT, $TERMFONT, $PHONECARD, $TERMCOLORS and $WINCOLORS in "*System Variables*."

itemname

A Returns the name of an option set item into a string.

itemname itemtype index strvar

itemtype	A keyword or keyword combination that represents an option set item type. Note that only one of these may be specified per **itemname** command. For example: **itemname data options 0 s8**
index	An integer constant or variable, specifying the zero-based index of the option set item whose name is to be returned.
strvar	Will contain the name of the item specified by the *index* value.

Comments

itemname returns FAILURE if there is no item associated with that *index*, or if the *index* exceeds the item count in the specified option set.

See also

itemfind, itemname, itemcreate and itemremove; $DATAOPTIONS, $DIALOPTIONS, $FAXOPTIONS, $FTPOPTIONS, $TCPIPOPTIONS, $TELNETOPTIONS, $WWWOPTIONS, $PROTOCOL, $TERMINAL, $DATACONNECT, $FAXCONNECT, $MODEMCONNECT, $TERMFONT, $PHONECARD, $TERMCOLORS and $WINCOLORS in "*System Variables*."

itemremove

A Removes a script- or user-created entry from an option set list.

itemremove itemtype index | string

itemtype	A keyword or keyword combination that represents an option set item type. Note that only one of these may be specified per **itemremove** command. For example: **itemremove data options "my options"**
index	An integer constant or variable, specifying the zero-based index of the item to be removed.

string	A string constant or variable, specifying the name of the item to be removed. The string must match the item’s name exactly, including upper- and lowercase. **itemname** can be used to find an item name based on a partial string.

Comments

itemremove returns FAILURE if the option set item could not be removed. **itemremove** will fail if ***any*** of the following conditions exist at the time it executes:

- *Setup* is open.
- The *Connection Directory* is open.
- Another instance of Procomm Plus is running.
- You’re attempting to remove the last item remaining in an option set.
- You’re attempting to remove an item that is currently selected in *Setup*.
- You’re attempting to remove an itemtype of MODEM CONNECTION.
- You’re attempting to remove the “***direct connect - None***” in the ***Modem Connection*** option set.
- You’re attempting to remove one of the first 9 items or the “***IND$FILE***” item in the ***Current Transfer Protocol*** option set, located in the ***Data, Transfer Protocol*** settings panel in *Setup*.
- You’re attempting to remove one of the first 35 items or the “***RIPscrip 1.54***” item in the ***Current Terminal*** option set, located in the ***Data, Terminal Options*** settings panel in *Setup*.

See also

itemfind, itemname, itemcreate and itemremove; $DATAOPTIONS, $DIALOPTIONS, $FAXOPTIONS, $FTPOPTIONS, $TCPIPOPTIONS, $TELNETOPTIONS, $WWWOPTIONS, $PROTOCOL, $TERMINAL, $DATACONNECT, $FAXCONNECT, $MODEMCONNECT, $TERMFONT, $PHONECARD, $TERMCOLORS and $WINCOLORS in “*System Variables*.”

itoa

Converts an integer to an ASCII string and stores it in a string variable.

itoa integer strvar

integer	The integer value to convert.
strvar	Will contain the converted value.

See also

ftoa, ltoa, numtostr, atoi and strfmt.

kermserve

A Issues a Kermit server command, returning SUCCESS or FAILURE based on the outcome. **kermserve** is only valid if the host is in the Kermit server mode.

kermserve {GETFILE | SENDFILE filespec} | FINISH | LOGOUT

GETFILE filespec	Receives a file from a remote system. *filespec* is any filename that's valid on the remote system.
SENDFILE filespec	Sends a file to a remote system. *filespec* is any valid DOS filename, including the extension. If the script executes a **while** loop to monitor the status of an on-going file transfer, you may find it useful to include the **yield** command within the loop. The **yield** command causes ASPECT to release its processing time to the system, which can improve file transfer performance.
FINISH	Logs out of Kermit server mode.
LOGOUT	Logs out of Kermit server mode and logs off the host.

Comments

Under ASPECT, file transfers are asynchronous operations, meaning that they can run independently of script execution. A **kermserve** GETFILE or **kermserve** SENDFILE command only initiates a file transfer attempt. While the transfer is processing, ASPECT executes the next commands in the script. If the script needs to wait for the transfer to complete, a **while** loop should be used to "stall" the script.

If the script executes a **while** loop to monitor the status of an on-going file transfer, you may find it useful to include the **yield** command within the loop. The **yield** command causes ASPECT to release its processing time to the system, which can improve file transfer performance.

See also

sendfile and getfile.

keyflush

Clears accumulated keystrokes from the keyboard buffer. Any unprocessed keystrokes will be lost.

keyflush is only valid when **set aspect keys** is on.

keyflush

See also

keyget; $KEYHIT in “*System Variables*” and set aspect keys in “*Set and Fetch Statements*.”

keyget

A Retrieves a key pressed by the user and optionally stores it in the specified integer variable.

keyget [intvar [integer]]

intvar	The returned key value.
integer	An optional timeout value in seconds.

Comments

Keys can only be retrieved by the script when Procomm Plus’s main window has the input focus.

The value placed in the numeric variable depends upon the key pressed. The keypress shift state is contained in the high order byte, while the virtual key code is the low order byte. If a timeout value is specified, and **keyget** does not receive a key value within that time, a value of zero is assigned to the target *intvar*.

To place the ASCII character corresponding to a virtual key code into a string variable, use the **keytoansi** or **keytooem** command.

See also

keystate, keytoansi, keytooem, termvkey and sendvkey; $KEYHIT in “*System Variables*” and set aspect keys in “*Set and Fetch Statements*.”

keystate

Determines whether a key corresponding to a virtual key code is currently pressed.

keystate keyval intvar

keyval	The virtual key code of the key to be tested. The key code does not include the shift state.

intvar	Set to 1 if the key is currently pressed, 0 otherwise.

Comments

keystate can also be used to test the toggled state of the <Caps Lock>, <Num Lock> and <Scroll Lock> keys. It returns a 1 if these keys are toggled ON and a 0 if they're toggled OFF. For a list of virtual key code values, see "*Virtual Key Codes*" on page 603.

See also

keyget.

keytoansi

A Converts a key value to its corresponding ANSI character value. A key value is composed of a keyboard shift state and a virtual key value.

keytoansi keyval intvar

keyval	The key value to convert.
intvar	The converted ANSI value. If there is no corresponding ANSI key value, *intvar* will be -1. Otherwise, *intvar* will be an ANSI character value in the range of 0 to 255.

See also

keyget, termkey, sendkey, keystate, keytooem and ansitokey.

keytooem

A Converts a key value to its corresponding OEM character value. A key value is composed of a keyboard shift state and a virtual key value.

keytooem keyval intvar

keyval	The key value to convert.
intvar	The converted OEM value. If there is no corresponding OEM key value, *intvar* will be -1. Otherwise, *intvar* will be an OEM character value in the range of 0 to 255.

See also

keyget, termkey, sendkey, keystate, keytoansi and oemtokey; set aspect codepage in *“Set and Fetch Statements*.”

listbox

Adds a list box to an ASPECT dialog box, allowing a user to select one or more items from a list.

listbox id left top width height itemlist [tabstring] SINGLE | MULTIPLE strvar [HSCROLL] [SORT]

id	An integer constant, specifying the control id for the **listbox. dlgevent** will report this value when a **listbox** item is selected.
left top width height	Integer constants specifying the position and size of the list box. Coordinates are relative to the left and top edges of the dialog box, in Dialog Box Units or DBUs. For more information on DBUs, see "*Dialog Box Units*" on page 48.
itemlist	A string variable or constant containing a list of items to display within the **listbox**. Each item is separated by a comma.
tabstring	An optional, comma-separated string variable or constant containing numbers in DBUs indicating column locations for tabs contained within the list. If only one tab stop is specified, the **listbox** will be initialized with multiple stops separated by the single tab stop value. If more than one tab stop is given, then only those tab stops will be initialized.
SINGLE strvar	Allows a single item to be selected from the list. It will be stored in *strvar.*
MULTIPLE strvar	Allows one or more items to be selected from the list. They will be stored in *strvar*, separated by commas.
HSCROLL	Scroll bars will be placed on the **listbox**, allowing the user to scroll the listed items horizontally. Horizontal scrolling is appropriate only for list entries which contain no tab characters.
SORT	The contents of the **listbox** will be displayed in alphabetical order.

Comments

The **listbox** command can only occur within a **dialogbox** statement group. The *strvar* parameter can be initialized to a specific item, causing that item to be pre-selected when the dialog box is displayed.

See also

combobox, fcombobox, dlglist, dlgupdate, dlgevent, dialogbox and flistbox.

locate

Positions the terminal cursor to the location specified by row and column. Characters received from the remote system are displayed at the current cursor position.

locate row column

Comments

locate has no effect while the *Terminal window* is in *Scrollback* mode. Script execution merely continues to the next command.

If **locate** is executed while text is being marked in the *Terminal window*, however, script execution is suspended. After the marking operation is completed or cancelled, the **locate** command is executed and the script resumes normally.

See also

getcur; $TERMROWS, $TERMCOLS, $ROW and $COL in "*System Variables*" and the set terminal command family in "*Set and Fetch Statements*."

long

Defines a global or local long variable.

long name[=expression][,name[=expression]]...

name	The name of the new variable. This name must follow the standard ASPECT naming conventions. When defining an array, the subscript expression must result in a constant value.
= expression	An initializing expression. It can specify operators, constants or previously-declared numeric variables, or a function which returns a numeric result. The initializing expression cannot be used when defining an array.

Comments

Up to 256 global long variables can be defined, as well as 256 global long arrays. Local long definitions are limited by the available stack space at the time the procedure or function is called. For more information, see "*User-defined Variables*" on page 28 and "*Global and Local Variables*" on page 28.

See also

integer, float, param and string.

longjmp

Returns to a location within a script previously marked with a **setjmp** command.

longjmp id integer

id	An integer id specifying the desired jump point previously set using **setjmp**.
integer	An integer value to return to the **setjmp** command. This value can be used in commands following **setjmp** for reporting or decision-making.

Comments

The marked location is the command following the **setjmp**. The marked location remains active until a return from the function where it was set, or until another routine uses a **setjmp** with the same *id* value.

longjmp does not update variables passed by reference to the procedure in which the **longjmp** occurs, nor does it update any variables passed by reference for any procedure calls between the **setjmp** and **longjmp** commands that have not been returned.

See also

goto, return and setjmp.

loopfor

Skips the remaining commands in the **for** command block, and increases or decreases the control variable in a **for** statement group based on the step value specified in the **for** command itself.

loopfor

Comments

The control variable is tested to determine whether the limit has been reached. If another iteration should be performed, execution continues with the command following the **for** command. For further information, see the **for** command.

See also

for, endfor and exitfor.

loopwhile

Jumps to the conditional test specified in the **while** command. If another iteration should be performed, execution continues with the command following the **while** command.

loopwhile

See also

while, endwhile and exitwhile.

ltimeelapsed

Converts a *timeval* into a time string.

ltimeelapsed timeval strvar

timeval	A long *timeval*.
strvar	The resulting time string.

Comments

ltimeelapsed can be used to convert the difference between any two *timevals* into a string that contains the number of hours, minutes and seconds that have elapsed between the two values. Hours can exceed 24.

For example, **ltimeelapsed** can be used to convert the connect time field returned by the **dialstats** command.

See also

dialstats; $LTIME in "*System Variables*."

ltimeints

Converts a long time/date value in system time format into integer variables containing the individual date and time values.

ltimeints timeval intvar intvar intvar intvar intvar intvar

timeval	The timeval to convert.
intvar	The year. A four-digit integer from 1970 to 2106.
intvar	The month. An integer from 1 to 12.
intvar	The day. An integer from 1 to 31.
intvar	The hour. An integer from 0 to 23.
intvar	The minutes. An integer from 0 to 59.
intvar	The seconds. An integer from 0 to 59.

See also

ltimestring, ltimestrs and intsltime; $LTIME in “*System Variables*.”

ltimemisc

Returns miscellaneous information about a given *timeval*.

ltimemisc timeval intvar intvar intvar

timeval	A long value.
intvar	Will contain an integer from 1 to 7, denoting the day of the week from 1 for Sunday through 7 for Saturday.
intvar	Will contain an integer from 1 to 365 or 366, indicating the number of the day within the current year.
intvar	Will equal 1 if the current year is a leap year; 0 otherwise.

See also

ltimeints. weekdaystr and monthstr; $LTIME in “*System Variables*.”

ltimestring

Converts a long value in system time format into a string variable representing date and time.

ltimestring timeval strvar

timeval	A long value.
strvar	The converted time/date string.

Comments

The string will be formatted based on the information contained within the ***Long Date Format*** fields of the Windows Control Panel ***International*** section.

See also

ltimeelapsed, ltimeints and ltimestrs; $LTIME in “*System Variables*.”

ltimestrs

Converts a long value in system time format into two string variables representing date and time.

ltimestrs timeval strvar strvar

timeval	The *timeval* to convert.
strvar	The converted date.
strvar	The converted time.

Comments

The string will be formatted based on the information contained within the ***Short Date and Time Format*** fields of the Windows Control Panel **International** section.

See also

ltimeints, ltimestring and strsltime; $LTIME in “*System Variables*.”

ltoa

Converts a long to an ASCII string and stores it in a string variable.

ltoa long strvar

long	The long value to convert.
strvar	Will contain the converted value.

See also

atol, ftoa, itoa, strfmt and numtostr.

makepath

Creates a path and/or file specification from individual components.

makepath strvar drive pathname filename extension

strvar	Will contain the created pathname.
drive	A string, containing a drive letter.
pathname	The path information to include in the pathname.
filename	The filename to include in the result.
extension	The extension to include in the result.

See also

addfilename, splitpath, getfilename and getpathname.

mapisend

Allows the user to send mail whenever the MAPI interface is loaded and accessible.

mapisend string string string string string string [FILE] string string [strvar]

string	A string containing your mail logon ID
string	A string containing your mail password.
string	A string containing the names of the recipients of the message. Multiple recipients may be selected by separating the names with semicolons.
string	A string containing the names of the people receiving a “carbon copy” of the message.
string	A string containing the names of recipients receiving a “blind carbon copy.”
string	The subject line text for the mail message.
FILE	A keyword which is used to indicate that the following string contains the name of a text file to be used as the message contents.
string	A string containing either the contents of the message or the name of a file if the FILE parameter was specified.
string	A string which allows you to attach file(s) to the message. If you attach multiple files, each filename must be separated by a semicolon.
strvar	An optional parameter which will contain a MAPI error if an error occurs.

Comments

If you are using Microsoft Exchange, the logon name you specify must match the name of a Microsoft Exchange Profile that is currently configured on the computer. In addition, the password argument is not used, and can thus be specified as a null string.

mciexec

A Executes a high-level multimedia command.

mciexec string

string	A valid multimedia command.

Comments

mciexec is used with multimedia commands like open, set, play and close to control a multimedia device. **mciexec** requires Multimedia Extension software or direct Windows multimedia support.

mcisend

A Allows you to issue an MCI command and retrieve the error message or the result of the command.

mcisend string [strvar [strvar]]

string	A valid multimedia command.
strvar	Will store any error messages that may be returned.
strvar	Will store any result message that the MCI commmand may provide.

Comments

Unlike **mciexec**, **mcisend** will not automatically display an error message; you must include coding to make the message appear.

See also

mciexec

memaddress

Returns the memory address associated with the specified memory ID.

memaddress id longvar

id	The id referencing a memory block created with a **memalloc** command.
long	The memory address returned.

Comments

The value returned by a **memaddress** has no other value within the ASPECT command set. It is designed so that you can reference the memory with typical standard library APIs that accept pointer arguments. Note that it is your responsibility to ensure that no data is written outside the area allocated: doing so could likely cause some form of application or system error. Allowing an ASPECT script to manipulate a memory block while a DLL or outside process is using it will produce similar results.

memalloc

A Allocates a block of contiguous memory for use by an ASPECT script.

memalloc id memlength [SHARED]

id	An integer value which will be used to reference the allocated memory block.
memlength	An integer whose value is treated as an unsigned value, specifying the size, in bytes, to be allocated.
SHARED	Allows the memory block to be shared with *child* scripts. Child scripts, as long as the are aware of the memory block ID, can perform any file I/O operations on the SHARED memory without having to **memalloc** the block. Note, however, that any changes to the memory block are in effect when the parent script resumes control.

Comments

Use **memavail** to determine the amount of memory available for use. The results of the allocation can be tested with **if** SUCCESS.

If the *id* given to **memalloc** equals an *id* already in use by an existing memory block, the existing block will be destroyed and freed. The *id* will then reference the newly-created block.

The maximum size of a block allocated by **memalloc** is 64K. If more memory is required, subsequent **memalloc** calls can reserve other contiguous blocks. If a size of zero is specified, **memalloc** will create a valid *id* with a block size of zero, and report SUCCESS. Because of data alignment issues, the actual size of the allocated block may be slightly larger than the specified amount. **memsize** will always return the actual size of the allocated block. Memory accesses are valid up to the actual size of the allocated block. ASPECT will release unfreed memory allocated by a script, but users are advised to employ **memfree** to release a block when it's no longer needed.

➥ *The allocated memory block will be un-initialized. To be certain of its contents, use the* ***memset*** *command.*

See also

memavail, memfree, memrealloc, memread, memwrite and memset.

memavail

Returns the amount of contiguous free memory available for allocation, or for running other programs into a long variable.

memavail longvar

See also

memalloc, memfree and memrealloc.

memchr

A Searches for the first occurrence of the specified character in an allocated memory block.

memchr id memindex character [intvar]

id	The *id* of the memory block to search.
memindex	The starting position of the search within the target memory block.
character	The desired character value. An integer value from 0 to 255 inclusive.

intvar	Will contain the position of the first occurrence of the desired character in the block.

Comments

If the desired character is not found **memchr** will return FAILURE. *Intvar* in that case will contain the number of characters from the starting point of the search to the end of the allocated block.

See also

memalloc, memcmp, memgetc and memicmp.

memcmp

A Performs a byte-by-byte comparison of two memory locations.

memcmp id memindex id memindex memlength [intvar]

id	The control *id* of an allocated memory block.
memindex	The starting point of the comparison in the first block. An integer value, equal to the number of bytes from the start of the block.
id	The control *id* of an allocated memory block.
memindex	The starting point of the comparison in the second block. An integer value, equal to the number of bytes from the start of the block.
memlength	The number of bytes to compare.
intvar	Will return 1 if the first location is greater, -1 if the second is greater, and 0 if the two are equal for the length of the search.

Comments

Similar to the **memcmp** function in the “C” language, ASPECT’s **memcmp** compares each successive byte in the allocated block(s) until they do not have the same value or until the number of bytes specified in length have been compared. It reports SUCCESS if the comparisons were equal for the length of the test; FAILURE otherwise. If the same id is specified for both memory block id’s, **memcmp** will compare data within a single block.

See also

memalloc, memicmp, memchr, memgetc, memmove, memputc, memread, memset and memwrite.

memfree

Releases a block of memory previously reserved by **memalloc** or **memrealloc**.

memfree id

id	The *id* of the memory block to release.

See also

memalloc and memrealloc.

memgetc

Retrieves a single character from a block of memory reserved by **memalloc**.

memgetc id memindex intvar

id	The *id* of the memory block to access.
memindex	An integer value, equal to the number of bytes from the start of the block, specifying the location of the character to be read.
intvar	The value read, ranging from 0 to 255 inclusive.

See also

memalloc, memputc and memread.

memicmp

A Performs a byte-by-byte, case-insensitive comparison of two memory locations.

memicmp id memindex id memindex memlength [intvar]

id	The control *id* of an allocated memory block.

memindex	The starting point of the comparison in the first block. An integer value, equal to the number of bytes from the start of the block.
id	The control *id* of an allocated memory block.
memindex	The starting point of the comparison in the second block. An integer value, equal to the number of bytes from the start of the block.
memlength	The number of bytes to compare.
intvar	Will return 1 if the first location is lexicographically greater, -1 if the second is greater, and 0 if the two are equal for the length of the search.

Comments

Similar to the **memicmp** function in the "C" language, ASPECT's **memicmp** compares each successive byte in the allocated block(s) until they do not have the same value or until the number of bytes specified in length have been compared. It reports SUCCESS if the comparisons were equal for the length of the test; FAILURE otherwise. If the same id is specified for both memory block id's, **memicmp** will compare data within a single block.

See also

memalloc, memcmp, memset and stricmp.

memmove

Copies characters from one location to another, either between different blocks of memory or within the same block.

memmove id memindex id memindex memlength

id	The destination memory block *id*.
memindex	The starting point of the **memmove** into the destination block. An integer value, equal to the number of bytes from the start of the block.
id	The source memory block *id*.
memindex	The starting point of the **memmove** from the source block. An integer value, equal to the number of bytes from the start of the block.

memlength	The number of characters to move.

Comments

If the source and destinations overlap, the source is copied before it is overwritten. To clear a memory location, use the **memset** command.

See also

memalloc, memgetc, memset and memwrite.

memputc

Writes a single character to a location within a memory block allocated by **memalloc**.

memputc id memindex character

id	The *id* of the memory block to access.
memindex	An integer value, equal to the number of bytes from the start of the block, specifying the location of the character to be written.
character	The value to write, either a character constant or an integer ranging from 0 to 255 inclusive.

See also

memcmp, memgetc, memputc, memread, memset and memwrite.

memread

Reads a block of data from an allocated memory block.

memread id memindex numvar | {strvar strlength}

id	The *id* of the memory block to access.
memindex	An integer value, equal to the number of bytes from the start of the block, specifying the location of the data to be read.
numvar	A float, integer, or long variable.
strvar strlength	A string variable, with a specified number of characters to be read.

Comments

ASPECT automatically determines the number of bytes to be read if a numeric variable is specified for **memread**. An integer or long argument will cause 4 bytes to be read from the block; a float 8 bytes. A string will read the number of bytes specified in *strlength*.

See also

memalloc, memgetc and memwrite.

memrealloc

A Changes the size of a block of memory previously reserved by **memalloc**.

memrealloc id memlength

id	The *id* of the target memory block.
memlength	A value specifying the new size of the block. If zero is specified, **memrealloc** will adjust the block size accordingly and report SUCCESS.

Comments

If the *id* of the block does not exist, that is, if a block was not previously reserved with this *id* using **memalloc**, the **memrealloc** command will behave exactly like **memalloc**. A new block of memory will be reserved and given the *id* value for identification. **memrealloc** will report FAILURE if insufficient memory is available to accommodate an expansion. The target memory block will still be valid, with its original size intact.

See also

memalloc, memavail and memfree.

memset

Initializes a block of memory to a specified value.

memset id memindex character memlength

id	The *id* of the target memory block.
memindex	An integer value, equal to the number of bytes from the start of the block, specifying the starting position of **memset** operation.

character	An integer value from 0 to 255 inclusive, specifying the value to set.
memlength	The number of character positions to set.

Comments

memset is useful in initializing a freshly **memalloc**ated block, or in resetting a block for new operations.

See also

memalloc, memmove, memread and memwrite.

memsize

Reports the size of a memory block allocated with **memalloc**. It can be used to find the size of a block before read or write operations are performed.

memsize id intvar

id	The *id* of the target memory block.
intvar	The size in bytes of the allocated block, represented as an unsigned integer value.

See also

memalloc, memavail, memfree and memrealloc.

memwrite

Writes a block of data to an allocated memory block.

memwrite id memindex number | {string strlength}

id	The *id* of the target memory block.
memindex	An integer value, equal to the number of bytes from the start of the block, specifying the starting position of **memwrite** operation.
number	A float, integer, or long constant or variable to be written.

string strlength	A string variable, with a specified number of characters to be written.

Comments

ASPECT automatically determines the number of bytes to be written if a numeric constant or variable is specified for **memwrite**. An integer or long argument will cause 4 bytes to be written to the block; a float 8 bytes. A string writes the number of bytes specified in *strlength*.

See also

memalloc, memputc, memset and memread.

menubar

A Creates a new script menu bar.

menubar intvar

intvar	Use the value returned to this variable with the **menupopup** and **menuitem** commands to place menus on the bar and the **menushow** command to display the completed menu bar when all menu pop-ups and menu items have been added to it.

Comments

Any number of menu bars can be created from a script. When the script terminates, existing menu bars it generated will be destroyed automatically.

The **menupopup** and **menuitem** commands can be used to attach new items to Procomm Plus's default menu bar. These items will automatically be destroyed when the script that created them is terminated. If you have created a menu bar it will remain available when you **chain** or **execute** another script.

See also

menupopup, menuitem and menushow; $PWMENUBAR in "*System Variables*."

menucheck

Places or removes a check mark on a menu item created by an ASPECT script.

menucheck integer OFF | ON

integer	The menu item to be checked or unchecked. This is the same value used in creating the menu item.

Comments

A menu pop-up, or an item on the menu bar itself cannot be checked. The **menucheck** command only works for a currently displayed menu item that was created by an ASPECT script.

See also

menubar and menuitem.

menuitem

A Adds an item to a menu pop-up or menu bar.

menuitem integer {integer string [string]} | SEPARATOR

integer	The integer value, previously returned from a **menubar**, **menupopup**, or **menupopupid** command, identifying the menu upon which the item is placed.
integer	The id value of the new menu item. Valid ids can range from 1 up to 299. The id is returned in $ASPMENU when the menu items is selected by the user.
string	The menu item text.
string	If specified, will be displayed on the status line as help text whenever the menu item is highlighted.
separator	Adds a menu separation line. A separator does not require an id.

Comments

The **menuitem** command allows an ampersand (“&”) to precede the “accelerator” character on a menu. The character will be displayed with an underscore in the menu, indicating that the user can hold down the <Alt> key and press this character to select the item from the menu. However, the ampersand should not precede a numeric character. If the ampersand is not specified, Windows defaults to the first character of the menu text as the accelerator key.

Created menu items remain available when you **chain** or **execute** another script.

See also

disable, enable, menubar, menucheck, menupopup, menupopupid, menushow, menustate, menuselect and when $ASPMENU; $ASPMENU in *“System Variables*.”

menuitemcount

Returns the number of items on a menu bar or pop-up menu.

menuitemcount integer intvar

integer	The ID of the menu pop-up to be counted.
intvar	The returned number of items on the menu.

Comments

The returned value can be used to assist in enumerating pop-up menus using **menupopupid**. Menu item separators, the bars that separate two menu items, are included in the count.

See also

menubar, menuitem, menupopup, menuselect.

menupopup

A Adds a pop-up menu to an existing pop-up menu or menu bar.

menupopup integer string [string] intvar

integer	Identifies the menu or menus to which the new menu will be added. This value is taken from a previous **menubar**, **menupopup**, or **menupopupid** command. The $PWMENUBAR system variable contains the menu bar id currently used by Procomm Plus. Several pop-up menu ids are reserved by Procomm Plus and may not be used; for details, see the **menuitem** command.
string	The menu text.

string	If specified, will be displayed on the Procomm Plus status line as help text for the menu pop-up.
intvar	The id value of the new menu.

Comments

"Nested" menus can be built by adding multiple pop-up menus atop one another. Simply use the integer variable returned from the **menupopup** command, or one of Procomm Plus's predefined menu pop-up ids as the integer value when creating nested pop-ups.

The **menupopup** command allows an ampersand ("&") to precede the "accelerator" character on a menu. The character will be displayed with an underscore on the menu, indicating that the user can hold down the <Alt> key and press this character to select the item from the menu. However, the ampersand should not precede a numeric character. If the ampersand is not specified, Windows defaults to the first character of the menu text as the accelerator key.

If you have created any menu pop-ups, they will remain available when you **chain** or **execute** another script.

See also

menubar, menuitem and menushow; $PWMENUBAR in "*System Variables*."

menupopupid

A Returns the menu ID of a pop-up menu on a menu bar or other pop-up menu.

menupopupid integer integer intvar

integer	The owner menu ID.
integer	The position on that menu where the pop-up is located.
intvar	The returned ID of the menu pop-up.

Comments

If there is no pop-up at the corresponding position (the second *integer*), SUCCESS/FAILURE is set, and 0 (zero) is returned to the *intvar*.

See also

menupopup and menushowpopup.

menuselect

Selects a Procomm Plus or ASPECT menu item.

menuselect ASPMENU | PWMENU integer

ASPMENU | PWMENU	Specifies whether the menu item to be selected originates from a script or from Procomm Plus itself.
integer	The id of the menu item to be selected.

Comments

Procomm Plus's menu items can be selected even when a different menu bar is displayed, but menu items created by an ASPECT script must be displayed in order to be selected. A menu item cannot be selected when it is **disable**d. You cannot retrieve the state of a menu pop-up or an item on the menu bar itself. Procomm Plus's main window must have the input focus for **menuselect** to be used.

➥ *Two useful files are provided with Procomm Plus that can be* ***#include****d into your scripts. These files are* ***menuids.inc****, which defines all of Procomm Plus's menu values, and* ***vkeys.inc****, which defines the virtual key codes as shown in "Virtual Key Codes" on page 603. Both of these files are installed to the default ASPECT directory.*

See also

menuitem and menucheck.

menushow

Displays the menu bar corresponding to the specified menu bar id value. **menushow** can be used to display updates to the current menu bar.

menushow integer | PWMENU

integer	Identifies the menu bar to show. This value is taken from a previous **menubar** command. The $PWMENUBAR system variable contains the menu bar id currently used by Procomm Plus. If the value is 0, the current **menubar** will be removed.
PWMENU	If specified, the default Procomm Plus menu bar will be restored.

See also

menubar, menuitem, menupopup and when $ASPMENU; $ASPMENU in *"System Variables."*

menushowpopup

A Displays a floating pop-up menu.

menushowpopup integer integer integer LEFT | CENTER | RIGHT LEFT | RIGHT

integer	The pop-up menu ID.
integer	The X screen coordinate for the menu.
integer	The Y screen coordinates for the menu.
LEFT	The left side of the pop-up menu is aligned with the specified X screen coordinate.
CENTER	The pop-up menu is centered relative to the specified X screen coordinate.
RIGHT	The right side of the pop-up menu is aligned with the specified X screen coordinate.
LEFT	The left mouse button is tracked.
RIGHT	The right mouse button is tracked.

See also

menuitem, menupopup, and menupopupid; $ASPMENU in *"System Variables."*

menustate

Reports the state of a Procomm Plus or ASPECT menu item.

menustate ASPMENU | PWMENU integer intvar

ASPMENU | PWMENU	Specifies whether the menu item originates from a script or from Procomm Plus itself.
integer	The id of the menu item to be tested; a variable or constant.

intvar	A bitflag value, with the following possible values: 0x01 for grayed, 0x02 for disabled or 0x04 for checked.

Comments

If the menu item was generated by an ASPECT script, **menustate** will only work if the menu item is currently displayed. Procomm Plus's menu items, on the other hand, can always be tested. When a menu item is grayed, it is **disable**d by default. You cannot test the state of a menu pop-up, or a menu item on the menu bar itself.

➥ *Two useful files are provided with Procomm Plus that can be* ***#include****d into your scripts. These files are* ***menuids.inc****, which defines all of Procomm Plus's menu values, and* ***vkeys.inc****, which defines the virtual key codes as shown in "Virtual Key Codes" on page 603. Both of these files are installed to the default ASPECT directory.*

See also

menuitem, menubar, menupopup, menucheck and menuselect.

metafile

A Displays a Windows metafile (**.wmf**) or enhanced metafile (**.emf**) graphic within a User window or dialog box. **metafile** can only be tested for SUCCESS/FAILURE when it appears in a User window.

Dialog box format

metafile id left top width height filespec

id	A unique integer constant value given to the **metafile**, used by the **dlgupdate** command to refresh the metafile.
left top	Integer constants which determine the top left corner position of the **metafile** with respect to the top left corner of the dialog box. Values are in Dialog Box Units or DBUs. For more information on DBUs, see "*Dialog Box Units*" on page 48.
width height	The **metafile** display size integer constants. Values are expressed in DBUs.
filespec	String constant or variable specifying the name of the **metafile** to display. If a path is not given, the default Aspect Path is assumed.

See also

bitmap, dialogbox, dlgevent, dlgupdate, icon, and iconbutton.

User Window form

metafile id left top width height filespec BACKGROUND | USERWIN

id	A unique integer value given to the **metafile,** used by the **objremove** command to remove the **metafile**.
left top	Integer values which determine the top left corner position of the metafile with respect to the top left corner of the User window or background graphic. Values are in User Window units or UWUs. For more information on UWUs, see "*User Window Units*" on page 48.
width height	The **metafile** display size in UWUs.
filespec	String constant or global variable specifying the name of the **metafile** to display in the window. If a path is not given, the current Aspect Path is assumed.
USERWIN | |BACKGROUND	Specify USERWIN to "fix" the top left corner of the **metafile** in position relative to the upper left corner of the User window. If BACKGROUND is specified, the **metafile** stays over the same spot on the background graphic; if the size of the background graphic changes, the **metafile** will also change in proportion.

Comment

Metafile are usually identified with a **.wmf** extension, and these files must contain the placeable metafile format that is typical within Windows. Enhanced metafile format files, usually identified with a **.emf** extension, are also supported.

The User window must exist before a **metafile** can be displayed. The **uwinpaint** command must be issued to display User window changes. A **metafile** cannot be selected by mouse click. The **metafile** will fill the rectangle defined by the width and height values, but the aspect ratio will not be automatically maintained.

See also

uwincreate, uwinpaint, objpaint, objcoord, objpointid, objremove, metafilebkg, bitmap, hotspot, icon, iconbutton, pushbutton, dllobject and when $OBJECT; $OBJECT in "*System Variables*."

metafilebkg

A Places a Windows metafile or enhanced metafile background in the User window.

metafilebkg CENTER | LEFT | RIGHT CENTER | TOP | BOTTOM SCALE | EXACT filespec

CENTER | LEFT | RIGHT	The horizontal position of the metafile background with respect to the User window. If the User window is based on the total screen size in pixels rather than the Terminal window and the metafile is larger than the User window, the metafile will be clipped to fit.
CENTER | TOP | BOTTOM	The vertical position of the metafile background with respect to the User window.
SCALE | EXACT	Determines how the background will appear. If set to SCALE, Procomm Plus displays the graphic using arbitrarily scaled axis and the image will be stretched or squeezed to fill the entire image area. This is called anisotropic mapping. If set to EXACT, the original graphic's height and width ratio is maintained, which preserves the exact shape of the image. Because of this ratio, the image may not fill the entire User window. This is called isotropic mapping.
filespec	The name of the metafile to display as the background. If a path is not specified, the current Aspect Path is assumed.

Comments

Metafile are usually identified with a **.wmf** extension, and these files must contain the placeable metafile format that is typical within Windows. Enhanced metafile format files, usually identified with a **.emf** extension, are also supported.

See also

bitmapbkg, uwincreate, uwinpaint, uwinremove and metafile.

metakey

Executes the specified *Meta key.*

metakey ALT | ALTSHIFT | ALTCTRL | ALTCTRLSHIFT integer

integer	An integer value from 0 to 9, designating the metakey to execute.

Comments

Use the **set metakeyfile** command to load individual keyboard *Meta Key* files. Issue the statement **set aspect spawn** ON before the **metakey** command if the desired metakey is defined to run a script file. **set metakeys** can be used to control display of the *Meta Keys*.

See also

termkey; set metakeyfile, set metakeys and set aspect spawn in "*Set and Fetch Statements*."

mkdir

A Creates a new directory using a path and/or directory name you provide.

mkdir pathname

pathname	The desired path or directory name. If a pathname is not provided, the User Path is assumed for the default directory.

See also

chdir, getdir and rmdir; $USERPATH in "*System Variables*"; set aspect path in "*Set and Fetch Statements*."

monthstr

Returns the name of the specified month.

monthstr integer strvar [SHORT]

integer	A value from 1 to 12.
strvar	Will contain the name of the month.
SHORT	If specified, the returned name will be abbreviated.

See also

ltimeints, ltimestring, ltimestrs and weekdaystr; $LTIME in "*System Variables*."

mspause

Pauses script execution for the specified number of milliseconds.

mspause integer

Comments

Unlike the **pause** command, **mspause** can’t be aborted with the <Ctrl><Break> key sequence. The delay has a maximum length of 1000 milliseconds. The number of milliseconds specified is subject to a 55 millisecond granularity.

See also

pause.

nexttask

A Returns subsequent tasks in the Windows Task List after the execution of a **firsttask** command.

nexttask intvar

intvar	The task id.

Comments

The **firsttask** and **nexttask** commands return *ids* only for those tasks that have a top-level window with a caption containing text. Since certain special tasks running in Windows may not meet these requirements, they will not be enumerated. Therefore, the number of tasks returned may be less than that indicated by the $NUMTASKS system variable. A return value of 0 indicates there are no further tasks to enumerate.

Note that the number of tasks enumerated is current with respect to the last **firsttask** command. If your script doesn't make use of this information quickly, the number of tasks could easily change!

See also

firsttask, taskname and taskpath; $NUMTASKS and $TASK in "*System Variables*."

nullstr

A Tests a string variable for the null or empty condition.

nullstr string [intvar]

string	The string to test.
intvar	If specified, will be updated to 1 if the string is null or 0 if the string is not null.

Comments

A null string is one whose first character has the value 0 (zero). **nullstr** is more commonly used as an expression for **if**/**while**/**elseif** statements.

numtostr

Converts a numeric value to a string.

numtostr number strvar [integer]

number	A float, integer or long value to convert.
strvar	The result of the conversion.
integer	If specified, the base value of the number to be converted. The base value can range from 2 up to 36, with the default base being 10. The letters '***A***' to '***Z***' are used to represent digits above 9 for bases greater than 10. The base type is not allowed for floating-point conversions.

Comments

The optional base type is not allowed for floating-point conversions. The default base is 10.

See also

ftoa, itoa, ltoa, strfmt and strtonum.

objcoord

A Returns the location coordinates of a User window object.

objcoord id intvar intvar UWUS | PIXELS

id	The control id given to the target object when it was defined in the **uwincreate** block.
intvar intvar	Will be updated to values reflecting the top left corner position of the button with respect to the top left corner of the User window or background graphic, depending on how the object was created.
UWUS	If specified, the integer values are interpreted as relative to the top left corner of the background graphic or User Window depending on how the object was created. For more information on UWUs, see "*User Window Units*" on page 48.
PIXELS	If specified, the integer values are treated as absolute user window pixel coordinates, with no relative conversions applied. The 0,0 pixel is defined as the top left corner of the User window.

Comments

The **objcoord** command returns the x and y (left and top) coordinates of the object with the corresponding id. The coordinates are relative to the User window, and are the same type of coordinates as returned by $LMOUSEX, $LMOUSEY, $RMOUSEX and $RMOUSEY.

ojbcoord returns FAILURE if the target object is currently hidden. If **objcoord** fails, no coordinate values are returned.

See also

objmove; $LMOUSEX, $LMOUSEY, $RMOUSEX and $RMOUSEY in "*System Variables*."

objhide

Hides a User window object, or a range of objects.

objhide startid [endid]

startid	The id of the target object.

endid	If specified, all objects whose ids are in the range of *startid* and *endid* will be hidden. If *startid* is greater than *endid*, a run-time error will occur.

Comments

objhide removes an object from the User window display, and prevents repaints of that object until it is shown using **objshow**. **objshow** will repaint an object immediately, even if it was not hidden. **objpaint** cannot paint a hidden object.

A User window event will not be generated if the user clicks within the region where a hidden object currently resides.

An object can be made to “blink” by creating a loop cycling **objshow** and **objhide**.

See also

objshow.

objmove

Moves a User window object identified by an id number to a new location in the User window.

objmove id left top UWUS | PIXELS [NOPAINT]

id	The control id given to the target object when it was defined in the **uwincreate** block.
left top	Integer values which determine the new top left corner position of the button with respect to the top left corner of the User window or background graphic, depending on how the object was created.
UWUS	If specified, the integer values are interpreted as relative to the top left corner of the background graphic or User Window depending on how the object was created. For more information on UWUs, see “*User Window Units*” on page 48.
PIXELS	If specified, the integer values are treated as absolute user window pixel coordinates. No relative conversions are applied. The 0,0 pixel is defined at the top left corner of the User window.
NOPAINT	An optional parameter which prevents the re-display of the moved object until an **objpaint** command is executed.

Comments

By default, the object is repainted when it’s moved. Use **objremove** to remove the object from the User window.

When NOPAINT is specified, the object will still be visible in its old location. If the User window is re-sized or repainted, any objects which were moved but not repainted may be painted in their new locations. Unless the window is entirely repainted, the objects in their old locations may still appear on the display. To prevent this, use **objhide** to hide the object being moved until a repaint of the moved object is reasonable.

See also

uwincreate, objhide, objpaint and objremove.

objpaint

Updates the User window and some or all of the objects it contains. **objpaint** should be used to update the display after any change to a User window object.

objpaint [startid][endid]

startid	The id of the window object to paint.
endid	If specified, all objects whose ids range between *startid* and *endid* will be painted. If *startid* is greater than *endid*, a run-time error will occur.

See also

uwincreate, objcoord, objpointid, objhide, objmove, objshow, and objremove.

objpointid

A Returns the object id, if any, of an object located at a specified point.

objpointid integer integer intvar UWUS | PIXELS

integer	The X-User window coordinate desired.
integer	The Y-User window coordinate desired.
intvar	Will be updated with the object id of an object located at the given coordinates, if one exists.

UWUS	If specified, the integer values are interpreted as relative to the top left corner of the background graphic or User Window depending on how the object was created. For more information on UWUs, see "*User Window Units*" on page 48.
PIXELS	If specified, the integer values are treated as absolute user window pixel coordinates, with no relative conversions applied. The 0,0 pixel is defined at the top left corner of the User window.

Comments

objpointid returns FAILURE if the specified coordinates are not contained by a User window object, or if the point is contained by a hidden object.

See also

uwincreate, objcoord, objpaint, objhide, objmove, objshow, and objremove.

objremove

Removes one or more User window objects. However, **objpaint** must be used to update the display after the objects are removed.

objremove startid [endid]

startid	The id of the window object to remove.
endid	If specified, all objects whose ids range between *startid* and *endid* will be removed. If *startid* is greater than *endid*, a run-time error will occur.

See also

uwincreate, objcoord, objpointid, objhide, objmove, objshow, objremove, objpaint, icon, iconbutton, metafile, bitmap, pushbutton, bitmapbkg, dllobject, metafilebkg, and hotspot.

objshow

Displays an object previously hidden with **objhide**.

objshow startid [endid]

startid	The id of the window object to show.

endid	If specified, all objects whose ids range between *startid* and *endid* will be shown. If *startid* is greater than *endid*, a run-time error will occur.

Comments

objshow will repaint an object immediately, even if it was not previously hidden. An object can be made to “blink” by creating a loop cycling **objshow** and **objhide**.

See also

uwincreate, objcoord, objpointid, objhide, objmove, objpaint and objremove.

oemtoansi

Converts an OEM character or string to their respective ANSI character equivalents.

oemtoansi {character intvar} | {string strvar [strlength]}

character	The character to convert.
intvar	Will contain the OEM character’s ANSI equivalent.
string	The string to be converted.
strvar	Will contain the converted string.
strlength	An optional integer, indicating the number of characters to convert in the target string. If *strlength* is not specified, the entire string will be converted.

See also

oemtokey and ansitooem; set aspect codepage in “*Set and Fetch Statements*.”

oemtokey

A   Converts an OEM character to its key value.

oemtokey character intvar

character	The character to convert.

intvar	On SUCCESS, *intvar* will contain the OEM character’s keyboard equivalent. On FAILURE, *intvar* is undefined.

See also

ansitokey, keytooem and oemtoansi; set aspect codepage in “*Set and Fetch Statements*.”

param

Defines a parameter variable in a procedure or function.

param datatype name[,name]...

datatype	The desired data type, which can be either **float**, **integer**, **long** or **string**. Statements calling the function or procedure in which the **param** appears must use the identical data type and order for the arguments that are passed to it.
name	The name of the parameter variable. It will only be valid within the function or procedure in which it appears, and it must follow the ASPECT naming conventions.

Comments

param commands must follow directly after a **proc** or **func** command. Up to 12 **param**eter variables can be defined. No other commands may appear between any two consecutive **param** commands. Multiple definitions of the same data type must be separated by commas. Initializing expressions are not allowed in parameter declarations. Arrays cannot be defined as parameters to procedures or functions. For more information, see "*Parameter Variables*" on page 29.

See also

func, proc, integer, float, long, string and call.

pastetext

Sends text in the Windows Clipboard to the active port.

pastetext

Comments

When **pastetext** is executed, the Clipboard data must be text.

See also

filetoclip, strtoclip, cliptostr and cliptofile.

pause

A Halts script execution for the specified number of seconds.

pause integer | FOREVER

integer	A positive integer value indicating the number of seconds to **pause** execution.
FOREVER	If specified, causes script execution to be **pause**d indefinitely.

Comments

Unlike the **mspause** command, **pause** can be aborted with the <Ctrl><Break> key sequence. This command can be tested with the **if** SUCCESS and **if** FAILURE statements. FAILURE is set if **pause** was terminated by the <Ctrl><Break> key sequence. Any active **when** commands will still “fire” while the **pause** command is executing.

➥ *The **pause** FOREVER command will **yield** processing time to other applications.*

See also

mspause and when elapsed.

pkmode

A Toggles error-free packet mode on and off. If **pkmode** is on, your script can receive an error-free packet from another PC running an ASPECT script using the **pksend** statement.

pkmode OFF|{ON txtimeout rxtimeout}

txtimeout	The timeout value in seconds for transmitted packets.
rxtimeout	The timeout value in seconds for received packets.

Comments

FAILURE is set if packet transfer mode was not successfully initiated. Additional SUCCESS and FAILURE information can be obtained from the $PKSEND and $PKRECV system variables. If you have activated **pkmode** and **chain** or **execute** a script, **pkmode** will remain enabled until a **pkmode off** is issued.

See also

pksend, pkrecv and when; $PKSEND and $PKRECV.

pkrecv

A Receives an error-free packet from another PC running an ASPECT script that has issued a **pksend** statement.

pkrecv intvar strvar intvar [rxtimeout]

intvar	An integer variable assigned after the packet has been successfully received. *Intvar*'s value is the byte specified in the sending PC's **pksend** statement. It is typically used to identify the contents of the packet. Valid values range from 0 to 255.
strvar	The data packet sent by the **pksend** statement after it is successfully received. The packet may include binary data.
intvar	The number of received bytes in this packet. Valid values range from 0 to 256.
rxtimeout	The amount of time that Procomm Plus will wait in packet receive mode before signaling a timeout. This value is initially set using the **pkmode** command, but you can specify this optional parameter to change the *timeout* when the **pkrecv** statement is executed.

Comments

pksend and **pkrecv** allow the script to send commands and any type of data between PCs running Procomm Plus, guaranteeing error-free transfer. A **when** $PKRECV event statement will be triggered when a change occurs as the result of the **pkrecv.** $PKRECV can also be tested to determine the status. The values for $PKRECV are:

Value	***Meaning***
0	packet not received or idle (default state)
1	packet received successfully
2	timeout occurred, packet not received
3	errors occurred, packet not received

See also

pkmode, pksend and when; $PKRECV in "*System Variables*."

pksend

Sends an error-free packet to another PC running an ASPECT script.

pksend integer string strlength [txtimeout]

integer	This value can be used as an identifier for the contents of the packet, or for any other purpose. It can be any value from 0 to 255. This value is available as an integer variable from the **pkrecv** command after the packet has been successfully received.
string	The data to be sent in the packet. Raw data may be included.
strlength	Determines the number of bytes to send in this packet.
txtimeout	An integer specifying the amount of time Procomm Plus will attempt to send the packet. This value is initially set with the **pkmode** command, but you can specify this optional parameter to change the timeout when the **pksend** statement is executed.

Comments

After a **pksend** statement is executed, Procomm Plus will attempt to send the packet three times during the timeout period. A **when** $PKSEND event statement will be triggered when a change occurs as the result of the **pksend** statement; the system variable $PKSEND can be tested to determine the status. The values for $PKSEND are:

Value	***Meaning***
0	packet in progress or idle (default state)
1	packet sent successfully
2	timeout occurred, packet not sent
3	errors occurred, packet not sent
4	packet send canceled by receiver, packet not sent

See also

pkmode, pkrecv and when; $PKSEND in "*System Variables*."

playback

Plays back a *Capture* file or stops a playback currently in progress.

playback OFF | filespec

OFF	Aborts the **playback** of a *Capture* file in progress.
filespec	The *Capture* file to be played back.

Comments

If **set aspect rxdata on** has been issued, ASPECT will "see" **playback** data as if it were incoming data from the communications port. If a path is not specified, **playback** defaults to the *Capture* file directory.

See also

set aspect rxdata in "*Set and Fetch Statements*" and $PLAYBACK in "*System Variables*."

printalign

Determines how text will be positioned when sent to the printer with the **printstr** and **printchar** commands. Choices are left-justified, right-justified or centered text.

printalign LEFT | CENTER | RIGHT

Comments

The justification stays in effect until the next **printalign** statement is executed. LEFT alignment must be set whenever the **printtabstr** command is used. If alignment is set to CENTER or RIGHT, do not attempt to change the print attribute or font except at the beginning of a new line.

See also

printer, printattr, printmargin, printstr, printchar, printtabstr, printtabs and printfont; the set print command family in "*Set and Fetch Statements*."

printattr

Sets the character attributes for text sent to the printer with the **printstr**, **printtabstr** and **printchar** commands. **printattr** can be used to format printed output with any combination of bold, italic or underlined characters.

printattr NORMAL | { [BOLD] [ITALIC] [UNDERLINE] }

NORMAL	Sets or resets the current printer font to normal character attributes.
BOLD	Sets the current printer font to bold character attributes.
ITALIC	Sets the current printer font to italic character attributes.
UNDERLINE	Sets the current printer font to underline character attributes.

Comments

The attributes stay in effect until the next **printattr** statement is executed. NORMAL removes any previously set attributes. If alignment is set to CENTER or RIGHT, do not attempt to change the print attribute or font except at the beginning of a new line.

See also

printer, printmargin, printstr, printchar, printtabstr, printtabs and printfont; the set print command family in *"Set and Fetch Statements*."

printcapture

A Routes characters received by Procomm Plus or entered by the user to the printer.

printcapture OFF | ON

Comments

printcapture sends characters to the current printer or the printer specified by the **set print device** script command.

See also

capture, snapshot and sbsave; the set print command family in *"Set and Fetch Statements*."

printchar

Prints a character on the "open" printer at the current position.

printchar character

Comments

A Tab character, ASCII 9, causes subsequent data to be printed at the next 8-character interval on the current line. A Carriage Return, ASCII 13, or Line Feed character, ASCII 10, causes the current line to be terminated and a new line to begin, while a Form Feed character, ASCII 12, causes subsequent data to print on a new page. These “special” formatting characters are ignored when printing raw data. For more information on opening a printer and defining the current character position, please refer to the **printer** command.

See also

printer, printtabstr and printstr.

printer

A Opens the current printer for any of the **print** command family.

printer { OPEN [RAW] } | CLOSE | CANCEL

OPEN	Activates the current Windows printer. You can select a printer to open with the **set print device** command. If the printer was previously opened, it will be closed and re-opened, which terminates the previous print job.
RAW	If specified, causes all data to pass through to the printer, with no Line Feed, Form Feed, Tab or print attributes. Also, all print commands such as **printattr**, **printfont**, **printalign**, **printmargin** and **printtabs** will be ignored. This mode is useful for printing data that is already in PostScript form. If the printer’s Windows driver does not support a “pass-through” mode, Procomm Plus will attempt a best-fit output mode.
CLOSE	Closes the printer previously opened within a script, terminating the current print job. If a script terminates before closing the printer, it will be closed automatically by ASPECT.
CANCEL	Ends a printing session and deletes any outstanding text to be printed.

Comments

A **printer** must first be opened before you issue any **print** statements. The current printer parameters or the values specified by **set print** commands act as defaults when the printer is first opened.

printchar and **printstr** begin printing at the current character position, which is defined as one of three different positions: the top left margin of a new page, the next character position after a previous **printchar** or **printstr** or the left margin one line below a previous **printtabstr**. The current position for any **printtabstr** is the left margin one line below the last line printed. If any print command is attempted that would print the line below the bottom margin, the new line will be printed at the top left margin on the next page.

To print formatted text from a script, simply use **printer open**, followed by any combination of these printing commands: **printalign**, **printattr**, **printfont**, **printmargin**, **printtabs**, **printtabstr**, **printchar** or **printstr**.

See also

printalign, printattr, printfont, printmargin, printtabs, printfit, printchar, printstr and printtabstr; the set print command family in "*Set and Fetch Statements*."

printfit

A Determines if a string will print on the current page.

printfit string [strlength]

string	The string to test.
strlength	The length of the test string. If *strlength* is not provided, a null-terminated string is assumed and its length is determined automatically.

Comments

In essence, **printfit** operates the same as **printstr,** except that no printing occurs. **printfit** sets the SUCCESS flag if the string can be printed on the current page. If the string would cause the printing to begin on the next page, the FAILURE flag is set. **printfit** can be used to prevent page breaks within paragraphs. **printfit** always returns SUCCESS if the printer was opened for raw data printing.

See also

printer, printstr and printtabstr; the set print command family in "*Set and Fetch Statements*."

printfont

Selects a new printer font for subsequent **printchar**, **printtabstr** or **printstr** commands.

printfont font size

font	A string specifying the name of the font to use.
size	An integer which determines the size of the font to use. Values can range from 0 to 255 points. The value 0 automatically selects a standard font size for the open printer.

Comments

The printer will use the font and size specified in a **printfont** command until another **printfont** command is issued. **printfont** has no effect if the printer was opened for raw data printing. If alignment is set to CENTER or RIGHT, do not attempt to change the print attribute or font except at the beginning of a new line.

See also

printer, printalign, printmargin, printtabs, printstr, printchar, printtabstr and printattr.

printmargin

Determines the page margins for data printed from Procomm Plus.

printmargin left [right [top [bottom]]]

left	Specifies the left margin in inches. A float value.
right	Specifies the right margin in inches. A float value.
top	Specifies the top margin in inches. A float value.
bottom	Specifies the bottom margin in inches. A float value.

Comments

The optional margins must be specified in sequence. They define a rectangle where the text will be printed. For example, a left margin setting of 5 inches would result in a column alignment down the right side of the page. If a margin is set to a value that is less than the minimum margin or greater than the available printing area, the minimum margin will be used. A negative value is ignored, and will result in no change to the particular margin value.

printmargin has no effect if the printer was opened for raw data printing.

See also

printer, printattr, printalign, printfont and printtabs; the set print command family in "*Set and Fetch Statements*."

printstr

Sends a specified string to the open printer.

printstr string [strlength]

string	The text string to send to the printer.
strlength	An optional integer specifying the amount of data to print. If *strlength* is not provided, a null-terminated string is assumed and its length is determined automatically.

Comments

printstr "streams" text onto the page using the selected font and attributes within the margins. Continued use of the **printstr** command without the use of an intervening **printmargin** will "append" each string to the previous string. An attempt to print below the bottom margin prints the remainder of the string on the next page.

If the printer was opened in raw mode, all "special" formatting is disabled. If the printer was not opened in raw mode, a Tab, ASCII 9, is expanded to fixed tab stops at eight character intervals, a Carriage Return, ASCII 13, a Line Feed, ASCII 10, or a Carriage Return/Line Feed pair force a new line, and a Form Feed, ASCII 12, forces the printer to advance to the next page. For more information on opening a printer and defining the current character position, please see the **printer** command.

See also

printer, printchar and printtabstr; the set print command family in "*Set and Fetch Statements.*"

printtabs

Determines the tab positions for the strings printed with the **printtabstr** command.

printtabs CLEAR | float [float...]

CLEAR	Clears any previously-set tabs, defaulting to 8-character tab stops.

float [float...]	A series of float values representing the desired tab stops. Values are referenced in sequence from the left margin, in units of inches. Up to 12 tab stops may be specified.

Comments

The tab values represent distances from the left margin. If more tabs occur within a string than are set by **printtabs**, the default 8-character tab stop value will be assumed. **printtabs** only sets the tabs for the **printtabstr** statement. Tabs in strings that are printed using **printstr** will still default to every eighth character position.

See also

printer, printtabstr, printalign, printmargin, printstr and printattr.

printtabstr

Prints a line of text, expanding tabs to the positions set with the **printtabs** command.

printtabstr string [strlength]

string	The text to print.
strlength	If *strlength* is not provided, a null-terminated string is assumed, and its length is determined automatically.

Comments

The line of text is printed beginning at the left margin, one line below the last line printed or, if printing on a new page, at the top margin. The text will be clipped if it extends beyond the right margin. **printtabstr** is especially useful for printing tables. The **printtabstr** is only useful when text alignment is left-justified, since tab values are measured from the left margin. If text alignment is centered or right-justified, **printtabstr** behaves exactly like **printstr**.

Other formatting characters such as Carriage Returns, ASCII 13, Line Feeds, ASCII 10, or Form Feeds, ASCII 12, are processed as in the **printstr** and **printchar** commands. All formatting is disabled if the printer has been opened in raw mode.

See also

printer, printchar, printstr and printtabs; the set print command family in "*Set and Fetch Statements*."

proc

Marks the beginning of a procedure block. Each procedure must be given a unique name, and every ASPECT script must have a procedure called "**main**" which indicates the starting point of script execution.

proc name

Comments

Any procedure except **main** may define a parameter list; the parameters are variables of any type, and are local to the procedure. Parameters are initialized with arguments passed via the **call** which invoked the procedure.

Procedures can be referenced either by using the **call** command:

call myproc [with arglist]

or by using the "shortened" form:

myproc([arglist])

Each form constitutes a procedure **call** which causes execution to transfer to the start of the procedure block. When the procedure is exited, execution returns to the point where the **call** was made.

➥ *The shortened form of procedure **call**s is highly recommended. It is shorter, just as readable, and less verbose. Also, future enhancements to ASPECT may require it.*

See also

call, func, param, return, setjmp, longjmp and endproc; and *"Procedures and Functions*."

profilerd

Reads a specified item value from any portion of a specified Windows **.ini** disk file.

profilerd filespec string string strvar | intvar

filespec	The name of the **.ini** file to reference. If a path is not specified the Windows directory ($WINPATH) will be used.
string	The name of the topic containing the desired item.

string	The item name containing the desired value.
strvar | intvar	The value returned into a string or integer variable.

Comments

If the name of the topic or item does not exist in the target **.ini** file, *strvar* will be null; *intvar* will equal -1. If the item is being stored into *intvar*, but the item does not contain numeric data, then 0 will be assigned.

See also

profilewr.

profilewr

A Writes a specified item value to any portion of a specified Windows **.ini** disk file.

profilewr filespec string string string | integer

filespec	The name of the target **.ini** file. If a path is not specified the Windows directory ($WINPATH) will be used.
string	The name of the topic containing the desired item.
string	The item name.
string | integer	The data to be written.

Comments

Any strings containing TAB characters (ASCII 0x9) used with this command will be truncated at the TAB.

See also

profilerd.

pushbutton

A Places a standard button control within an ASPECT dialog box or User window. The user can click on this control to indicate a choice. The
pushbutton command has two forms depending on where the button is displayed.

Dialog box format

pushbutton id left top width height label [OK | CANCEL] [DEFAULT]

id	An integer constant used as a control value for the button. This value is reported by **dlgevent** when the button is selected by the user.
left top	Location of the button from the top left corner of the dialog box. Values are integer constants in Dialog Box Units or DBUs. For more information on DBUs, see "*Dialog Box Units*" on page 48.
width height	Integer constants representing the size of the button in DBUs.
label	A string variable or constant containing the text displayed on the button face. To specify a keyboard shortcut key, preface the desired character with an ampersand (&) character. To display an ampersand in the label, use two ampersands. For example, the text "***&R&&D***" used as a label would display "***R&D***," with the ***R*** displayed as an underscored accelerator. **dlgupdate** can be used to update the text.
OK	The dialog box is removed when an OK button is pressed. Changes made to file-related controls will be saved to disk.
CANCEL	The dialog box is removed when a CANCEL button is pressed. Any changes made to file-related controls will not be saved.
DEFAULT	Specifies that the button is to be selected when the <Enter> key is pressed and no other button has the input focus, or when a list box, file list box, combo box, file combo box or directory list box item is double-clicked. Note that only a single button or icon button can be named as DEFAULT in a dialog box.

See also

dialogbox, enable, disable, iconbutton, dlgevent and dlgupdate.

User window format

pushbutton id left top width height label USERWIN | BACKGROUND

id	A unique integer value given to the button. The value is assigned to $OBJECT when the button has been selected by the user. It is used by the **objremove** command to remove the button.

left top	Integer values which determine the top left corner position of the button with respect to the top left corner of the User window or background graphic. Values are in User Window Units or UWUs. For more information on UWUs, see "*User Window Units*" on page 48.
width height	The size of the button in UWUs.
label	A constant or string variable containing the text displayed on the button face. To specify an accelerator for the button, preface the desired character with an ampersand (&) character. To display an ampersand in the label, use two ampersands. For example, the text "***&R&&D***" used as a label would display "***R&D***," with the ***R*** displayed as an underscored accelerator.
BACKGROUND | USERWIN	Specify USERWIN to fix the top left corner of the button in position relative to the upper left corner of the User window. Specify BACKGROUND to fix the button over the same spot on a background graphic. If the background is a metafile displayed with the SCALE option, the button both stays over the same spot and is re-sized in proportion to the new background dimensions.

Comments

A User window must exist before buttons can be placed. A **uwinpaint** command must be issued to display User window changes.

See also

uwincreate, uwinpaint, objcoord, objpointid, objpaint, objremove, hotspot, icon, iconbutton, dllobject, bitmap and metafile; $OBJECT in "*System Variables*."

putenv

A Adds or changes an environment variable definition.

putenv string

string	A string containing an environment variable and setting. For example, "VARIABLE=TRUE."

Comments

Environment variables can be used to pass information between scripts or applications. **putenv** can be tested with the **if** SUCCESS and **if** FAILURE statements to determine whether the environment modification was successful.

See also

getenv.

pwexit

Terminates the executing script file, disconnects and then exits Procomm Plus.

pwexit

See also

exit, exitwindows, halt, taskexit and winclose.

pwmode

A Places the communication window into one of six execution modes.

pwmode TERMINAL | FTP | WWW| MAIL | NEWS | TELNET | integer

TERMINAL	A keyword which places Procomm Plus into *Terminal* mode.
FTP	A keyword which places Procomm Plus into *FTP* mode.
WWW	A keyword which places Procomm Plus into *Web Browser* mode.
MAIL	A keyword which places Procomm Plus into *Internet Mail Client* mode. If there is no **mail serveraddress** specified in *Setup*, a switch to *Mail* mode will result in a dialog prompt.
NEWS	A keyword which places Procomm Plus into *News Reader* mode.If there is no **news serveraddress** specified in *Setup*, a switch to *News* mode will result in a dialog prompt.
TELNET	A keyword which places Procomm Plus into *Telnet* mode.
integer	A value from 0 (zero) to 5 which places Procomm Plus into the associated mode, listed above.

See also

menuselect; $PWMODE in “*System Variables*.”

pwtitlebar

Specifies the text to display in the Procomm Plus program title bar.

pwtitlebar {string [PERMANENT]}| RESTORE

string	The title text. A maximum of the first 64 characters of this string can be displayed.
PERMANENT	If specified, will cause the new titlebar information to remain after the script has terminated.
RESTORE	If specified, this keyword disables the PERMANENT setting, and causes Procomm Plus to redisplay the current title bar.

Comments

Procomm Plus changes the title bar based on numerous conditions and situations, such as switching modes and minimizing the application. Use the PERMANENT keyword to prevent Procomm Plus from changing the title bar. Use the RESTORE keyword to disable the permanent setting, and redisplay the current Procomm Plus title bar.

See also

$TITLEBAR and $PWTITLEBAR in “*System Variables*.”

radiobutton

Places a radio button/option button control in an ASPECT dialog box.

radiobutton id left top width height label

id	An integer constant used as the control id for the option button**.** It will be assigned to the control variable used by the option button's associated **radiogroup** statement when the option button is selected by the user.
left top width height	Integer constants specifying the position and size of the option button relative to the dialog in Dialog Box Units or DBUs. For more information on DBUs, see "*Dialog Box Units*" on page 48.
label	A string constant or variable which will be displayed with the button control. To specify a keyboard accelerator for the option button, simply include an ampersand (&) before the desired character. To display an ampersand in the label, use two ampersands. For example, the text "***&R&&D***" used as a label would display "***R&D***," with the ***R*** displayed as an underscored accelerator.

Comments

The **radiobutton** command may only appear within a **radiogroup**/**endgroup** block. The control variable used by the **radiogroup** statement can be initialized to pre-select a particular option button.

A single option button, or a range of option buttons can be **enable**d or **disable**d using their respective control ids. To **disable** an entire group, the option button group id can be referenced.

dlgupdate can be used to update an option button.

See also

radiogroup, endgroup, enable, disable, dlgupdate, dlgevent and dialogbox.

radiogroup

Marks the beginning of a group of **radiobutton** statements.

radiogroup id intvar

id	An integer constant used as a control id.
intvar	The control variable to be associated with this group.

Comments

When an option button within a group is selected, the *intvar* associated with that group is assigned the id value of the selected button. **dlgevent** reports the id of the option button group when an option button is selected.

Each **radiogroup** statement must be matched with an **endgroup** statement. **radiogroup** statements cannot be nested. A option button group, including the buttons it contains, can be **disable**d or **enable**d by specifying its control id. A single option button, or a range of option buttons can be **enable**d or **disable**d using their respective control ids.

See also

dialogbox, enable, disable, dlgupdate, endgroup and radiobutton.

rand

Returns a random value, ranging from 0 to 32767.

rand intvar [integer]

intvar	Will store the returned value.
integer	Specifies a seed value for the random number generator. The default seed value is 1 when Procomm Plus starts up.

Comments

The optional seed integer is generally specified for the initial call to **rand**. Subsequent calls should not include the same seed value, or the same value will be repeatedly returned.

rename

A Renames an existing file, reporting its results in SUCCESS and FAILURE.

rename filespec filespec

filespec	The source filename.
filespec	The destination filename.

Comments

If no path is supplied, **rename** searches the User Path for the source file. Files can be **rename**d to a different directory on the same drive, which is equivalent to moving them from one directory to another.

See also

delfile, chdir and copyfile; $USERPATH in “*System Variables*.”

return

Exits the current procedure or function and resumes processing at the statement following the procedure or function **call**.

return [expression]

expression	*Expression* is required for functions, but not allowed within procedures. If a function returns a string, then specify a string constant or variable as the *expression*. If a function returns a numeric value, specify any numeric constant, variable or expression.

Comments

A **return** statement placed in the “**main**” procedure terminates the script. The **endproc** command includes a tacit return, but a function must include a **return** *expression* statement, or an error will occur at compile-time.

Any **setjmp** commands invoked in the current procedure or function will no longer be active upon a **return** from that procedure. The same is true for any **when** commands that were executed to monitor changes in any local variables within the current procedure or function.

See also

proc, endproc, setjmp, longjmp, func and call.

rewind

A Repositions the file pointer corresponding to the specified id back to the beginning of the file. All end-of-file and error flags are cleared as the pointer is moved to the start of the file.

rewind id

See also

fseek, fopen and fclear.

rget

A Receives and stores a text string sent by a remote system.

rget strvar [strlength [integer | FOREVER]] [RAW]

strvar	The variable where the data will be stored.
strlength	An integer value ranging from 0 to 256, specifying the maximum number of characters to receive before continuing processing. If *strlength* is not included, the maximum of 256 characters is used.
integer	Determines the maximum number of seconds to wait for the string from the remote system before timing out. If not specified, the default of 30 seconds is used.
FOREVER	Forces the **rget** to wait indefinitely for the requested number of characters, or for the character value specified with **set aspect rgetchar**.
RAW	If specified, **rget** data will be retrieved unchanged from the receive data buffer, bypassing the translate table and 8th-bit stripping.

Comments

rget completes when the character defined by **set aspect rgetchar** is received, when the specified number of characters are received or when the specified/default time expires. Execution continues with the next statement in the script.

Data retrieved by the **rget** command will not be displayed in the *Terminal* window, regardless of the **set aspect rxdata** state. **rget** will not be interrupted by a **when** $RXDATA condition, since **rget** effectively "steals" data from the received data buffer.

rget can be tested with the **if** FAILURE command, which returns true when **rget** times out. **rget** will fail if the script does not have control of the communications port.

rget can be exited with the <Ctrl><Break> key sequence, returning FAILURE. To **rget** a result message from a modem, if *rgetchar* is set to its default of Carriage Return, you need to do two **rget**s: one to strip the Carriage Return or *rgetchar* sent by the modem before its message, and one to capture the modem message itself, since **rget** ends as soon as it encounters a Carriage Return or *rgetchar*.

See also

comgetc and comread; set aspect rgetchar in "*Set and Fetch Statements*."

rmdir

A Removes an empty directory using a specified path.

rmdir pathname

pathname	The directory to be removed.

Comments

If no path is supplied, **rmdir** removes a sub-directory in the User Path directory. **rmdir** reports SUCCESS if the directory was removed; FAILURE otherwise.

See also

chdir, mkdir and getdir; $USERPATH and $ASPECTPATH in "*System Variables*" and set aspect path in "*Set and Fetch Statements.*"

rstrcmp

A Compares the contents of two strings up to the specified length.

rstrcmp string string strlength

string string	The string constants or variables to be compared. They can contain raw data, or nulls.

strlength	The number of character positions to test.

Comments

rstrcmp’s results can be tested with the **if** SUCCESS or **if** FAILURE. SUCCESS indicates that the strings are equivalent for the *strlength* tested.

See also

stricmp, strcmp, strnicmp and strncmp.

run

A Executes an external program in a separate window.

run string [MINIMIZED | MAXIMIZED | HIDDEN] [intvar]

string	A string describing any executable program, optionally including a DOS path and program arguments. Arguments should be separated from the program name with spaces.
MINIMIZED	Starts the program in a minimized or iconic environment.
MAXIMIZED	Starts the program in a maximized or full-screen window.
HIDDEN	Starts the program without a visible window.
intvar	If specified, *intvar* is assigned the task id number of the program which is executed. This id can then be used in the **taskactivate** and **taskexit** commands, or other commands that use a task id.

Comments

run can be tested with the **if** SUCCESS statement. It returns false if the program could not be executed. **run** cannot be used for DOS internal commands and batch files; use the **dos** command instead. If a path is not specified, the program must reside in the User Path or in a directory included in the DOS path. ASPECT will change to the directory specified in $USERPATH before attempting to execute the **run** command.

run defaults to a normal startup environment if MINIMIZED, MAXIMIZED, or HIDDEN is not specified.

See also

taskactivate, taskexists, metakey, dos and shell.

rxflush

Clears the receive data buffer. Any characters that have been received but not processed or displayed will be lost when **rxflush** executes.

rxflush

Comments

rxflush will function only when ASPECT has access to the communications port. For example, it will not have any effect during file transfers or fax operations.

See also

txflush; $RXCOUNT and $RXDATA in "*System Variables*."

sbsave

A Empties the *Terminal window*'s *Scrollback Buffer* into a file, the Windows Clipboard, the current *Capture* file or the "open" system printer.

sbsave CLIPBOARD | CAPTURE | {FILE filespec [APPEND]} | PRINTER

CLIPBOARD	The *Scrollback Buffer* will be copied to the Windows Clipboard.
CAPTURE	The *Scrollback Buffer* will be copied into the *Capture* file.
FILE filespec	The *Scrollback Buffer* will be copied into a text file specified by *filespec*. If a path is not specified, the file will be created in the ***Capture path*** specified in *Setup*.
APPEND	Appends the contents to the end of the file specified in *filespec*. If the file does not exist, it will be created
PRINTER	The *Scrollback Buffer* will be copied to the printer.

Comments

When the copy destination is an existing disk file, the current contents of the file are erased. Similarly, existing contents are destroyed if the copy destination is the Windows Clipboard. If CAPTURE is specified, a *Capture* file must have been opened previously either by a script command or user input. If FILE is specified, the file described by *filespec* already exists, and the APPEND parameter is not given, its contents will be erased.

See also

snapshot; set terminal sbpages in "*Set and Fetch Statements*."

screentowin

Converts X/Y screen coordinates into coordinates relative to the specified window ID.

screentowin window intvar intvar

window	The window id.
intvar	Initially contains the screen's X coordinate and is updated to contain the window's X coordinate.

intvar	Initially contains the screen’s Y coordinate and is updated to contain the window’s Y coordinate.

Comment

Coordinates are converted relative to the client area of the window. The client area begins at a point inside the window frame, if any, and below the caption bar and menu bar, if any. The screen coordinates can be retrieved from the $POINTERX and $POINTERY system variables.

See also

setpointer, wincoord and wintoscreen; $POINTERX, $POINTERY, $XPIXELS, $YPIXELS, $POINTERWIN and $POINTERTASK in “*System Variables*.”

sdlgfopen

A   Displays a standard **File Open** dialog, allowing the user to search drives and paths and select files. **sdlgfopen** returns SUCCESS or FAILURE, depending on whether the user pressed the ***OK*** button.

sdlgfopen title filespec {SINGLE strvar} | {MULTIPLE filespec}

title	Specifies a title string for dialog.
filespec	A *filespec* specifying the type of files to display. This variable may contain multiple file types in the form “***.txt;*.doc;*.etc**” and will be filled with the *filespec* of the selected file when successful.
SINGLE strvar	On SUCCESS, *strvar* is filled with the *filespec* of the file selected by the user.
MULTIPLE filespec	On SUCCESS, *filespec* is filled with the name of a file which contains a list of *filespec*s selected by the user.

See also

dir, sdlgsaveas, sdlgmsgbox and sdlginput.

sdlginput

A Displays a standard Windows dialog box allowing the user to input a single line of text. Returns SUCCESS if user presses the ***OK*** button; FAILURE otherwise.

sdlginput title prompt strvar [strlength] [MASKED] [DEFAULT]

title	Specifies a title string for the dialog.
prompt	Specifies a prompt string to be displayed in the input box.
strvar	Returns the string entered by the user.
strlength	If specified, the number of characters a user is allowed to enter. If the user attempts to enter more than *strlength* characters, an error beep will sound.
MASKED	Causes input to be displayed as asterisks (*). This is convenient for input of sensitive data, such as passwords.
DEFAULT	If DEFAULT is specified, the current value of *strvar* is displayed as a default. The user need only press <Enter> to select this string.

See also

sdlgfopen, sdlgmsgbox and sdlgsaveas.

sdlgmsgbox

Displays a standard dialog box with a specified icon and button.

sdlgmsgbox title string icon button intvar [integer] [BEEP]

title	Specifies the text to be displayed as the dialog title.
string	Specifies the text to be displayed within the dialog. The text can act as a user prompt or warning.
icon	Determines the graphic icon to display in the dialog box. Values are: INFORMATION - A lowercase letter "i". EXCLAMATION - An exclamation point. QUESTION - A question mark. STOP - A stop sign. NONE - No icon displayed.

button	Determines the standard button configuration provided in the dialog box. Values are: OK - A single button labeled ***OK***, suitable for user confirmation of an information display. OKCANCEL - Two buttons, labeled ***OK*** and ***Cancel***. RETRYCANCEL - Two buttons, labeled ***Retry*** and ***Cancel***. YESNO - Two buttons, labeled ***Yes*** and ***No***. YESNOCANCEL - Three buttons labeled ***Yes***, ***No*** and ***Cancel***. ABORTRETRY - Three buttons labeled ***Abort***, ***Retry*** and ***Ignore***.
intvar	Returns the button selected by the user. Values are: 1 - ***OK*** 2 - ***Cancel*** 3 - ***Abort*** 4 - ***Retry*** 5 - ***Ignore*** 6 - ***Yes*** 7 - ***No***
integer	Specifies the default button for this dialog. Valid values can be 1, 2 or 3, corresponding to the first, second and third buttons in the order they appear in the dialog.
BEEP	Specifies that the default Windows beep sound will accompany the dialog.

See also

statmsg, sdlginput, sdlgfopen, sdlgsaveas, usermsg and errormsg.

sdlgsaveas

A Displays a standard **File Save As** dialog. **sdlgsaveas** is used to name a file or confirm a file name before it is saved. **sdlgsaveas** reports SUCCESS if the ***OK*** button was pressed.

sdlgsaveas title filespec strvar

title	Specifies a title string for the dialog.
filespec	A *filespec* specifying the type of files to display. This variable may contain multiple file types in the form “***.txt;*.doc;*.etc**” and will be filled with the *filespec* of the selected file when successful.
strvar	Returns the *filespec* chosen by the user.

Comments

sdlgsaveas tests for a valid *filespec*, and will prompt the user to change it if it's invalid. If the *filespec* already exists, **sdlgsaveas** queries the user before overwriting it.

See also

sdlgfopen, sdlgmsgbox and sdlginput.

sendfile

A Sends or "uploads" a file to a remote computer using the indicated transfer protocol, returning SUCCESS if the transfer successfully initializes.

sendfile protocol | index | string | DEFAULT [filespec [filespec]]

protocol	A keyword representing a file transfer protocol.
index	An integer value corresponding to the desired protocol.
string	A string constant or variable which represents a file transfer protocol, or a named-item version of a protocol.
DEFAULT	If DEFAULT is specified, the current default protocol will be used to upload the file.
filespec	If specified, the name of the file to be uploaded. The *filespec* may contain a path and/or a filename. If a path is not provided, the file is sought in the default upload directory specified in *Setup*.
filespec	Valid only for and required with Ind$file transfers. It specifies the name of the file to be created on the host system.

Comments

Under ASPECT, file transfers are asynchronous operations, meaning that they can run independently of script execution. A **getfile** or **sendfile** command only initiates a file transfer attempt. While the transfer is processing, ASPECT executes the next commands in the script. If the script needs to wait for the transfer to complete, a **while** loop like the one shown in the example should be used to "stall" the script. If the script executes a **while** loop to monitor the status of an on-going file transfer, you may find it useful to include the **yield** command within the loop. The **yield** command causes ASPECT to release its processing time to the system, which can improve file transfer performance.

The **when** $XFERSTATUS condition can be used to **call** a separate procedure if the value of $XFERSTATUS changes.

The Zmodem, Kermit, Ymodem, Ymodem-G, Xmodem, 1K-Xmodem, 1K-Xmodem-G, ASCII, and Raw ASCII protocols require a single *filespec* for uploads. The CIS-B+ protocol does not require any *filespec,* as it is provided by the host. The Ind$file protocol is unique in that it requires two *filespec*s: one for local access of the stored file, and one specifying the name of the file as it will be created on the mainframe.

➥ *If index, string or DEFAULT is specified, the compiler will treat the filespecs as optional. It is up to the user to provide the proper filespec information based on the protocol.*

For a list of the protocol names, index values, and **sendfile** *filespec* requirements, see "*Protocol Names and Indices*" on page 59.

See also

getfile, kermserve, yield and when $XFERSTATUS; $XFERSTATUS in "*System Variables*" and the set protocol, set aspect filexferbox and the protocol-specific set commands in "*Set and Fetch Statements*."

sendkey

Processes a keyboard shift state and a virtual key value and sends it to the focus window of the active application.

sendkey [ALT | CTRL | SHIFT | ALTSHIFT | ALTCTRLSHIFT | CTRLSHIFT] vkey [EXTENDED]

ALT ... CTRLSHIFT	The shift states of the described keys.
vkey	The virtual key value to process. For a complete list of virtual key code values, see "*Virtual Key Codes*" on page 603.
EXTENDED	Extends the range of keycode values to those used by the extended 101/102 keyboard.

Comment

If your script uses the command **set aspect keys on**, any keys received by Procomm Plus when the main window has the input focus will be processed by **sendkey**.

See also

sendkeystr, keyget, termkey, termvkey, winfocus and sendvkey; $FOCUSWIN in "*System Variables*" and set aspect keys in "*Set and Fetch Statements*."

sendkeystr

Sends a string to the active window or dialog, just as if the user had typed it.

sendkeystr string

Comment

The currently-active window receives keys representing all characters in the string. If the active window changes, however, the window that was active at the time the **sendkeystr** command was executed will still receive the keys. If the window is destroyed, any remaining keys not yet processed are discarded. In addition to sending keys to an active Windows application, **sendkeystr** can also send keystrokes to another Procomm Plus session, if it is the active window.

See also

termkey, sendkey, sendvkey, taskactivate and winfocus; $FOCUSWIN and $TASK in "*System Variables*" and set aspect keys command in "*Set and Fetch Statements*."

sendvkey

Processes an encoded key value and sends it to the focus window of the active application.

sendvkey keyval

keyval	An integer value representing both a key and the current keyboard state. For more information on *keyval*, see "*ASPECT Conventions*" on page 42. A complete list of virtual key code values is contained in "*Virtual Key Codes*" on page 603.

Comment

If your script uses the command **set aspect keys on**, any keys received by Procomm Plus when the main window has the input focus can be processed by **sendvkey**. Keystrokes cannot be sent to a DOS compatibility window.

See also

termvkey, keyget, sendkey, sendkeystr, taskactivate and winfocus; the set aspect keys command in "*Set and Fetch Statements*" and $ACTIVEWIN and $FOCUSWIN in "*System Variables*."

set

A Changes system parameters that control various Procomm Plus and ASPECT operations. **set** commands can change the settings in *Setup* and the *Connection Directory* for your applications.

set param data

param	One or more keywords identifying the parameter to be changed.
data	The new setting for this parameter. Data can be a keyword, integer, long or string, depending on the parameter requirements.

Comments

Any **set** command which expects a keyword when its value is being set will also allow an integer value to be used. The value is range-checked at run-time. This means that when a **fetch** statement is used to obtain the current setting, that same value returned by **fetch** can be used as an argument for the related **set** command. It is recommended, however, that a script use the keyword form when setting values not previously obtained from a **fetch** statement rather than using a hard-coded value.

To save changes to the current settings, use the **setup save** command. To restore the current changes, use the **setup restore** command. To change a *Connection Directory* entry's information, first use the **set dialentry access** command. To save or restore *Connection Directory* changes, use the **dialsave** and **dialload** commands.

set will return SUCCESS or FAILURE, depending on whether the **set** command could be executed. For complete information on **set** statements, refer to "*Set and Fetch Statements*."

See also

fetch.

setjmp

Marks a location within a script that you can immediately “jump” to with the **longjmp** command.

setjmp id intvar

id	An integer id value identifying the marked location for the **longjmp** command.
intvar	An integer variable, initialized to 0 at execution. This variable will contain a new value when the **longjmp** is executed, allowing further processing, depending on the script’s execution at the point of the **setjmp**.

Comments

The marked location is the command following the **setjmp**. It remains active until a return from the function where it was set, or until another routine uses a **setjmp** with the same id. Any number of **setjmp** locations can be active at once.

See also

longjmp, goto and call.

setpointer

Moves the mouse pointer to the specified X/Y screen coordinates. **setpointer** is valid only when Procomm Plus is the active application.

setpointer integer integer

integer	The X screen coordinate. The system variable $POINTERX returns this value for the current mouse pointer position.
integer	The Y screen coordinate. The system variable $POINTERY returns this value for the current mouse pointer position.

See also

wincoord; $POINTERTASK, $POINTERWIN, $XPIXELS, $YPIXELS, $POINTERX and $POINTERY in *“System Variables.”*

setup

A Manipulates the Procomm Plus *Setup* facility.

setup SAVE | RESTORE

SAVE	Saves the current Procomm Plus settings to **pw4.prm**.
RESTORE	Makes the last settings saved to **pw4.prm** the current settings.

Comments

Typically, user changes are saved to disk as soon as the user presses ***OK*** to leave *Setup*. However, changes caused by an ASPECT script or by dialing a directory entry are not saved automatically. This prevents temporary settings from affecting the start-up settings. **setup** provides the ability to undo temporary changes, or to make them permanent. **setup** RESTORE will fail if either the *Connection Directory* or *Setup* is open, if a file transfer or fax operation is in progress or if Procomm Plus is dialing a directory entry at the time **setup** executes.

See also

set.

shell

A Displays the DOS command prompt.

shell [intvar]

intvar	An integer variable which will return the task id of the DOS session.

Comments

The DOS session created by the **shell** command will run full-screen or windowed, depending on the Windows mode and the settings in the **_default.pif** file. You may run any program that can be normally run from DOS at this time. To close the DOS window, type EXIT at the DOS command line. **shell** sets SUCCESS/FAILURE depending on whether the DOS box was successfully opened. ASPECT will change to the directory specified in $USERPATH before attempting to execute the **shell** command.

See also

taskexists, metakey, dos and run.

shortpath

A Sets a string variable to contain the short path name of the filespec.

shortpath filespec strvar

filespec	A string containing a path.
strvar	A string variable to receive the short version of *filespec*.

Comments

This command resolves paths with respect to the User Path. To obtain the long name of a file from the short name, call **findfirst**. This command fails if the file cannot be found, or if the volume does not support 8.3 aliases.

See also

findfirst.

snapshot

Copies the contents of the current *Terminal* display to a disk file, the Windows Clipboard, the current *Capture* file, the system printer or the *Scrollback Buffer.*

snapshot CLIPBOARD | CAPTURE | {FILE filespec [APPEND]} | PRINTER | SBBUFFER

CLIPBOARD	The *Terminal* display contents will be written to the Windows Clipboard. Any existing Clipboard contents will be erased.
CAPTURE	The *Terminal* display contents will be written to the *Capture* file. A *Capture* file must be open before **snapshot** executes.
FILE filespec	The *Terminal* display contents will be written to the specified file. If a path is not specified, the ***Capture path*** specified in *Setup* will be used.
APPEND	Appends the contents to the end of the file specified in *filespec*. If the file does not exist, it will be created

PRINTER	The *Terminal* display contents will be printed. **snapshot**s to the printer are usually performed when print capturing is off. However, it may be useful for certain full-screen applications where data doesn't scroll off the screen automatically.
SBBUFFER	The *Terminal* display contents will be written to the *Scrollback Buffer*.

Comments

The *Terminal* display contents captured by **snapshot** may be comprised of information from the *Scrollback Buffer* or from the current logical terminal screen. **snapshot** captures the information that is visible to the user at the time the command is executed. If the file described by *filespec* already exists and the APPEND parameter is not given, its contents will be erased.

See also

sbsave and printcapture.

splitpath

Splits a file and pathname.

splitpath filespec strvar strvar strvar strvar

filespec	The source *filespec*.
strvar	Will contain the drive letter and colon, for example, "**e:**."
strvar	Will contain the path information, for example, "**\program files\procomm plus\aspect\**."
strvar	Will contain the filename, for example, "**fiddler**."
strvar	Will contain the filename extension, if any. For example, "**bmp**."

See also

getfilename, getpathname and makepath.

statclear

Clears the status line at the bottom of the *Terminal window.*

statclear

See also

statmsg and clear.

statmsg

A Displays a formatted message on the *Terminal* window status line. The message remains visible until another command clears or resets it.

statmsg string | [formatstr arglist] [beep]

string	A string of up to 256 characters.
formatstr arglist	A string of up to 256 characters containing up to 12 format specifiers followed by a list of arguments. The arguments must be listed in the order they are specified within *formatstr.*
beep	An optional parameter which plays the sound file listed in the ***Miscellaneous errors*** field *on the* ***System***, ***System Options*** pane of the **Setup** dialog. If this sound is not defined, the Windows default beep sound is played. The message text will blink for the duration of the beep.

See also

errormsg, sdlgmsgbox, statclear, termmsg and usermsg.

strcat

A Concatenates a string onto a string variable.

strcat strvar string [strlength]

strvar	The variable to which the second *string* is concatenated. The resulting string must be less than or equal to 256 characters in length; it is always null-terminated.
string	The string constant or variable concatenated to *strvar.*

strlength	An integer specifying the maximum number of characters to be copied from the second string.

See also

strcpy, strupdt and strfmt.

strchr

A Searches for the first occurrence of a character in a given string, returning SUCCESS or FAILURE depending on the outcome.

strchr string character [intvar]

string	The string constant or variable to search.
character	The desired character value, between 0 and 255 inclusive.
intvar	If specified, will contain the zero-based index within the string where the character occurs or -1 if the character is not contained within the string.

See also

strcspn, strrchr, strsearch, strfind, strcmp, stricmp, strncmp and strnicmp.

strcmp

A Performs a case-sensitive comparison on two strings, for the length of the shorter string.

strcmp string string [intvar]

string	The first string constant or variable.
string	The second string constant or variable.
intvar	If specified, *intvar* will be set to 0 if the strings are identical. It will be greater than 0 if the first string is lexicographically greater than the second and less than 0 if the first string is lexicographically less than the second.

Comments

For case-insensitive comparisons, see **stricmp** or **strnicmp**. To compare strings for a specified number of characters, see **strnicmp** or **strncmp**.

See also

stricmp, strfind, strnicmp, strncmp, rstrcmp, strupr and strlwr.

strcpy

Assigns a string to a string variable.

strcpy strvar string [strlength]

strvar	The target string variable. It will always be null-terminated.
string	The string constant or variable to be copied. It can contain “raw” data to be copied.
strlength	An integer specifying the maximum number of characters to copy from the second string.

Comments

Strings can also be assigned to each other with the assignment operator “=” For example, **s0 = s3**.

See also

strfmt, substr and strupdt.

strcspn

A Searches a string for a character contained in a second string.

strcspn string string [intvar]

string	The string to be searched.
string	A string containing the characters to be tested.
intvar	If specified, will return the position of the first character in the first string which equals a character contained in the second string.

Comments

If no characters within the second string are found in the first string, **strcspn** will return the length of the first string, and report FAILURE. **strcspn**'s results are zero-indexed. To search for characters that do not occur in a string, see the **strspn** command.

See also

strspn, strcmp, stricmp, strncmp, strnicmp and rstrcmp.

strdelete

Removes characters from the contents of a string variable.

strdelete strvar strindex strlength

strvar	The target string variable.
strindex	The index of the first character in the string to be removed.
strlength	An integer specifying the number of characters to remove, beginning at the location specified by *strindex*.

See also

strextract, strinsert, strreplace, strtok and substr.

strextract

Returns a string from a list of elements.

strextract strvar string string integer

strvar	A string variable which will receive the output characters.
string	A list of elements separated by a common series of characters.
string	The common series of characters that separate each element in the list string. This string could contain a single character such as a space, comma or tab, or several characters.
integer	A zero-based integer specifying the occurrence to return.

Comments

strextract is useful for extracting specific items from a comma- or tab-separated string. To determine the number of items in a delimited string list, use the **strsearch** command.

See also

strchr, strcspn, strfind, strsearch, strtok and substr.

strfind

A Tests for an occurrence of the specified text within a string, reporting SUCCESS or FAILURE depending on the outcome of the test.

strfind string string [intvar] [MATCHCASE]

string	The string constant or variable to be searched.
string	The string constant or variable containing the target characters.
intvar	An integer variable that if specified, will contain the zero-based index within the string where the first character of the search string occurs or -1 if the search string is not contained within the string.
MATCHCASE	By default, **strfind** is case-insensitive. Specifying MATCHCASE forces a case-sensitive search.

See also

strchr, strcspn, strsearch and strrchr.

strfmt

Creates a formatted string using a template, and modifies it with string or numeric variables. **strfmt** is similar to “sprintf” in the “C” programming language.

strfmt strvar formatstr [arglist]

strvar	Variable where the formatted string will be stored. The maximum length of the expanded string is 256 characters.
formatstr	A string constant or variable, which may contain format specifiers. For a full description of the format specifiers supported by ASPECT, see “*Formatting Text and Data*” on page 49.

arglist	An optional list of up to 12 arguments matching specifiers within the *formatstr.* Values are taken sequentially, left-to-right from the first parameter listed to the last for format types found in the *formatstr.*

See also

strcpy, strupdt and strcat.

strgetc

Returns the ASCII value of a single character within a string.

strgetc string strindex intvar

string	The source string.
strindex	The character position of the desired character.
intvar	Will contain the character value.

See also

strchr, strcspn, substr and strputc.

stricmp

A Performs a case-insensitive comparison of two strings.

stricmp string string [intvar]

string	The first string constant or variable.
string	The second string constant or variable.
intvar	If specified, *intvar* will be greater than zero if the first string is lexicographically greater than the second, 0 if the strings are equal, and less than zero if the second string is greater than the first.

Comments

stricmp performs a case-insensitive comparison for the length of the shorter string. To perform comparisons on specified string lengths, see **strncmp** or **strnicmp**.

See also

strcmp, strncmp, strnicmp and rstrcmp.

string

Defines a global or local **string** variable, or an array of **string** variables.

string name[=expression][,name[=expression]]...

name	The name of the new variable. *Name* must follow the standard ASPECT naming conventions. When defining an array, the subscript expression must result in a constant value.
=string	An initializing expression. It can be a quoted string or a previously-declared string variable, or a function that returns a string result.

Comments

Up to 256 global string variables can be defined, as well as 256 global string arrays. Local **string** definitions are limited by the available stack space at the time the procedure or function is called. See "*User-defined Variables*" on page 28, "*Global and Local Variables*" on page 28, and "*Arrays*" on page 30 for more information.

See also

param, long, float and integer.

strinsert

Inserts the contents of a string into another string.

strinsert strvar string strindex [strlength]

strvar	The target string. Its resultant length must be less than or equal to 256 characters.
string	A string constant or variable containing the characters to be inserted.
strindex	The index within the *strvar* to insert the specified characters.

strlength	If specified, the number of characters from the source string to insert into *strvar*. Otherwise, a null-terminated string is assumed and the string will be inserted up to the null character. Raw data can be inserted by specifying a *strlength*.

See also

strcpy, strdelete, strfmt and strupdt.

strlen

Returns the length of a null-terminated string into an integer variable.

strlen string intvar

intvar	Will equal the count of characters in the string, up to but not including the null character, ASCII 0.

strlwr

Converts the contents of a string variable to all lowercase characters.

strlwr strvar

strvar	The string to be converted to lowercase.

See also

strupr.

strncmp

A Performs a case-sensitive comparison of two strings for a specified length.

strncmp string string strlength [intvar]

string	The first string constant or variable.
string	The second string constant or variable.
strlength	An integer specifying the maximum number of characters to be compared.

intvar	If specified, *intvar* will be greater than zero if the first string is lexicographically greater than the first, 0 if the strings are equal, and less than zero if the second string is greater.

See also

strcmp, stricmp, strnicmp and rstrcmp.

strnicmp

A Performs a case-insensitive comparison of two strings for a specified length.

strnicmp string string strlength[intvar]

string	The first string constant or variable.
string	The second string constant or variable.
strlength	An integer specifying the maximum number of characters to be compared.
intvar	If specified, *intvar* will be greater than zero if the first string is lexicographically greater than the first, 0 if the strings are equal, and less than zero if the second string is greater.

See also

strcmp, strncmp, stricmp and rstrcmp.

strputc

Modifies the value of a single character within a string.

strputc strvar strindex character

strvar	The target string. **strputc** can be used to shorten a string variable by specifying a character value of 0 at the desired *strindex* location.
strindex	The position within the string to place the character. *strindex* is zero-based from the beginning of the string.
character	The character to put into the string.

See also

strgetc, strinsert, strupdt and strset.

strquote

Places the contents of a string variable in double quotes.

strquote strvar

See also

strfmt.

strrchr

A Searches for the last occurrence of a character in a given string, reporting SUCCESS or FAILURE depending on the outcome of the search. **strrchr** behaves identically to **strchr**, except that its search begins at the end or “right” of the target string.

strrchr string character [intvar]

string	The string constant or variable to search.
character	The desired character value between 0 and 255 inclusive.
intvar	An integer variable that if specified, will contain the zero-based index within the string where the character occurs or -1 if the character is not contained within the string.

See also

strchr, strsearch, strfind, strcmp, stricmp, strncmp, strcspn and strnicmp.

strread

Reads a block of data from a string.

strread string strindex numvar | {strvar strlength}

string	The *string to* access.
strindex	An integer value, equal to the number of bytes from the start of the *string*, specifying the location of the data to be read.

numvar	A float, integer, or long variable.
strvar strlength	A string variable, with a specified number of characters to be read.

Comments

ASPECT automatically determines the number of bytes to be read if a numeric variable is specified for **strread**. An integer or long argument will cause 4 bytes to be read from the string; a float 8 bytes. A string will read the number of bytes specified in *strlength*.

See also

strwrite, strgetc and substr.

strreplace

Searches a string variable for a specific character or pattern of characters, replacing the target character or character pattern with another string.

strreplace strvar string string [integer] [MATCHCASE]

strvar	The string variable to be searched and updated.
string	The target character or pattern of characters to be replaced.
string	The replacement text. If string is specified as empty quotations (““), the target character(s) will be deleted and not replaced.
integer	An optional value specifying the maximum number of replacements to perform. The default is all occurrences found of the target text.
MATCHCASE	If specified, forces the search to be case-sensitive.

See also

strputc, strdelete, strinsert, strsearch, strfind, strfmt and strextract.

strrev

Reverses the contents of a string variable.

strrev strvar

See also

strright, strupdt and strputc.

strright

Copies a string of characters from the end of a string. **strright** behaves identically to **substr**, except that it returns the characters from the right side of the string without having to specify a starting index.

strright strvar string strlength

strvar	The output string.
string	The input string.
strlength	The number of characters to return.

See also

strcpy, substr and strextract.

strsearch

A Searches a string for a specific character or pattern of characters.

strsearch string string [intvar] [MATCHCASE]

string	The string to be searched.
string	The target character or pattern of characters.
intvar	The number of occurrences of the target.
MATCHCASE	If specified, forces the search to be case-sensitive.

Comments

strsearch can be used to count delimiting characters within a string, making it extremely useful with **strextract** and list operations.

See also

strcspn, strchr, strrchr, strfind, strreplace and strextract.

strset

Sets the characters in a string variable to a specified ASCII value.

strset strvar strindex character strlength

strvar	The target string variable. It is not automatically null-terminated.
strindex	The starting position within the target string.
character	The value to set, from 0 to 255, inclusive.
strlength	The number of character positions to set. The starting position plus *strlength* must not exceed the allowed length of an ASPECT string variable, or an error will occur at run-time.

See also

strputc, strdelete, strupdt and strinsert.

strsltime

A Converts a date string and time string into a long variable.

strsltime string string longvar

string	The date, in the format specified by ***Short date style*** in the ***Date*** tab of the **Regional Settings** section of the Windows Control Panel.
string	The time, in the format specified by ***Time style*** in the ***Time*** tab of the **Regional Settings** section of the Windows Control Panel.
longvar	Will receive the resultant *timeval*.

Comments

This command will set FAILURE if an invalid date string and/or time string is used as an argument to the command.

See also

ltimestrs, intsltime and ltimestring; $LTIME, $DATE and $TIME in “*System Variables*.”

strspn

A Searches a string for the first character that ***does not*** occur within a second string.

strspn string string [intvar]

string	The string to be searched. If it is comprised entirely of characters from the second string, **strspn** will return the length of the first string and report FAILURE.
string	The string containing the target character(s). To search for characters in a string that ***do occur*** within another string, see the **strcspn** command.
intvar	If specified, will return the position of the first character in the first string which does not occur in the second string.

See also

strchr, strcspn, strfind, strrchr, strsearch.

strtoclip

A Copies information from a string to the Windows Clipboard.

strtoclip string | [formatstr arglist]

string	A string of up to 256 characters.
formatstr arglist	A string of up to 256 characters containing up to 12 format specifiers followed by a list of arguments. The arguments must be listed in the order they are specified within *formatstr.*

See also

cliptofile, cliptostr and filetoclip.

strtok

A Parses a delimited string into tokens.

strtok strvar strvar string [integer]

strvar	Will be assigned the token string.
strvar	The delimited source string. Its contents will be altered by **strtok**. As tokens are written to the first *strvar*, they are removed from the delimited source *strvar.*
string	A string constant or variable containing delimiters for the search. If more than one delimiter is specified in this string, **strtok** searches for the first occurrence of a character not within the delimiter set.
integer	Optional parameter with a value of 1 or greater, specifying the occurrence of a token to extract. If this argument is given, all tokens preceding the specified token index will be thrown away.

See also

strextract, strspn, strcspn and strsearch.

strtonum

Converts a string to a numeric value.

strtonum string numvar [integer]

string	The string to convert.
numvar	The result of the conversion.

integer	If specified, the base value of the number to be converted. Valid values range from 2 to 36, inclusive. If a base is not specified it will be determined by the first two characters of the string. If the first character is ‘0’ (zero) and the second character is NOT ‘x’ or ‘X’, octal notation will be assumed and the string will be converted using base 8. If the first character is a ‘0’ (zero) and the second character is an ‘x’ or ‘X’, hex notation will be assumed and the sting will be converted using base 16. If the first character is a digit from ‘1’ to ‘9’, decimal notation will be assumed and the string will be converted using base 10. If the first character is a + (plus) or - (minus) the base will be determined using the second and third characters. Bases greater than 10 use the alphabetical characters ‘A’ to ‘Z’ to represent digits greater than 9. The base specifier is not valid for floating-point conversions.

See also

atof, atoi, atol and numtostr.

strupdt

Overwrites a string with another string at a specified index. **strupdt** is similar to **strputc,** but the characters replaced in the target string are taken from a string rather than an integer.

strupdt strvar string strindex strlength

strvar	The target string which will be overwritten. If **strupdt** writes past the end of *strvar*, *strvar* will not automatically be null-terminated, unless the null character from the source string is copied.
string	The source string, which will be written to the first string. It may include “raw” character data, including nulls, ASCII 0.
strindex	The starting position in the target string where the source string will be written. *strindex* is zero-based.
strlength	The number of characters to write. *strlength* plus *strindex* must not exceed 256, the maximum length of a string, or a run-time error will occur.

See also

strputc, strfind, strinsert and strreplace.

strupr

Converts the contents of a string variable to all uppercase characters.

strupr strvar

strvar	The string to be converted to uppercase.

See also

strlwr.

strwrite

Writes a block of data to a string.

strwrite string strindex numvar | {strvar strlength}

string	The *target string*.
strindex	An integer value, equal to the number of bytes from the start of the string, specifying the starting position of the **strwrite** operation.
numvar	A float, integer, or long variable to be written.
strvar strlength	A string variable, with a specified number of characters to be written.

Comments

ASPECT automatically determines the number of bytes to be written if a numeric constant or variable is specified for **strwrite**. An integer will cause 4 bytes to be written to the string; a long will write 4 bytes, a float 8 bytes. A string writes the number of bytes specified in *strlength*.

See also

strread, strputc, and strupdt.

substr

Copies the indicated number of characters from a string, beginning at a specified position.

substr strvar string strindex strlength

strvar	The target string.
string	The source string. It can contain “raw” character data.
strindex	The character position in the source string from which copying should begin. The value is zero-indexed.
strlength	The number of characters to extract from the source string. *strlength* plus *strindex* must not exceed 256, the maximum length of a string, or a run-time error will occur.

See also

strgetc, strupdt, strcpy, strsearch and strfind.

switch

Provides multiple decision points by comparing a string, integer, or long to one or more values. **switch** requires the use of **case**, **endcase,** and **endswitch** commands. The use of the **default** command is optional.

```
switch integer | long | { string [strlength] [MATCHCASE] }
case string|integer|long
        ...
[exitswitch]
        ...
[endcase]
[default]
        ...
endcase
endswitch
```

integer | long | string	The source variable or constant to be compared with the string or value specified in each **case** statement.

strlength	If specified, determines the number of characters to test in the comparison(s).
MATCHCASE	String comparisons are case-insensitive by default. MATCHCASE causes them to be case-sensitive.
case string | integer | long	Compares the **switch** command's source argument to the "target" of the **case** command. If the **switch** argument does not match a **case** target, the processing of subsequent **case** or **default** commands continues. When a match is found between the **switch** source and the **case** target, command processing continues on the line following the **case** command until an **endcase** or **exitswitch** command is encountered. Items in the **case** statement must match the data types used in the **switch** statement. For example, you cannot compare a **string** to an **integer**.
exitswitch	Causes processing to pass unconditionally to the command on the line following the **endswitch** statement. Statements between **exitswitch** and **endswitch** will not be processed.
endcase	Concludes processing of commands following a **case** or **default** command and passes control to the command on the line following the **endswitch** command. The **endcase** command is an implied **exitswitch.** Once an **endcase** is encountered, the only commands that can follow it are another **case**, a **default** or an **endswitch**.
default	An optional statement used within the **switch** structure. If no match is found in any previous **case** statement, the commands following the **default** statement will be processed unconditionally until an **endcase** or **exitswitch** command is encountered. Processing then passes to the command on the line following the **endswitch** command. **default** has no effect on overall execution.
endswitch	Terminates a **switch** command group.

Comments

A **switch** statement, in effect, substitutes for a group of **if**/**elseif**/**else** conditions. **switch** statement groups can be nested if required. **switch** supports multiple **case** commands within a single **case...endcase** statement group. A single statement group can be executed by matching the target item with any one of several **case** commands. Since the end of a command group is always an **endcase** command, at least one **endcase** must occur whenever **case** or **default** is used.

If a **case** target is matched, the statements following it are executed until an **endcase** or **exitswitch** is encountered. If another **case** is encountered, it's skipped. Execution then falls through to the next commands following the matched **case** target. If a match is made with more than one **case** command, only the statements following the first matched **case** command will be executed. The **default** command is optional, but there can be only one occurrence of it within a **switch** statement group. If a **default** isn't used and none of the **case** commands were matched, no action takes place at all. **default** can occur anywhere within the **switch** statement. **case** statements occurring between **default** and its terminating **endcase** or **exitswitch** will be ignored.

If you're using strings with **switch**, you can control the case-sensitivity of a match with the MATCHCASE keyword. By **default**, all matching is case-insensitive.

If the **switch**ed-upon item is a variable, a copy of its value is kept at the time the **switch** command was processed. Any changes to that variable will not affect matches that are made among the associated **case** commands.

See also

case, default, endcase, endswitch and exitswitch.

taskactivate

A Activates the task with the specified task id, reporting FAILURE if the task could not be activated, or didn't exist.

taskactivate task

Comments

The **taskactivate** command will only activate top level windows that have a caption containing text.

See also

run, taskexists and taskexit; $TASK and $PWTASK in *"System Variables*."

taskexists

A Tests a task id value, setting SUCCESS and FAILURE based on whether the task still exists.

taskexists task [intvar]

task	A task id previously returned by the **run**, **dos**, **firsttask**, **nexttask** or **wintask** command.
intvar	If specified, *intvar* will be set to either 1, meaning the task exists, or 0, meaning the task does not exist.

Comments

taskexists is typically used to test for completion of a task within a DOS compatibility window to allow “pacing” of multiple asynchronous tasks.

See also

taskactivate, taskexit, firsttask and nexttask.

taskexit

Terminates a Windows task. The **taskexit** command will only work with top level windows that have a caption containing text.

taskexit task

task	The id of the task to be terminated.

See also

taskactivate, taskexists, taskname, taskpath and exitwindows; $TASK and $PWTASK in "*System Variables*."

taskname

A Returns the executable name associated with a specified task id.

taskname task strvar

task	The id of the desired task.
strvar	Will contain the associated executable name.

See also

run, taskpath, firsttask and nexttask; $TASK in "*System Variables*."

taskpath

A Returns the executable directory used by a specified task id.

taskpath task strvar

task	The id of the desired task.
strvar	Will contain the path to the executable directory used by the task id.

See also

taskexists, taskname, firsttask, nexttask and run; $TASK in "*System Variables*."

taskwin

A Returns the main or top-level window id associated with the specified task id, setting FAILURE if the task does not exist, or has no top-level window.

taskwin task intvar

task	The desired task id.
intvar	Will contain the window id associated with the specified task.

Comments

The **taskwin** command will only work top-level windows that have a caption containing text. In addition, some tasks may have more than one top-level window and only the first window id associated with the task will be returned.

See also

run and wintask; $TASK in "*System Variables*."

termgetc

Returns a character value from the *Terminal* window.

termgetc row column intvar

row column	Integer values specifying the desired terminal location.
intvar	Will contain the character value.

Comments

termgetc does not alter the current cursor location on the *Terminal* display.

termgetc has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termgetc** is executed while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, the **termgetc** command is executed and the script resumes normally.

See also

termgets, termwritec, termreadc, termputc and locate; $ROW and $COL in "*System Variables*."

termgets

Returns a string from the *Terminal* window.

termgets row column strvar [strlength]

row column	Integer values specifying a terminal location.
strvar	Will contain the ASCII string.
strlength	The number of characters to read from the *Terminal* window. If *strlength* is not specified, **termgets** will read to the end of the current line. If *strlength* is specified and is greater than the number of characters remaining on the current line, the process will “wrap” to the next line and continue reading characters up to the specified *strlength*.

Comments

termgets has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termgets** is executed while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, **termgets** executes and the script resumes normally.

See also

termgetc, termputs, termreads and locate; $ROW and $COL in “*System Variables*.”

termkey

Processes a virtual key value with the keyboard shift state and sends it to the *Terminal* window, just as if the key had been pressed.

termkey [ALT | CTRL | SHIFT | ALTSHIFT | ALTCTRLSHIFT | CTRLSHIFT] vkey [EXTENDED]

ALT ... CTRLSHIFT	Keyboard shift state values.
vkey	A key-character value. For a complete list of virtual key code values, see “*Virtual Key Codes*” on page 603.
EXTENDED	Extends the range of keycode values to those used by the extended 101/102 keyboard.

Comment

The **termkey** command only sends keys to the *Terminal* or *Telnet* communication windows.

If your script uses the command **set aspect keys on**, the **termkey** command will not allow the script to receive the key value sent.

If the key corresponds to a function, that function is performed. If a menu pop-up is displayed, it will respond to keys sent by **termkey**. Otherwise, if the key corresponds to a character, it's sent out the active port.

See also

termvkey, sendkey, keyget and menuselect; set aspect keys in "*Set and Fetch Statements*."

termmsg

Writes a formatted string to the *Terminal* window at the current cursor location, updating that location accordingly. **termmsg** does not process caret ("^") sequences.

termmsg string | [formatstr arglist] [RAW]

string	A string of up to 256 characters.
formatstr arglist	A string containing text and/or format specifiers, followed by an optional list of up to 12 arguments. The arguments must be listed in the order as the specifiers within *formatstr.*
RAW	Writes the resultant string to the *Terminal* window without any processing of its contents.

Comments

termmsg has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termmsg** is executed while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, **termmsg** executes and the script resumes normally.

See also

strfmt, termputs, termwrites and locate.

termputc

Writes a single character to a specified location in the *Terminal* window without changing the current cursor location.

termputc row column character [RAW]

row column	Integer values specifying a terminal location.
character	An integer value from 0 to 255, inclusive.
RAW	Writes the character to the *Terminal* window without any processing of its value.

Comments

termputc has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termputc** is executed while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, **termputc** executes and the script resumes normally.

See also

termmsg, termputs, termgetc, termreadc and locate; $ROW and $COL in "*System Variables*."

termputs

Writes a string to a specified location in the *Terminal* window without updating the current cursor location.

termputs row column string [RAW]

row column	Integer values specifying a terminal location.
string	The string to be written.
RAW	Writes the string to the *Terminal* window without any processing of its contents. If RAW is not specified, **termputs** translates caret sequences into their appropriate ASCII values.

Comments

termputs has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termputs** is executed while text is being marked in the *Terminal* window, however, script execution is suspended. After the marking operation is completed or canceled, the **termputs** command is executed and the script resumes normally.

See also

termmsg, termputc and termwrites; $ROW and $COL in *"System Variables."*

termreadc

Reads a character value from the terminal window at the current cursor location, updating the location accordingly.

termreadc intvar

intvar	The character value.

Comments

termreadc has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termreadc** is executed while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, **termreadc** executes and the script resumes normally.

See also

termwritec and termreads; $ROW and $COL in *"System Variables."*

termreads

Reads a string from the *Terminal* window, starting at the current cursor location. **termreads** updates the location accordingly.

termreads strvar [strlength]

strvar	Receives the data read from the *Terminal* window.
strlength	The number of characters to read. If string length is not specified, **termreads** will read to the end of the current line. If *strlength* is specified and is greater than the number of characters remaining on the current line, the process will "wrap" to the next line and continue reading characters up to the specified *strlength*.

Comments

termreads has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termreads** executes while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, **termreads** executes and the script resumes normally.

See also

termwrites and termreadc; $ROW and $COL in "*System Variables*."

termreset

Clears the *Terminal* window, moves the cursor to row 0, column 0, and re-initializes the emulation.

termreset

Comments

If **termreset** is executed while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, **termreset** executes and the script resumes normally.

See also

clear and locate; set terminal command family in "*Set and Fetch Statements*."

termvkey

Processes an encoded key value and acts upon it as if it were typed in the *Terminal* window.

termvkey keyval

keyval	An integer value representing both a key and the current keyboard state. For more information on key values, see "*keyval*" on page 44.

Comment

The **termvkey** command only sends keys to the *Terminal* or *Telnet* communication windows.

If your script uses the command **set aspect keys on**, the **termvkey** command will not allow the script to receive the key value sent.

If the key corresponds to a function, that function is performed. If a menu pop-up is displayed, it will respond to keys sent by **termvkey**. Otherwise, if the key corresponds to a character, it's sent out the active port.

A complete list of virtual key code values appears in "*Virtual Key Codes*" on page 603.

See also

termkey, sendkey, keyget and menuselect; set aspect keys in "*Set and Fetch Statements*."

termwritec

Writes a single character to the *Terminal* window at the current cursor location, updating the location accordingly.

termwritec character [RAW]

character	The character value to write.
RAW	Writes the character to the *Terminal* window without any processing of its value.

Comments

termwritec has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termwritec** executes while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, **termwritec** executes and the script resumes normally.

See also

termputc, termwrites, termmsg and termreadc.

termwrites

Writes a string to the *Terminal* window at the current cursor location, updating the location accordingly.

termwrites string [RAW]

string	The string to write.
RAW	Writes the string to the *Terminal* window without any processing of its contents. If RAW is not specified, **termwrites** will process caret sequences.

Comments

termwrites has no effect while the Procomm Plus *Terminal* is in *Scrollback* mode. Script execution merely continues to the next command. If **termwrites** executes while text is being marked in the *Terminal* window, script execution is suspended. After the marking operation is completed or canceled, **termwrites** executes and the script resumes normally.

text

Displays a text string in an ASPECT dialog.

text id left top width height label LEFT | RIGHT | CENTER

id	An integer constant used as the control id.
left top width height	Integer constants specifying the size and position of the text box in Dialog Box Units or DBUs. For more information on DBUs, see "*Dialog Box Units*" on page 48.
label	The string constant or string variable to be displayed. If a string variable is specified for the label, it can be dynamically changed and then updated on-screen with **dlgupdate**. If the text control is to be used as a label for another control, it should be placed immediately preceding that control. You can then specify a keyboard accelerator for that control simply by including an ampersand (&) in the label in front of the desired character. To display an ampersand in the label, use two ampersands. For example, the text "***&R&&D***" used as a label would display "***R&D***," with the ***R*** displayed as an underscored accelerator.
LEFT	Align the text with the left margin of the text box.
CENTER	Align the text in the center of the text box.
RIGHT	Align the text with the right margin of the text box.

See also

dlgupdate, dialogbox and ftext.

transmit

A Sends a character string to the active port. If local echo is on, **transmit** echoes the string to the *Terminal* display.

transmit string [RAW]

string	May contain non-printing characters. For more information, see "*Escape Sequences*" on page 25.
RAW	If specified, causes the characters to be sent to the port without translation of any kind.

Comments

By default, the **transmit** command performs caret translation and outgoing translate table conversions on transmitted data. For an explanation of caret (^) translation, see "*Caret Translation*" on page 49. If the ***Pause character*** is contained within the *string*, it will be translated into a time delay and not transmitted. Data will be transmitted at a rate determined by the current baudrate and transmit pacing values. For fastest possible throughput at the current baudrate, make sure that transmit pacing is set to 0. **transmit** reports FAILURE if the port is unavailable or if the transmit data buffer can't accommodate the length of the string.

See also

comwrite and computc; $TXDATA in "*System Variables*."

txflush

Clears the transmit data buffer. Any un-transmitted characters remaining in the buffer are discarded.

txflush

Comments

txflush is used to clear the transmit data buffer in preparation for some task. **txflush** will not clear the buffer if ASPECT does not have access to the port, such as during file transfers or fax operations.

See also

rxflush; $TXCOUNT and $TXDATA in "*System Variables*."

#undef

Removes the current definition of a previously-defined macro.

#undef name

Comments

#undef is often paired with a **#define** command to create a “region” in a script where an identifier has a special meaning. **#undef** must also be used prior to redefining a macro that was previously **#define**d.

See also

#define and #ifdef.

usermsg

Displays a message string in a standard Windows message box with an information icon. The dialog remains until the <Esc> or <Enter> key is pressed, or a button is selected.

usermsg string | [formatstr arglist]

string	A string of up to 256 characters.
formatstr arglist	A string of up to 256 characters containing up to 12 format specifiers followed by a list of arguments. The arguments must be listed in the order they are specified within *formatstr.*

See also

errormsg, statmsg, sdlgmsgbox and termmsg.

uwincreate

Defines a User window which occupies all or part of the *Terminal window.* A User window displays objects such as **pushbutton**s and **bitmaps** on the *Terminal window*. These objects can be moved, hidden, shown, refreshed, and removed using the appropriate script commands.

➥ *While User windows can be created and manipulated in any mode, User widows and all associated controls are only displayed in Terminal and Telnet modes.*

uwincreate TOP | BOTTOM | LEFT | RIGHT | FULL SCREEN | WINDOW | PIXELS integer R G B BITMAP | METAFILE

TOP | ... | FULL	Specifies the position of the User window as it corresponds to the *Terminal* window. The remainder of the *Terminal* window displays normally. If the smaller *Terminal* window is not large enough to display all of the text, vertical and horizontal scroll bars are added automatically. If FULL is specified, the *Terminal* window area is completely filled by the User window, and no text will be displayed.
SCREEN |	SCREEN displays the User window at a “fixed” size based on a percentage of the screen size. The percentage is specified with the integer parameter.
WINDOW |	WINDOW displays the User window in proportion to a percentage of the *Terminal* window. The percentage is specified with the integer parameter.
PIXELS	PIXELS specifies the actual size of the User window in pixels, specified with the integer parameter. The User window will have different physical sizes under different screen resolutions.
integer	For SCREEN or WINDOW, the size of the User window specified as a percentage of the physical screen or the Terminal window; if the location is set to FULL, this value defaults to 100 percent. For PIXELS, the number of pixels used relative to the window placement. If the window is located at the TOP or BOTTOM, this figure is relative to the number of pixels on the vertical axis; if the window is positioned at the LEFT or RIGHT, it’s relative to the horizontal axis.
R	Defines the level of red for the User window background. An integer value ranging from 0, for no red to 255 for maximum red.
G	Defines the level of green for the User window background. An integer value ranging from 0, for no green to 255 for maximum green.
B	Defines the level of blue for the User window background. An integer value ranging from 0, for no blue to 255 for maximum blue.

	For example, to set a white background for the User window, use the RGB values 255, 255, 255. A black background is set with 0, 0, 0.
BITMAP | METAFILE	Affects only those objects placed with the BACKGROUND option in a User window without a bitmap or metafile background. BITMAP forces an object placed in the User window to act as if placed on a bitmap background. The object's position is fixed. METAFILE forces an object placed in the User window to act as if placed on a metafile background. The object's position is maintained in proportion to the User window size.

Comments

Because of the complex interaction between the User window, background graphics, and objects, we strongly recommend that you use the *User Window Editor* to create these graphic displays. The *User Window Editor* generates these statements and calculates the position and size information for each object automatically!

The following objects can be placed in a User Window:

bitmap	Displays a bitmap graphic.
dllobject	Defines an area within the User window which will be updated by a DLL module.
hotspot	Defines an area within the User window that can be selected with a mouse.
icon	Displays an icon graphic.
iconbutton	Displays an icon on a clickable button.
metafile	Displays a Windows metafile graphic.
pushbutton	Displays a button with a text label.
bitmapbkg	Displays a Windows bitmap as a background for the User window. A **hotspot** can be used to make an area of the background reactive to a mouse click.
metafilebkg	Displays a Windows metafile as a background for the User window. A **hotspot** can be used to make an area of the background reactive to a mouse click.

When objects are created in the User window, they are placed and sized based on User Window Units or UWUs. For more information on UWUs, see “*User Window Units*” on page 48.

User window object ids may have any integer value greater than 0.

➥ *There is a limit of 32 active objects for each object type and only one DLL object file can be active at a time.*

The **uwinpaint** command must be used to display the User Window after it is defined. It can also be used to update the User window if any changes are made to it. To refresh one or more objects within the window, use the **objpaint** command with the id(s) of the object(s) you wish to update.

A User window responds to both left and right mouse button events that occur within its boundaries. When a left mouse button is pressed, the $LMOUSEEVENT system variable is set to 1, and the $LMOUSEX and $LMOUSEY system variables are set to the User window coordinates corresponding to the current mouse pointer location. When the mouse button is released, the $LMOUSEX and $LMOUSEY coordinates are updated again, possibly with different user window coordinates if the mouse was moved while the button was held down. The $LMOUSESTATE system variable indicates the current state of the left mouse button, up or down. The $LMOUSEEVENT, however, is only generated for the mouse button down event. The right mouse button has a corresponding set of system variables which are updated in the same manner.

When the left mouse button is clicked upon a **bitmap**, **hotspot**, **pushbutton**, or **iconbutton**, a User window event is generated in the $OBJECT system variable, its contents being set to the id value of the clicked-upon object. The $OBJECT event is not generated until the mouse button is released.

When a right mouse button is clicked and held down while the mouse pointer is not currently over any User window object other than a **hotspot**, the User window will reveal the locations of all **hotspot** objects by inverting their locations on the screen. Hidden **hotspot**s will not be revealed by this action.

No User window event will be generated if the user clicks over a hidden User window object.

A User window will remain available if you **chain** or **execute** another script without issuing a **uwinremove**.

➥ *A User window is different from both the Terminal window and a Windows dialog box. A User window can co-exist with any portion of the Terminal window or may entirely replace it depending on the Terminal window size setting.*

See also

bitmap, metafile, pushbutton, bitmapbkg, metafilebkg, icon, iconbutton, hotspot, dllobject, objhide, objpaint, objpointid, objremove, objshow, dllobjfile, dllobjupdt, uwinpaint, uwinremove, wintouwu and uwutowin; $LMOUSESTATE, $LMOUSEX, $LMOUSEY, $LMOUSEEVENT, $RMOUSESTATE, $RMOUSEX, $RMOUSEY, $RMOUSEEVENT, $OBJECT and $USERWIN in “*System Variables*.”

uwinpaint

Paints the entire User window, and all of its objects. **uwinpaint** must be called to initially display the User window after it is created.

uwinpaint

See also

uwincreate, objremove, objmove, objshow, objhide, objpointid and objcoord; $USERWIN in “*System Variables*.”

uwinremove

Removes the User window background, or the entire User window. **unwinremove** does not affect User window objects.

uwinremove [BACKGROUND]

See also

uwincreate, objremove, objmove, objshow, objhide, objpointid and objcoord; $USERWIN in “*System Variables*.”

uwutowin

Converts X/Y User Window Units, which are measured in pixels, to window coordinates.

uwutowin intvar intvar USERWIN | BACKGROUND

intvar	The X coordinate value to convert. For more information, see “*User Window Units*” on page 48.
intvar	The Y coordinate value to convert.

USERWIN	Convert the X/Y values relative to the upper left corner of the User window.
BACKGROUND	Convert the X/Y values relative to the background graphic. If X and Y are both zero, **uwutowin** *intvar intvar* BACKGROUND will report the background graphic's actual User window coordinates.

See also

uwincreate, when mouseevent, wintouwu and wintoscreen; $LMOUSEX, $LMOUSEY and $USERWIN in "*System Variables*."

waitfor

A Pauses processing until a target string is received from the active port. **waitfor** reports SUCCESS if the target string was received. It reports FAILURE if it timed out, or if it was aborted with the <Ctrl><Break> key sequence.

waitfor string [integer | FOREVER] [MATCHCASE] [RAW] [STRIP]

string	A string of text and/or numeric characters up to 256 characters in length. An exact character match is required, but the match is not case-sensitive unless you specify MATCHCASE. Caret sequences in the string will be translated to control characters unless RAW is specified. For more information about caret (^) translation, see "*Caret Translation*" on page 49.
integer	An integer value, specifying the number of seconds to wait for the target string before timing out and continuing execution. If neither *integer* nor FOREVER is specified, **waitfor**'s default timeout period is 30 seconds.
FOREVER	If FOREVER is specified, Procomm Plus will wait indefinitely for the target string. It will not time out.
MATCHCASE	If specified, forces a case-sensitive match of the specified string.
RAW	If specified, incoming data will be retrieved unchanged from the receive data buffer, bypassing caret translation, the *Translate Table* and 8th-bit stripping if it is enabled for the current emulation.
STRIP	If specified, causes the target string to be removed from the incoming data stream. The stripped target is thus prevented from being processed by the *Terminal* window or ASPECT.

Comments

Any active **when** commands will continue to “fire” during the execution of **waitfor**.

See also

waitquiet and when target.

waitquiet

A Pauses processing until the receive data line has been inactive for a number of seconds. **wait-quiet** reports FAILURE if it was exited with the <Ctrl><Break> key sequence, or if it timed out before satisfying the inactive period.

waitquiet [integer [integer | FOREVER]]

integer	The number of seconds that the line must be inactive. If not specified, the default of 15 seconds is used.
integer | FOREVER	The number of seconds that Procomm Plus will wait to satisfy for the inactive period. If neither *integer* nor FOREVER is specified, the default of 30 seconds is used. FOREVER forces the script to pause indefinitely until the line is quiet for the required number of seconds.

Comments

If none of the optional parameters are used, **waitquiet** pauses execution of the script for up to 30 seconds while waiting for the receive data line to be inactive for 15 seconds.

Any active **when** commands will continue to “fire” during the execution of the **waitquiet** command.

See also

waitfor and when quiet; $RXDATA and $RXCOUNT in “*System Variables*.”

waituntil

A Pauses processing until the specified date and/or time.

waituntil { string [string] } | timeval

string [string]	A time string and optional date string, specifying when script processing should continue. The date and time strings must conform to the ***Short date style*** and ***Time style*** formats specified in the Windows Control Pane’s **Regional Settings** section. If the date string is not specified, the default is the current date.
timeval	A long time value that specifies both time and date.

Comments

This command will set FAILURE if an invalid date string and/or time string is used as an argument to the command. Active **when** commands will continue to “fire” during the execution of the **waituntil** command.

See also

when elapsed and pause.

weekdaystr

Returns the day of the week.

weekdaystr integer strvar [SHORT]

integer	A value from 1 to 7, corresponding to Sunday through Saturday.
strvar	Will contain the day of the week. If SHORT is specified, the day will be abbreviated.

See also

monthstr and ltimemisc.

when

Forces a **proc**edure or **func**tion **call** when a particular event occurs. Once a **when** command is executed, it remains in effect until a **when clear** command is encountered, or the script terminates.

Active **when** commands are reset to inactive when another script is chained or spawned. If control reverts back to the calling script, however, any previously-set **when** commands are once again activated.

A **call**ed procedure that processes a particular event will finish completely before it is called again, preventing recursive event handling. However, a **when** event of a different type can interrupt another **when** procedure that’s currently executing. The **set aspect whensuspend** can be used to prevent a **when**’s firing on top of other **when**s that have not yet returned from their **call**ed procedures.

The **when** command is perhaps the most versatile of the ASPECT commands, and the number of **when**s allowed is virtually unlimited. The **when** has three general forms:

when *event Form*

when event {call name} [SUSPEND]

Where event is:

DIALOG id	**call** the procedure **when** a dialog event has occurred in the dialog identified by the id value. Generally, this would be used to allow the **call**ed procedure to handle the dialog event(s). **dlgevent** must be used to clear the targeted dialog's event queue, or the **when** will fire repeatedly.
ISKEY id keyval	**call** the procedure **when** the specified *keyval* has been pressed. *Id* is any unique integer index value. The *keyval* argument should only be specified when defining a new event.
STRIP	STRIP can be specified to prevent the targeted *keyval* from being passed to the *Terminal* window or script. STRIP should only be specified when defining a new ISKEY event.
ELAPSED id duration	**call** procedure when the specified period of time has elapsed since the **when** command was initialized or last triggered. *id* is a unique integer index value. *Duration* is a positive integer value, expressed in the number of seconds, and is only required when defining an active **when**. That is, when a **call** statement appears in the same **when** command line.
QUIET duration	**call** procedure when no characters have been received for the specified period of time. *Duration* is a positive integer value, expressed in the number of seconds, and is only required when defining an active **when.** That is, when a **call** statement appears in the same **when** command line. ASPECT can only “see” incoming data that was destined for the Terminal display or for ASPECT. It cannot “see” data that is received during a file transfer or fax operation.
TARGET id string	**call** the procedure when characters matching the specified string are received. *Id* is any unique integer index value. *String* should only be specified when defining a new **when.** *String* can contain “raw” data.
MATCHCASE	MATCHCASE can be specified for a case-sensitive comparison of the target string to received data. It should only be specified when defining a new **when** TARGET event.

RAW	RAW can be specified to disable caret translation on the target string, and to disable translate table conversion and 8th-bit stripping for the current emulation on received data. It should only be specified when defining a new **when** TARGET event.
STRIP	If specified, causes the target string to be removed from the incoming data stream. The stripped target is thus prevented from being processed by the terminal or ASPECT. It should only be specified when defining a new **when** TARGET event.
USEREXIT	**call** procedure when the user presses the ***Script halt*** button on the *Action Bar* or selects ***Script | Stop Script*** command. This allows the script to branch to a “cleanup” procedure before it terminates. If the script issues the statement **set aspect control on**, the USEREXIT procedure will not be **call**ed. The **call**ed procedure must issue an **exit** or **halt** command to terminate the script. Otherwise the script will not terminate.
datavar	**call** procedure when the target *datavar* has changed. If a local *datavar* is specified, the **when** will be automatically cleared when the procedure or function returns.
sysvar	**call** procedure when the target *sysvar* has changed. **when** can be used for any variable, but it is perhaps most useful for system variables such as $ASPMENU, $CARRIER, $CNCTMSG, $DDEADVISE, $KEYHIT, $LMOUSEEVENT, $OBJECT, $RMOUSEEVENT, $RXDATA and $XFERSTATUS. If the specified system variable is one that automatically resets its value after it is accessed, the use of its name in a **when** command does not constitute an “access.” A direct assignment or other reference must be made to clear the value and to prevent the **when** from firing repeatedly. Certain system variables have special properties in **when** commands, due to the unique needs of Windows and serial communications. Please refer to “*System Variables*” on page 427 for more information, and for other system variables that may be useful in when conditions.
CALL name	The name of the procedure to **call** when the appropriate event has occurred, or the specified variable has changed. The **call**ed procedure must not require any parameters.

SUSPEND	If specified, response to other **when** conditions will be suppressed until the **call**ed procedure or function returns. SUSPENDing **when** conditions does not eliminate ASPECT's response to those conditions. Instead, the SUSPENDed **when**s are saved in a queue until the SUSPENDing **when** command has finished executing. This may result in unexpected **when** firings if the script is not configured to account for **when** events that occur during the SUSPENDed period. This is true of **when** ELAPSED, **when** ISKEY, **when** QUIET, **when** TARGET and the **when** USEREXIT commands. It is not true of **when** variable commands, meaning that changes to a variable which occur during a SUSPENDed period will not cause a **when** variable to fire.

Comments

In general, **when** commands using the system variables "fire" when those variables change values from their default or inactive state to another state. The procedure handling the **when** event should access the system variable to reset its value. Otherwise it will be automatically cleared when the procedure returns and the script will lose the value. Refer to "*System Variables*" on page 427 for more information.

Specifying a **when** index already in use by another **when** of the same type will cause the original **when** to be removed.

***when** event Control Form*

Any number of **when** DIALOG/ISKEY/TARGET commands can be active simultaneously. Each condition can be individually or globally enabled or disabled using the **when** [command] CLEAR/SUSPEND/RESUME commands, as explained below:

when event CLEAR | {RESUME [RESET]} | SUSPEND

Where event is:

DIALOG id	The **when** condition associated with the specified dialog id will be affected.
ELAPSED ID	The **when** ELAPSED condition associated with the specified id will be affected.
ISKEY id	The **when** condition associated with the specified ISKEY id will be affected.

QUIET	The **when** QUIET condition will be affected.
TARGET id	The **when** condition associated with the specified TARGET id will be affected.
USEREXIT	The **when** USEREXIT condition will be affected.
datavar	The **when** *datavar* condition will be affected.
sysvar	The **when** *sysvar* condition will be affected.

And the actions are:

CLEAR	Remove the specified **when** condition(s).
RESUME	Enable the specified **when** condition(s) which had previously been disabled with SUSPEND. If **when**s have been globally suspended, the specified **when** will remain disabled.
RESET	Clears any of the specified pending **when event**s from firing when it is RESUMEd.
SUSPEND	Disable the specified **when** condition(s). SUSPENDing **when** conditions does not eliminate ASPECT's response to those conditions. Instead, the SUSPENDed **when**s are saved in a queue until the SUSPENDing **when** command has finished executing. This may result in unexpected **when** firings if the script is not configured to account for **when** events that occur during the SUSPENDed period. This is true of **when** ELAPSED, **when** ISKEY, **when** QUIET, **when** TARGET and the **when** USEREXIT commands. It is not true of **when** variable commands, meaning that changes to a variable which occur during a SUSPENDed period will not cause a **when** variable to fire.

when *Global Control Form*

when conditions can also be globally controlled:

when CLEAR | {RESUME [RESET]} | SUSPEND

CLEAR	CLEAR all **when** conditions.
RESUME	RESUME all **when** conditions. Those **when**s which had not been individually SUSPENDed will now be allowed to fire.

RESET	Clears all pending **when events** from firing when they are RESUMEd.
SUSPEND	SUSPEND all **when** conditions. SUSPENDing **when** conditions does not eliminate ASPECT's response to those conditions. Instead, the SUSPENDed **when**s are saved in a queue until the SUSPENDing **when** command has finished executing. This may result in unexpected **when** firings if the script is not configured to account for **when** events that occur during the SUSPENDed period. This is true of **when** ELAPSED, **when** ISKEY, **when** QUIET, **when** TARGET and the **when** USEREXIT commands. It is not true of **when** variable commands, meaning that changes to a variable which occur during a SUSPENDed period will not cause a **when** variable to fire.

See also

waitfor, pause, call, func, proc and waitquiet; set aspect whensuspend in "*Set and Fetch Statements*" and all of "*System Variables*."

while

Repeats a series of commands until the test condition becomes false.

while condition

condition	The *conditions* allowed for the **while** command are identical to those of the **if** command. For more information, see the **if** command on page 214. To create an infinite loop within a **while** construct, specify a non-zero constant for the *condition* as in: **while 1**

Comments

Under Windows NT 4.0, the following loop can monopolize CPU utilization when, in fact, the script needs to perform very little processing:

```
while 1                    ;while TRUE would also work
.
.
```

.

endwhile

To prevent this wasteful CPU utilization, place a **yield** command at the end of the **while** loop.

See also

exitwhile, loopwhile, endwhile, yield and if.

winactivate

A Activates the window with the specified title or window id, reporting FAILURE if the specified window could not be activated. **winactivate** is used primarily with the **sendkey** command to ensure that the keystrokes are sent to the correct window.

winactivate window|string

window	The id of the window to activate.
string	The current title text of the window to be activated. Note that the title bar may change depending on the application, so you must ensure that the current title text matches this string. The string is not case-sensitive.

Comments

The **winactivate** command cannot be used on a hidden window.

See also

taskwin, winclose, winexists, winfocus, winstate, wintask, taskactivate and wintext; $ACTIVEWIN, $FOCUSWIN and $MAINWIN in "*System Variables*."

winclose

Sends a "close" message to a window, just as if the user had pressed <Alt><F4>.

winclose window

window	The window id.

See also

winactivate; $ACTIVEWIN, $FOCUSWIN and $MAINWIN in "*System Variables*."

wincoord

Returns the location and dimensions of the specified window in screen coordinates. **wincoord** can be used with **winmove** and **winsize** to control the location and size of the target window.

wincoord window left top [width [height]]

window	The target window id.
left top width height	Integer variables which will receive the requested values.

See also

screentowin, winmove and winsize; $XPIXELS and $YPIXELS in *"System Variables*."

winenabled

A Tests the target window to determine its availability to the user or to the script. **winenabled** can be used to determine whether a script-generated dialog box has been disabled.

winenabled window [intvar]

window	The target window id.
intvar	Will be set to 1 if the window is enabled or 0 if the window is disabled.

See also

enable, disable, winvisible, winhide, winmaximize, winminimize, winshow and winstate.

winexists

A Tests for the existence of a window. **winexists** can be used to test whether an application is still active.

winexists window [intvar]

window	The id of the target window.
intvar	If specified, *intvar* will be 0 if the window does not exist or 1 if the window does exist.

See also

taskexists, taskwin, winactivate, winfocus, winstate and wintask.

winfocus

A Specifies the window to receive keyboard input.

winfocus window

window	The id of the window that is to receive input. This value can be obtained by checking the $FOCUSWIN variable when the desired window is active. Alternately, you may use the title text of any window that has a title bar. This can be an application or any dialog box with a title bar.

Comments

Typically, the focus window and the active window are the same. Controls within dialog boxes, however, are an exception. The control has the focus but the dialog is active!

The **winfocus** command cannot be used to set the focus to a disabled or hidden window.

See also

taskwin, dlgwin, dlgctrlwin, enable, disable and winactivate; $FOCUSWIN in *"System Variables*."

winhide

Conceals the target window from view.

winhide window

window	The target window id.

Comments

Hiding a window prevents it from "seeing" mouse and keyboard events. If the window is currently active when it is hidden, another window will become active.

➥ *Be careful when hiding modal dialog boxes. A modal dialog box does not allow users to access Procomm Plus's parent window, and the user cannot access a hidden dialog!*

See also

disable, enable and winshow.

winmaximize

Maximizes an application window to fill the entire display screen. **winmaximize** only affects windows which have maximize buttons.

winmaximize window

window	The target window id.

See also

winminimize and winrestore.

winminimize

Minimizes an application window to an icon.

winminimize window

Comments

winminimize is only valid for windows which have a minimize button — typically these are top-level windows. If **winminimize** is used to minimize a main or top-level window which is currently disabled, you will be unable to restore the application from its iconic state.

See also

winmaximize and winrestore.

winmove

Positions an application window. **winmove** works only with windows which can be moved.

winmove window x y

window	The target window id.
x y	The top and left coordinates, respectively, of the desired location. Coordinates are measured in screen pixels.

Comments

Use the **winsize** command to change the size of the application window.

See also

wincoord and winsize.

winowner

A Returns the owner ID of a window ID.

winowner window intvar

window	The ID of the window whose ownership is unknown.
intvar	An *integer* variable to receive the owner window's ID.

Comments

A window that has no owner is actually owned by the desktop. FAILURE is set if the window has no owner, or if an invalid *id* was specified.

See also

taskwin, winactivate, winclose, winfocus, winstate, wintask, and wintext; $ACTIVEWIN, $FOCUSWIN, and $PWMAINWIN in "*System Variables*."

winrestore

Restores an application window to its previous size from a minimized or maximized state.

winrestore window

window	The target window id.

See also

winmaximize and winminimize.

winshow

Makes a hidden window visible.

winshow window

window	The target window id.

See also

disable, enable and winhide.

winsize

Sizes the specified application window.

winsize window dx dy

window	The window id.
dx	The new width of the target window, measured in screen pixels.
dy	The new height of the target window, measured in screen pixels.

See also

winhide, winminimize, winmaximize, winmove and winshow.

winstate

A Returns an integer reflecting the state of a window. If the window does not exist, **winstate** will report FAILURE.

winstate window intvar

window	The window id.
intvar	*Intvar* will equal 0 if the window is minimized, 1 if the window is restored and 2 if the window is maximized.

See also

winenabled, winvisible, winmaximize, winminimize, winrestore and winshow.

wintask

A Returns the task id of a window. **wintask** reports FAILURE if no task could be found for the given *window.*

wintask window intvar

window	The window id.
intvar	The task id. This value can be used with **taskactivate**, **taskexit**, and other **task**-related commands.

See also

taskwin; $MAINWIN, $TASK, $PWTASK, $ACTIVEWIN and $FOCUSWIN in "*System Variables*."

wintext

Retrieves the caption or title bar text of a window.

wintext window strvar

window	The window id. If the window is a dialog box control such as an **editbox** or **pushbutton**, **wintext** returns the text within the control rather than the dialog's caption. Note, however, that **wintext** cannot retrieve the text of an edit control within another application.
strvar	Will receive the requested text. *Strvar* will be null if the target window has no title text.

See also

dlgwin and dlgctrlwin; $ACTIVEWIN and $FOCUSWIN in "*System Variables*."

wintoscreen

Converts the X/Y coordinates that are relative to a window into absolute screen coordinates.

wintoscreen window intvar intvar

window	The window id.

intvar	The window X coordinate value to convert.
intvar	The window Y coordinate value to convert.

Comment

Screen and window coordinates are measured in pixels. Window coordinates are relative to the client area of a window, which begins at a point inside the window frame, if any, and below the caption and menu bars, if any.

➥ *The intvar arguments must be initialized to the values to convert.*

See also

screentowin, wincoord and uwutowin; $XPIXELS and $YPIXELS in *“System Variables*.”

wintouwu

Converts X/Y coordinates relative to a window into User Window Units.

wintouwu intvar intvar BACKGROUND|USERWIN

intvar	The User window X coordinate value to convert.
intvar	The User window Y coordinate value to convert.
BACKGROUND	Convert values relative to the User window background graphic.
USERWIN	Convert values relative to the User window proper.

Comment

Window coordinates are measured in pixels and are relative to the window’s client area. The client area of a window is the area that begins at a point inside the window frame, if any, and below the caption and menu bars, if any.

➥ *The intvar arguments must be initialized to the values to convert.*

See also

when mouseevent, uwutowin and wintoscreen; $LMOUSEX, $LMOUSEY and $USERWIN in “*System Variables*.”

winvisible

A Tests whether a window is hidden or visible. **winvisible** reports SUCCESS if the target window is visible. It reports FAILURE if the window is hidden or if a window could not be found using the specified window id.

winvisible window [intvar]

window	The target window id.
intvar	If specified, will be 1, meaning the window is visible or 0, meaning the window is hidden.

See also

winstate, winenabled, winexists, winhide and winshow.

wizard

A Activates a Procomm Plus wizard. The **wizard** command reports SUCCESS if the specified wizard is successfully finished. It reports FAILURE if the specified wizard was not found, or if the user presses the wizard's Cancel button.

wizard filespec]

filespec	A string variable or constant that contains the name of the wizard file.

Comments

The *filespec* can contain the name of any valid Procomm Plus wizard file. If a full path is not specified, the local task path is used by default.

Only one wizard may be executed at a time with ASPECT.

www

A Executes the specified Web action.

www {SAVE HTML | {TEXT filespec} | {OPEN {LOCAL filespec} | string} | PRINT

SAVE	Saves the current page.

HTML	Save in HTML format.
TEXT	Save in TEXT format.
filespec	The file to which the current page will be written.
OPEN	Opens the specified page.
LOCAL	Open a local file.
filespec	The HTML document to be loaded.
string	The address or URL to be loaded.
PRINT	Prints the current page.

Comment

www save html is an interactive command, since a dialog is displayed prompting for a filename. **www open local** resolves a file with respect to the current User Path.

The **www** command returns SUCCESS before the requested page is finished loading. If you want to know if and when the requested page finished successfully, use the $WWWSTATUS system variable.

See also

ftp and pwmode; $PWMODE, $WWWSTATUS in "*System Variables*."

xfercancel

Cancels a current file transfer operation. **xfercancel** has the same effect as the user pressing <Esc> or the ***Cancel*** button during a file transfer.

xfercancel

See also

sendfile and getfile; set aspect filexferbox in *"Set and Fetch Statements*."

xlatin

Converts a character or string using the current incoming *Translate Table* values.

xlatin intvar | {strvar strlength }

intvar	The character value to process.
strvar	The string to process.
strlength	The number of characters to process in the string.

See also

xlatout and xlatstr; set translate command in *"Set and Fetch Statements.*"

xlatout

Converts a character or string using the current outgoing *Translate Table* values.

xlatout intvar | { strvar strlength }

intvar	The character value to process.
strvar	The string to process.
strlength	The number of characters to process in the string.

See also

xlatin and xlatstr; set translate command in *"Set and Fetch Statements*."

xlatstr

Performs caret translation on a string.

xlatstr strvar intvar

strvar	The string to be translated.
intvar	Will be updated with the length of the translated string.

Comments

transmit, **waitfor**, **when** target, **termputs** and **termwrites** also perform caret translation. For more information, see "*Caret Translation*" on page 49.

See also

xlatin and xlatout.

yield

Suspends script execution, allowing other tasks within Procomm Plus to continue execution until ASPECT is permitted to resume script execution by the system.

yield

Comments

Under Windows NT 4.0, the following loop can monopolize CPU utilization when, in fact, the script needs to perform very little processing:

```
while 1 ; while TRUE would also work
.
.
.
endwhile
```

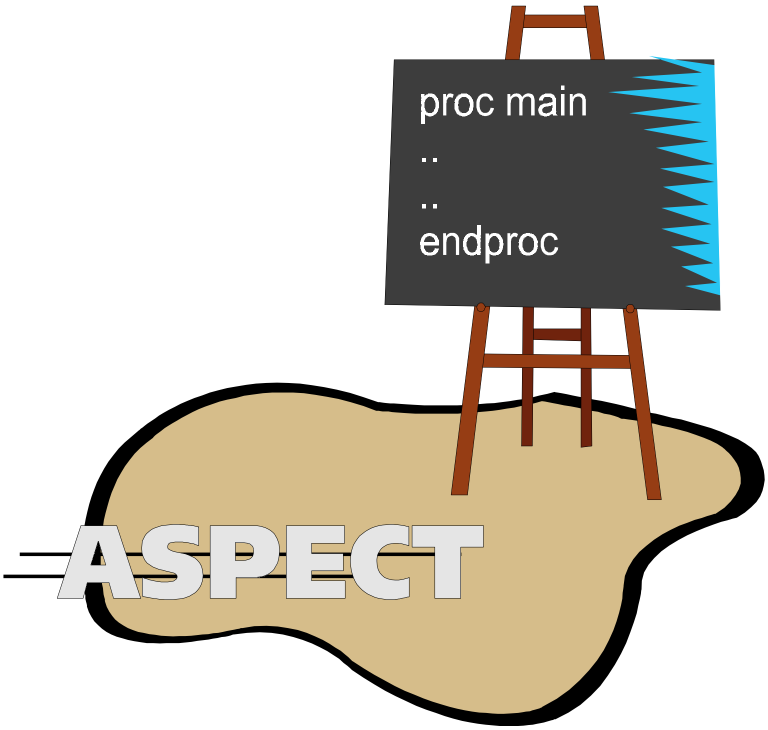
To prevent this wasteful CPU utilization, place a **yield** command at the end of the **while** loop.

See also

pause

Chapter 4

Set and Fetch Statements



Introduction

set and **fetch** commands are used to retrieve and change options in the *Connection Directory*, the *Setup* utility and other features of Procomm Plus. **fetch**ed values are placed into data variables of the appropriate type, and may be used directly in corresponding **set** commands.

Although most **set**/**fetch** commands will always succeed, all **set**/**fetch** commands update the SUCCESS and FAILURE system variables based on the results of the command. Certain **fetch** commands may fail if the information given to the command is invalid.

Using **set** to change *Connection Directory* entry data will fail when the *Connection Directory* is displayed.

On this documentation you'll find descriptions of all **set** and **fetch** statements grouped by type, such as the **set** ASPECT or **set** ZMODEM types. The first group of statements are general statements that do not fall into a specific group. Where general statements do relate to a specific statement group, a reference is provided so that you can find it more easily.

➥ *The **set** and **fetch** commands in the statements below are assumed, and are thus not included in the syntax. For example, **actionbar** must be preceded by **set** or **fetch** to form a valid command.*

The syntax for the **fetch** command generally follows that of its partner **set** command. For example, to determine the filename referenced as the top *Action Bar*, the **fetch** command would be:

```
fetch actionbar top strvar
```

since **fetch** *actionbar* returns a string value. To determine the number of seconds that the alarm would sound, the **fetch** command would be:

```
fetch alarmtime intvar
```

since **fetch** *alarmtime* returns an integer. In cases where the **set** command accepts an ON and OFF keyword, **fetch** will return 0 for OFF, and 1 for ON. For example:

```
fetch alarm intvar
```

would report the status of the ***Sound Alarm*** option in **Setup, Options, Sound & Alarm**.

The **set** and **fetch** commands are listed in alphabetical order, starting with the commands in "*General Set and Fetch Statements*" on page 365. Commands with related functionality are referenced by group names within the "*General Set and Fetch Statements*" section, such as

"*FAX Set and Fetch Statements*" on page 385. The commands within each group are documented after the "*General Set and Fetch Statements*" section.

General Set and Fetch Statements

actionbar {TOP | BOTTOM | LEFT | RIGHT | FLOAT} {filename | NONE | CURRENT} [PERMANENT [GLOBAL]]

Specifies the *Action Bar* file that's displayed for top, bottom, left, right, and float *Action Bars* in the communication window.

The PERMANENT keyword sets the specified *Action Bar* as the default for the current mode. If the PERMANENT keyword is not used, the previously specified *Action Bar* will be loaded when you next switch to this mode. The GLOBAL keyword indicates that the specified *Action Bar* should be used for all modes.

fetch returns a string containing the specified *Action Bar* filename and an integer set to 1 if the *Action Bar* is global or 0 if it is not.

The CURRENT keyword is only valid when the script is modifying a ***Data***- or ***Telnet***-class *Connection Directory* entry targeted with the **set dialentry access** command.

alarm OFF | ON

Enables or disables the alarm sound. This command corresponds to the ***Sound Alarm...*** check box in the **Setup, System, System Options** dialog. It has no effect on the ASPECT **alarm** command. **fetch** returns 0 for OFF or 1 for ON.

alarmtime integer

Specifies the number of seconds Procomm Plus sounds an alarm to signal the end of file transfers and dialing connections. The default is 2 seconds, and valid values range from 0 to 99. This command corresponds to the ***Sound alarm for x seconds*** field in the **Setup, System, System Options**. **fetch** returns the current number of seconds that the alarm sounds.

ASPECT—see "ASPECT Set and Fetch Statements" on page 375.

autodnld OFF | ON

Enables or disables auto-downloading for Zmodem and Kermit protocols. This command corresponds to the ***Enable automatic download start for Zmodem and Kermit*** check box in the **Setup, Data, Data Options, Advanced** dialog. **fetch** returns 0 for OFF or 1 for ON.

➥ *Control auto-downloading for the CIS-B+ protocol with the* ***set terminal enquiry*** *command.*

breaklen integer

Specifies the amount of time that Procomm Plus issues a break signal when the break command is used. Values range from 0 to 9999 in tenths of a second. This command corresponds to the ***Break Length*** edit field in the ***Setup, Data, Data Options, Advanced*** dialog. **fetch** returns an integer indicating the break length for the current connection.

callerid OFF | ON

Enables or disables Caller ID use throughout Procomm Plus. This command corresponds to the ***Use Caller ID*** check box in **Setup, System, System Options**. **fetch** returns 0 for OFF or 1 for ON.

callerid addcallinfo OFF | ON

Enables or disables Procomm Plus's ability to add caller information to the current database. This command corresponds to the ***Add incoming calls to Caller Information Directory:*** check box in **Setup, System, System Options**. **fetch** returns 0 for OFF or 1 for ON.

callerid database filename

Specifies the Caller ID database file to use for Caller Information logging, and Global Screening. This command corresponds to the ***Add incoming calls to Caller Information Directory:*** list box in the **Setup, System, System Options** dialog. **fetch** returns a string containing the filename.

CAPTURE—see "CAPTURE Set and Fetch Statements" on page 380.

cdinxfer OFF | ON

Determines whether Procomm Plus aborts file transfers when carrier is lost. This command corresponds to the ***Abort transfer if Carrier Detect signal lost*** check box in the **Setup, Data, Data Options, Advanced** dialog. **fetch** returns 0 for OFF or 1 for ON.

chatmode BLOCK | CHARACTER [CR_LF]

Specifies BLOCK or CHARACTER transmission of data in the *Terminal* window's *Chat*. This command corresponds to the ***Chat Mode transmit method:*** list in the **Setup, Data, Data Options, General** dialog. **fetch** *chatmode* requires two *intvar* arguments, returning 0 for BLOCK or 1 for CHARACTER in the first argument and 0 for no CR_LF or 1 for CR_LF in the second argument.

clipchar character

Specifies the separator character for filenames sent from the *File Clipboard*. The *character* value must be an integer (in octal, decimal, or hexadecimal notation) which corresponds to the ASCII value of one of the characters described in the ***File Clipboard separator*** list in the ***Setup, Data, Data Options, General*** dialog. **fetch** returns an integer equal to the ASCII value of the current separator character.

clipfilermv OFF | ON

Determines whether filenames are removed from the *File Clipboard* after they're sent to a remote system. This command corresponds to the ***Remove when sent*** check box in the *File Clipboard* dialog. **fetch** returns 0 for OFF or 1 for ON.

connection logging OFF | ON

Determines whether Procomm Plus records data, fax, and voice connections. This command corresponds to the ***Connection logging*** check box in the **Setup, System, System Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

connection statmsg OFF | ON

Determines whether status line messages related to the current connection are written to the connection log. This command corresponds to the ***Write status line messages to Connection Log*** check box in the **Setup, System, System Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

This option only controls status line messages related to the current connection. If connection logging is off, this command has no effect.

DIAL—see "DIAL Set and Fetch Statements" on page 381.

DIALENTRY—see "DIALENTRY Set and Fetch Statements" on page 382.

dnldpath pathname

Specifies the default path for received or downloaded files. This command corresponds to the ***Download path:*** field in the **Setup, Data, Data Options, Paths** dialog. **fetch** returns the contents of the field in a string.

dnldprompt OFF | ON

Enables or disables the display of the **Change File Transfer Protocol** dialog when a download is manually initiated. This command corresponds to the ***Prompt for download protocol before starting transfer*** check box in **Setup, Data, Data Options, Advanced**. **fetch** returns 0 for OFF or 1 for ON.

duplex FULL | HALF

Switches the duplex setting between full duplex and half duplex. In full duplex, data is not displayed to the screen as it is sent. In half duplex, Procomm Plus displays characters as they are sent. This command corresponds to the ***Duplex*** menu item on the Terminal window's ***Data*** menu. **fetch** returns 0 for FULL or 1 for HALF.

editor filespec

Specifies the program to run when the ***Edit*** or ***New*** button is selected in the **Compile/Edit ASPECT File** dialog. The default is **pwedit.exe,** the *ASPECT Editor*. This command corresponds to the ***Editor name & path*** field in the **Setup, System, System Options**. **fetch** returns the contents of the field in a string.

EMULATION...

Some terminal emulations have advanced command groups. These settings are described under "*Advanced Emulation Commands Set and Fetch Statements*" on page 407. You may also wish to reference "*TERMINAL Set and Fetch Statements*" on page 403.

FAX—see "FAX Set and Fetch Statements" on page 385.

FTP—see "FTP Set and Fetch Statements" on page 391.

iconflash OFF | ON

Iconflash determines whether the Procomm Plus icon flashes when a file transfer is completed and Procomm Plus is minimized. This command corresponds to the ***Flash icon when transfer complete*** check box in the **Setup, System, Data Options, Advanced** dialog. **fetch** returns 0 for OFF or 1 for ON.

INTERNET—see "INTERNET Set and Fetch Statements" on page 393.

largebars OFF | ON

Determines whether Procomm Plus will use large or small *Action Bar* icons. This command corresponds to the ***View | Large Action Bar*** menu item. **fetch** returns 0 for OFF or 1 for ON.

lmouse dblclick WORD | CHARACTER [CR]

Determines the action taken when the left mouse button is double-clicked. This command corresponds to the ***Left button double-click transmits:*** list in the **Setup, Data, Data Options, Left Mouse** dialog. **fetch** requires two *intvar* arguments, returning 0 for WORD or 1 for CHARACTER in the first argument and 1 in the second argument if the word or character is to be followed by a Carriage Return.

lmouse eolchar ENTERKEY | EOLSTR

Specifies whether an emulation **Enter** key press or a configurable string is sent after the word or character selected from the *Terminal* window by a Left mouse button double-click, when the ***Left button double-click*** field is set to send a Carriage Return. This command corresponds to the ***End of line character(s)*** option button in the **Setup, Data, Data Options, Left Mouse** dialog. **fetch** returns 0 for ENTERKEY or 1 for EOLSTR.

lmouse eolstr string

Specifies the text that is sent when ***End of line character(s)*** is not set to ***Enter key*** in the **Setup, Data, Data Options, Left Mouse Options** dialog. This command corresponds to the ***End of line character(s)*** edit field in the **Setup, Data, Data Options, Left Mouse** dialog. **fetch** returns a string containing the current value of the edit field. The end of line string has a maximum length of 7 characters.

lmouse prefix string

Determines the text that precedes information selected from the *Terminal* window or the *Scrollback Buffer* when that information is pasted. This command corresponds to the ***Selection paste prefix*** edit field in the **Setup, Data, Data Options, Left Mouse** dialog. **fetch** returns the current value of the edit field in a string. The prefix string has a maximum length of 7 characters.

longfilename OFF | ON

Determines whether the *File Clipboard* supports filenames that don't adhere to the 12-character DOS file-naming standard. This command corresponds to the ***Long filename support for File Clipboard*** check box in the **Setup, Data, Data Options, General** dialog. **fetch** returns 0 for OFF or 1 for ON.

MAIL—see "MAIL Set and Fetch Statements" on page 394.

metakeyfile filename | NONE | CURRENT

Loads the *Meta Key* file specified in the *filename* argument, which may or may not include a **.key** extension. This command can be tested with the **if** SUCCESS statement, returning false if the specified file cannot be loaded. This command corresponds to the ***Meta Keys*** list in the **Setup, Data, Setup Files** dialog. **fetch** returns the name of the currently-loaded *Meta Key* file in a string, or a null string if no file is currently loaded.

If the **set dialentry access** command has been used to specify a ***Data***- or ***Telnet***-class *Connection Directory* entry, **set/fetch metakeyfile** only affects that entry. Under this circumstance, **set metakeyfile none** selects the NONE value for that entry's ***Meta Key*** field. Use the CURRENT keyword to use the *Meta Key* file specified in the current *Setup*.

metakeys OFF | ON

Enables or disables the *Meta Key* display on the *Terminal* window. This command corresponds to the ***Meta Keys*** item on the ***View*** menu. **fetch** returns 0 for OFF or 1 for ON.

misc string

Specifies the miscellaneous string for a ***Data***-, ***Telnet***-, ***FTP***-, or ***Web***-class *Connection Directory* entry. This command corresponds to the ***Misc*** field in a *Connection Directory* entry's **Logon Info** dialog. **fetch** returns a string containing the ***Misc*** field contents of the currently-selected *Connection Directory* entry. Use the **set dialentry access** command to **set** the current *Connection Directory* entry.

Misc fields may also be read using the $MISC system variable. The maximum length of the ***Misc*** field is 40 characters.

MODEM—see "MODEM Set and Fetch Statements" on page 396.

NEWS—see "NEWS Set and Fetch Statements" on page 397.

notesfile filename | NONE | CURRENT

Specifies a default ***Notes*** file for *Setup* or for the current *Connection Directory* entry. If a current *Connection Directory* entry is selected with the **set dialentry access** command, this command affects that entry. If an entry is not targeted, this command affects the ***Notes File:*** list in the **Setup, System, System Options** dialog. **fetch** returns a string containing the name of the *Notes* file. The CURRENT keyword is valid only when setting the ***Notes*** file for a *Connection Directory* entry.

notespath pathname

Specifies the default path for ***Notes*** files. This command corresponds to the ***Notes path*** field in **Setup, Data, Data Options, Paths**. **fetch** returns the default path in a string.

password string

Specifies the password string for a ***Data***-, ***Telnet***-, ***FTP***-, or ***Web***-class *Connection Directory* entry. This command corresponds to the ***Password*** field in a *Connection Directory* entry's **Logon Info** dialog. The current *Connection Directory* entry can be specified by using the **set dialentry access** command. **fetch** returns the contents of the ***Password*** field in the current *Connection Directory* entry in a string. ***Password*** fields can also be read with the $PASSWORD system variable.

pausechar character

Identifies the character Procomm Plus translates into a half-second pause within modem commands and strings sent to the modem with ASPECT commands. The default character is a tilde (~), but any value from 32 to 126 is valid. This command corresponds to the ***Pause character*** field in the **Setup, System, System Options** dialog. **fetch** returns the numerical value of the current pause character as an integer.

playbackpace integer

Determines the speed at which information from a playback or capture file is displayed in the *Terminal window*. Valid values range from 0 to 6, 0 being slow and 6 being fast. This command corresponds to the ***File Playback Speed*** horizontal scroll bar in the **Setup, Data, Data Options, General** dialog**. fetch** returns the current playback speed as an integer.

PORT—see "PORT Set and Fetch Statements" on page 399.

PRINT—see "PRINT Set and Fetch Statements" on page 400.

protocol protocol | index | string

Specifies a default protocol for file transfers. You can either supply the name of the protocol as a keyword, give the zero-based index of the protocol within the ***Current Transfer Protocol*** list box in the **Setup, Data, Transfer Protocol** dialog, or provide a string containing the name of the protocol. **fetch** returns the name of the current default file transfer protocol in a string.

If you have used the **set dialentry access** command to a select a specific ***Data***- or ***Telnet***-class *Connection Directory* entry, the **set protocol** command affects that entry instead of *Setup*.

For a complete list of protocol names and indices, see "*Protocol Names and Indices*" on page 59.

PROTOCOL...

General comments regarding all of the protocols, in addition to each protocol's related command group, are listed beginning with "*ASCII Set and Fetch Statements*" on page 412.

quickoption index index

Specifies the information displayed in the selectable fields on the *Quick Select Line*. *index* is an integer value of 1, 2 or 3, corresponding to the first, second or third selectable field. Values of 1 to 7 are allowed for the second *index*. **fetch** returns values from 1 to 7, corresponding to *Modem Lights, Terminal Row/Column, Download Disk Space, Free Memory, Rx and Tx Buffer graph, Real-Time Clock* and *Current Aspect script*, respectively for the specified selectable field.

This command is only available in *Data*, Telnet, and *Web* modes.

quickselect OFF | ON

Determines whether the *Quick Select Line* is displayed on the *Terminal* window. This command corresponds to the ***View | Quick Select Line*** menu item. **fetch** returns 0 for OFF or 1 for ON.

This command is only available in *Data* and *Web* modes.

remotecmd OFF | ON [password]

Determines whether Procomm Plus responds to ASPECT commands sent from remote computers. *password* is an optional string constant or variable that you may use to configure the ***Remote command password***, which restricts the use of remote script commands by users dialing in to your system. This command corresponds to the ***Remote script commands...*** check box in **Setup, System, System Options**. **fetch** returns 0 for OFF or 1 for ON.

For security reasons, **fetch** will not return the value of the ***Remote script commands Password***.

rmouse action NONE | MENU | CURSORPOS

Determines the action of a right button click. This command corresponds to the ***Right mouse button action:*** list in the **Setup, Data, Data Options, Right Mouse** dialog. **fetch** returns 0 for NONE, 1 for MENU or 2 for CURSORPOS.

rmouse item index NONE | CURSORPOS | pw_menu_id

Determines the options that are available on the menu displayed when the right mouse button is clicked. *Index* is a number from 1 to 10 corresponding to each item in the right mouse button menu. *Pw_menu_id* is the ID of the item in the Procomm Plus main menu that you want to add to the right mouse button menu. This command corresponds to the ***'1'*** through ***'10'*** lists in the **Setup, Data, Data Options, Right Mouse** dialog. **fetch** returns 0 for NONE, 1 for

CURSORPOS or the ID of the Procomm Plus main menu item corresponding to the specified *index*.

The Procomm Plus menu IDs are defined in **pw4menu.inc**, which is installed by default in the ASPECT directory.

scriptpath pathname

Specifies the path Procomm Plus searches for ASPECT script files. This command corresponds to the ***Script path:*** field in the **Setup, Data, Data Options, Paths** dialog. **fetch** returns the current ASPECT script path in a string.

statusline OFF | ON

Determines whether the status line is displayed at the bottom of the *Terminal* window. This command corresponds to the ***Status Line*** item on the ***Setup*** menu. **fetch** returns 0 for OFF or 1 for ON.

This command is only available in Data and Web modes.

TELNET—"TELNET Set and Fetch Statements" on page 401

TERMINAL—see "TERMINAL Set and Fetch Statements" on page 403.

tooltips OFF | ON

Determines whether help labels appear below the *Action Bar* buttons when the mouse pointer is positioned over them. This command corresponds to the ***View | ToolTips*** menu item. **fetch** returns 0 for OFF or 1 for ON.

translate filename | NONE | CURRENT

Specifies the default *Translate Table* in *Setup* or the *Translate Table* to use for a ***Data***- or ***Telnet***-class *Connection Directory* entry. If used in conjunction with the **set dialentry access** command, this command affects the ***Translate Table:*** list in the current *Connection Directory* entry's **Advanced Options** dialog. The default extension is **.xlt**, but it need not be specified. The CURRENT keyword causes a *Connection Directory* entry to use the *Translate Table* specified in *Setup*.

If **set dialentry access** has not been specified, this command affects the ***Translate Table:*** list in the **Setup, Data, Setup Files** dialog. **fetch** returns a string containing the name of the current *Translate Table*.

txpace integer

Determines the length of pauses between transmitted characters. This command corresponds to the ***Transmit pacing (milliseconds):*** field in the **Setup, System, System Options** dialog. Valid values range from 0 to 999, in milliseconds. **fetch** returns an integer equal to the current transmit pacing value.

upldpath pathname

Specifies the default path for upload files. This command corresponds to the ***Upload path:*** field in the **Setup, Data, Data Options, Paths** dialog. **fetch** returns the current default upload path in a string.

userid string

Specifies the contents of the ***User ID*** edit field in a ***Data***-, ***Telnet***-, ***FTP-***, or ***Web***-class *Connection Directory* entry. The maximum length for the ***User ID*** string is 40 characters. **fetch** returns the contents of the ***User ID*** field from the current *Connection Directory* entry's **Logon Info** dialog in a string. The $USERID system variable may also be used to read the contents of this field.

The **set dialentry access** command can be used to specify the desired *Connection Directory* entry.

viewgif OFF | ON

Determines **.gif** graphics are displayed as they are downloaded. This command corresponds to the ***View .GIF files during download*** check box in the **Setup, Data, Data Options, Advanced** dialog. **fetch** returns 0 for OFF or 1 for ON.

wavefile index filespec

Specifies the name of a wave file to play when a specific event occurs. *Index* is a number from 1 to 9, specifying the event wave file to **set**. **fetch** returns the name of the wave file corresponding to the specified *index* in a string. The *index*es and their corresponding *Setup* fields include:

1	***File transfer complete*** field in the **Setup, Data, Data Options, General** panel.
2	***File transfer abort*** field in the **Setup, Data, Data Options, General** panel.
3	***Connect*** field in the **Setup, Data, Data Options, General** panel.
4	***Disconnect*** field in the **Setup, Data, Data Options, General** panel.

5	***Fax receive complete*** field in the **Setup, Fax, Fax Options** panel.
6	***Fax send complete*** field in the **Setup, Fax, Fax Options** panel.
7	***Fax send/receive failure*** field in the **Setup, Fax, Fax Options** panel.
8	***Calling queue*** complete field in the **Setup, System, System Options** panel.
9	***Miscellaneous errors*** field in the **Setup, System, System Options** panel.

wincolors index | string | CURRENT

Specifies the ***Current Window Colors*** options set. *Index* is a zero-based value, corresponding to the option set entry of the desired ***Current Window Colors*** options set. *String* is the name of the desired ***Current Window Colors*** options set. This command corresponds to the ***Current Window Colors:*** field in the **Setup, Data, Window Colors** dialog**. fetch** returns the name of the ***Current Window Colors*** options set in a string.

WWW—see "WWW Set and Fetch Statements" on page 423.

xferyield integer

Determines how much processor time Procomm Plus gives to other applications while a file transfer is in progress. Valid values range from 0 to 17, 0 meaning that Procomm Plus will yield very little and 17 meaning that it will yield generously. This command corresponds to the ***Yield During File Transfer*** horizontal scroll bar in the **Setup, Data, Data Options, Advanced** dialog. **fetch** returns the current setting in an integer.

ASPECT Set and Fetch Statements

The following **set aspect** statements control how Procomm Plus operates when running a script. Each setting has a default value, which is assigned when a script is launched.

Scripts that are launched using the **chain** or **execute** command only receive default values for the *scriptpath* and *remotecmd* settings. All other settings are inherited from the script which launched them.

aspect codepage integer

Specifies the information used to convert characters between character sets to aid in internationalization. This information is used by the **keytoansi**, **ansitooem** and **oemtoansi** commands to perform proper character set conversions. By default, this option is set to the code page used by Windows. Acceptable settings are shown below:

0	ANSI - no conversions applied
255	Windows currently-loaded code page
437	Default - North American.
850	International
860	Portugal
861	Iceland
863	French Canadian
865	Norway / Denmark

fetch returns the code page currently used by Procomm Plus in an integer.

aspect control OFF | ON

Enables and disables the ***Tools | Scripts | Start Script/Stop Script*** menu item, as well as the *Action Bar* ***Run Script*** button. If OFF, these controls operate normally. If ON, they are disabled, effectively preventing a user from interrupting a script. **fetch** returns 0 for OFF or 1 for ON.

Use this command in cases where you wish to guarantee that the script completes its actions without user intervention. Because this command prevents the user from shutting down Procomm Plus specifically, the only way to shut it down is by exiting Windows. In this case, Procomm Plus will prompt the user for permission to shut down all active scripts—this prompt occurs independent of the **aspect exitaction** setting. Also, a script that sets **aspect control on** may lose that control if it allows scripts to spawn on top of it, or if it **execute**s a child script without sharing the environment. The default setting for this command is OFF.

aspect ctrlbreak OFF | ON

Allows a user in *Terminal*- or *Telnet*- mode to abort a dead-end command like **pause** of **waitfor** with the <Ctrl><Break> key sequence. By default, this is set to ON. **fetch** returns 0 for OFF or 1 for ON.

aspect datakey integer

Specifies the integer key used in the **encrypt** and **decrypt** commands. **fetch** returns the value of the current integer key.

For more information on the use of the datakey in encryption, see the **encrypt** command and the **decrypt** command.

aspect decimal integer

Determines the default number of digits to the right of the decimal point to display for floating point numbers. This command affects the floating point precision used in **ftoa** and all commands that accept a format string. **fetch** returns an integer equal to the current decimal precision.

The default value for this setting is 2, but any value from 0 to 255 can be used. Any precision specifiers occurring in a format string override the default precision. The last digit in the displayed value is rounded as needed. For example, if the default precision is 2, a value of 123.456789 would become 123.46.

aspect dialingbox OFF | ON

Determines whether the Dialing Progress dialog is displayed after a **dial** or **dialnumber** command is processed. If the script stops running while this command is in effect, the Dialing Progress dialog is displayed immediately. **fetch** returns 0 for OFF or 1 for ON.

If the Dialing Progress dialog has been turned off, the Procomm Plus main window remains disabled, preventing any interaction with menus and *Action Bars*. Press the <Escape> key to cancel the dialing operation if you wish to reenable the main window.

If the Dialing Progress dialog is disabled, the user will not be able to select a new timeout value, or to cancel the dialing activity manually. This setting only affects dialing operations that follow it. It has no effect upon dialing operations underway when the command is executed.

aspect display OFF | ON

Determines whether characters received from a remote system are displayed on the *Terminal* window. **fetch** returns 0 for OFF or 1 for ON.

No characters are written to a *Capture* file or to the *Scrollback Buffer* when **set aspect display** OFF is used, nor are they printed if print capture is enabled. To display or capture data with **set aspect display** OFF, use **set aspect rxdata** ON and issue **termwritec** or **termwrites** commands as data is received.

aspect errormsg OFF | ON

Prevents ASPECT from displaying error messages that are not considered critical errors. **fetch** returns 0 for OFF or 1 for ON.

The $ERRORNUM system variable will contain a non-zero value when an error has occurred. It can be used in conjunction with the **when** command to build an error handling routine when the *errormsg* setting is OFF.

aspect exitaction PROMPT | HALT

Determines whether or not the user is prompted to exit all parent scripts, if any, when the current script is stopped. This can occur when the user presses the ***Run Script*** *Action Bar* button or selects ***Script Stop*** form the ***Tools | Scripts*** menu. **exitaction** also determines whether the user is prompted to exit all active scripts when Procomm Plus is shut down. The default action is PROMPT, which displays a message box prompt to the user. HALT stops all running scripts without prompting the user. This setting is ignored when **aspect control** is ON. **fetch** returns 0 for PROMPT or 1 for HALT.

aspect filexferbox OFF | ON

Determines whether Procomm Plus displays the file transfer progress dialog during a **sendfile** or **getfile** operation. **fetch** returns 0 for OFF or 1 for ON.

With **set aspect filexferbox** OFF, file transfers can only be canceled with the **xfercancel** command. The **set aspect filexferbox** setting only controls file transfer operations which follow it. File transfers already underway when the command is executed will not be affected.

aspect helpfile filespec

Specifies the name of the help file to use with on-line help. Typically, this is a custom help file written especially for a script. The **help** command can be used to open help and display the help file specified by this command. **fetch** returns the name of the current help file in a string.

The Aspect Path is the default path assumed for the ASPECT help file, and if no extension is specified, **.hlp** is assumed.

aspect keys OFF | ON

Determines how keystrokes are processed when a script is executed. With **set aspect keys** OFF, Procomm Plus processes all keystrokes normally. With **set aspect keys** ON, the script must process all keystrokes. **fetch** returns 0 for OFF or 1 for ON.

The $KEYHIT system variable will contain a non-zero value when a key is pressed. It can be read with the **keyget** command and processed normally by Procomm Plus with the **termvkey** command.

aspect mousecoord window | SCREEN

Specifies which window ID that the mouse coordinates will be relative to. SCREEN is the default.

aspect path pathname

Specifies the path, called the Aspect Path, referenced by the dialog box and user window commands. When a filename without a path component is specified within a *filespec* argument for one of these commands, ASPECT will look for that file within the Aspect Path. **fetch** returns the current default path value in a string. The $ASPECTPATH system variable can also be referenced for the current Aspect Path.

The Aspect Path is also assumed for the **chain**, **execute**, **compile**, **dllobjfile**, and **dllload** commands. The Aspect Path can be changed with the **set aspect path** command, or as an optional action in the **chdir** command.

For more information, see "*The ASPECT Script Environment*" on page 54.

aspect rgetchar character [STRIP]

Specifies the character which terminates an **rget** command. The default character is a Carriage Return, ASCII 13. If STRIP is specified, the terminating character is removed from the string returned by **rget**. **fetch** returns the current **rget** termination character as an integer.

aspect rxdata {OFF [UNTIL WHENTARGET index]} | ON

Specifies whether Procomm Plus or a script processes characters received from the communication port. The default, OFF, indicates that Procomm Plus processes received characters. With **set aspect rxdata** ON, the script must handle incoming data with commands like **comgetc**. With **set aspect rxdata** ON, commands dependent upon received data, such as **waitfor** and **when** *target* are only checked when characters are retrieved from the receive data buffer. **fetch** returns 0 for OFF or 1 for ON.

Setting **set aspect rxdata** ON does not prevent autodownload sequences from being recognized. You must explicitly **set autodnld** OFF to prevent Kermit and Zmodem transfers from starting, and **set terminal enquiry** OFF to prevent CISB+ transfers from starting.

The optional UNTIL WHENTARGET forces Procomm Plus to process characters normally until the **when** *target* specified by the *index* is triggered. At that time, the script is expected to process any characters received after the target string.

aspect spawn OFF | ON

Determines whether a script may be executed while another script is running. If ON, scripts can be spawned from a *Connection Directory* entry, in response to a *Meta Key* press, via a **ddeexecute** command or by other methods. By default, this setting is OFF when a script is initially executed. **fetch** returns 0 for OFF or 1 for ON.

This command has no effect on the **execute** and **chain** commands.

aspect whensuspend OFF | ON

Determines whether events specified by **when** statements are acted upon while a procedure called by another **when** statement is executing. If this setting is ON, all other **when** events are suspended until the procedure called by the currently-executing **when** event returns. **fetch** returns 0 for OFF and 1 for ON.

Suspending **when** conditions does not eliminate ASPECT's response to those conditions. Instead, the suspended **when**s are saved in a queue until the currently-executing **when** command has completed. This may result in unexpected **when** firings if the script is not configured to account for **when** events that occur during the suspended period. This is true of **when** ELAPSED, **when** ISKEY, **when** QUIET, **when** TARGET and the **when** USEREXIT commands. It is not true of **when** variable commands, meaning that changes to a variable which occur during a suspended period will not cause a **when** variable to fire.

CAPTURE Set and Fetch Statements

The **set capture** statements control the operation of the Procomm Plus *Capture* files. They correspond to the ***Capture File Options*** in the **Setup, Data, Setup Files** dialog. If **set dialentry access** is used to specify a ***Data***- or ***Telnet***-class *Connection Directory* entry, the *autostart, file, overwrite, query* and *recordmode* **set** statements only affect the ***Capture File Options*** of that entry. The **Capture File Options** dialog is accessed by clicking ***Setup*** within a *Connection Directory* entry's **Basic Options** dialog, or *Setup*'s **Data, Setup Files** dialog.

capture autostart OFF | ON

If *capture autostart* is ON, Procomm Plus opens a *Capture* file when you connect using a ***Data***- or ***Telnet***-class *Connection Directory* entry, and closes it upon disconnect. This command corresponds to the ***Start capture on connection*** field in the *Connection Directory* **Capture File Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

capture file filename | NONE | CURRENT

Specifies the default filename for a capture file. If **set** to NONE, the *Capture* file is disabled. **fetch** returns the name of the current default capture file in a string.

The **set dialentry access** command may be used to target a ***Data***- or ***Telnet***-class *Connection Directory* entry for the **set capture file** command. If CURRENT is specified, Procomm Plus logs data to the file specified in **Setup, Data, Setup Files**. The CURRENT keyword is valid for *Connection Directory* entries only.

capture overwrite OFF | ON

Determines whether capture files are appended to or overwritten when capturing to existing files. This command corresponds to the ***Overwrite existing...***check box. **fetch** returns 0 for OFF or 1 for ON.

capture path pathname

Determines the default path for *Capture* files. This command corresponds to the ***Capture path*** field in **Setup, Data, Data Options, Paths**. **fetch** returns the current default path for capture files in a string.

This command pertains to *Setup* only, regardless of **set dialentry access**.

capture query OFF | ON

Determines whether Procomm Plus prompts for a filename when a capture is started. This command corresponds to the ***Query for file name...*** check box. **fetch** returns 0 for OFF or 1 for ON.

capture recordmode SCREEN | FILTERED | RAW

Determines whether Procomm Plus writes information to capture files just as the information appears on the *Terminal* display (SCREEN), writes information without emulation controls (FILTERED), or writes information just as it is received from the remote system (RAW). This command corresponds to the ***Recording Options*** radio group. **fetch** returns 0 for SCREEN, 1 for FILTERED, or 2 for RAW.

With **set capture recordmode** RAW, the capture file will contain any emulation sequences sent by the host. These may appear as high-bit characters in your *Capture* files.

DIAL Set and Fetch Statements

The **set dialentry** commands are used to change the settings for specific *Connection Directory* entries.

When *Connection Directory* changes are complete, the **dialsave** command should be issued to save them.

dial location string

Specifies the location name appearing in the ***Current Location*** field in the **Setup, System, Dialing Options** dialog. **fetch** returns the current location in a string.

dial retries integer

Specifies the maximum number of times that Procomm Plus will attempt to connect with a *Connection Directory* entry. Valid values range from 0 to 999. This command corresponds to the ***Make x attempts and...*** field in the **Setup, System, Dialing Options** dialog. **fetch** returns the maximum number of dial attempts as an integer.

dial retrydelay integer

Specifies the number of seconds Procomm Plus pauses between dial attempts when redialing a number. Valid values range from 0 to 99. This command corresponds to the ***... pause x seconds between attempts*** edit field in **Setup, System, Dialing Options**. **fetch** returns the current value in an integer.

DIALENTRY Set and Fetch Statements

The **set**/**fetch** *dialentry* commands are used to retrieve and change the settings for specific *Connection Directory* entries. Other **set**/**fetch** commands that are common between the *Connection Directory* and *Setup* can also be used to retrieve and change the settings for *Connection Directory* entries. The **set dialentry access** command allows you to select the *Connection Directory* entry that these commands affect. When *Connection Directory* changes are complete, the **dialsave** command must be issued to save them.

Using **set** to change *Connection Directory* entry data will fail when the *Connection Directory* is open.

dialentry access OFF | {dialclass string}

Specifies the *Connection Directory* entry by name and class that subsequent **set** commands affect.

fetch dialentry acess is shown separately for clarity.

dialentry access intvar strvar

fetch dialentry access returns an integer containing the type of *Connection Directory* entry and the name of the specified entry in a string. If **set dialentry access is** OFF, **etch,** the *intvar* is assigned to 1 (DATA) and the *strvar* is a null string. The *intvar* can have any of the values that represent a single dialclass. For further information you may wish to see the **dialclass** command.

The **dialname** command can be used to get the name of a specific entry, based on its index within the *Connection Directory*. For more information, see the **dialname** command.

dialentry anonymouslogon OFF | ON

Specifies whether or not this entry uses anonymous logon or not. This command corresponds to the ***Anonymous logon*** check box in the *FTP* section of the *Connection Directory.*

In order for this command to be effective, the *dialclass* specified in the **set/fetch dialentry access** command must have been FTP.

dialentry areacode string

Specifies the Area Code for the selected *Connection Directory* entry. This command corresponds to the ***Area Code*** edit field in the selected ***Data-***, ***Fax-***, or ***Voice-***class *Connection Directory* entry. **fetch** returns the entry's area code in a string.

dialentry company string

Specifies the contents of the ***Company*** edit field for the selected *Connection Directory* entry. **fetch** returns the contents of the currently-selected *Connection Directory* entry's ***Company*** edit field in a string.

dialentry country string [strvar]

Specifies the contents of the ***Country*** field within the selected *Connection Directory* entry. **fetch** allows the optional string variable which is assigned the dialing code for the current ***Country*** selection. This command is only valid for ***Data-***, ***Fax-***, or ***Voice-*** class entries.

dialentry dialnumberonly OFF | ON

Specifies whether Procomm Plus should dial only the ***Data number*** for the current entry. **fetch** returns 0 for OFF and 1 for ON. This command is only valid for ***Data-***, ***Fax-***, and ***Voice-***class entries.

dialentry hostdir string

Specifies the directory on the host system to view. This command corresponds to the ***Default Host Directory:*** field in the **FTP, Basic Options** dialog of the *Connection Directory*. **fetch** returns the current contents of the field in a string, or a null string if the field is blank.

In order for this command to be effective, the *dialclass* specified in the **set/fetch dialentry access** command must have been FTP.

dialentry hosttype index

Specifies what type of host is on the remote system. This command corresponds to the ***Host Type:*** list in the **FTP, Basic Options** dialog of the *Connection Directory*. **fetch** returns the current index value in an integer.

In order for this command to be effective, the *dialclass* specified in the **set/fetch dialentry access** command must have been FTP.

dialentry ipaddress string

Specifies the IP Address used to contact a host for a ***Telnet***-, ***FTP***-, ***Web***-, ***Mail***-, or ***News***-class *Connection Directory* entry. This command corresponds to the ***Host/IP Address:***, ***Address:, Mail Address:,*** or ***News Group:*** field in the *Connection Directory*. **fetch** returns the current contents of the field.

dialentry ipport integer

Specifies which TCP/IP port to use when receiving data from the host. This command corresponds to the ***Telnet Port*** or ***FTP Port*** fields in the corresponding **Options** dialogs of a ***Telnet***- or ***FTP***-class *Connection Directory* entry. **fetch** returns the contents of the field in an integer.

dialentry logontimeout integer

Specifies how long Procomm Plus will wait for a response from the host while in *FTP* mode. This command corresponds to the ***Logon Timeout:*** field in the **FTP, FTP Options** dialog in the *Connection Directory*. **fetch** returns the value of the field in an integer.

dialentry longdistance OFF | ON

Specifies whether or not the *Connection Directory* treats the entry as a long distance number. This command corresponds to the ***Always dial as long distance*** check box in ***Data-***, ***Fax-***, or ***Voice-*** class entries. **fetch** returns a zero for OFF and a 1 for ON. This command is valid only for ***Data-***, ***Fax-***, and ***Voice-***class entries.

dialentry phonenumber string

Specifies the phone number to use for the selected ***Data***-, ***Fax***-, or ***Voice***-class *Connection Directory* entry.

➥ ***fetch dialentry phonenumber*** *is shown separately for clarity.*

fetch dialentry phonenumber strvar

fetch dialentry phonenumber returns a string containing the phone number for the selected *Connection Directory* entry. This command is valid if the *dialclass* specified in the **set/fetch dialentry access** command was either DATA, FAX, or VOICE.

dialentry scriptfile filename | NONE

Specifies the script file to be associated with the selected ***Data***-, ***Telnet-***, ***FTP-***, or ***Web***-class *Connection Directory* entry. This command corresponds to the *Connection Directory* entry's ***Script*** edit field. **fetch** returns the name of the selected entry's script filename in a string.

If the *filename* argument is specified, a filename extension is optional. However, if an extension is included, it must be **.wax**. **set dialentry scriptstart** may be used to determine when the script file should run.

dialentry scriptstart CONNECTED | DIALED

Specifies when the selected ***Data***-, ***FTP-***, or ***Web***-class *Connection Directory* entry's associated script file is run. This command corresponds to the settings in the **Script Execution Options** dialog, opened with the ***Script Setup...*** button in the *Connection Directory* entry. If DIALED is specified, the script is launched when the entry is dialed. If CONNECTED is used, the script is launched when a connection is established with the entry. **fetch** returns 0 for CONNECTED or 1 for DIALED.

dialentry terminalid string

Specifies the ID string to send when negotiating a Telnet connection. This command corresponds to the ***When negotiating...*** field in the **Telnet Options** dialog of the *Connection Directory*. **fetch** returns the contents of the field in a string or a null string if the field is empty.

In order for this command to be effective, the *dialclass* specified in the **set/fetch dialentry access** command must have been TELNET.

FAX Set and Fetch Statements

The **set** and **fetch** *fax* commands are used to retrieve and change Procomm Plus's current fax options. They correspond to options in *Setup*'s **Fax** dialogs.

__set fax__ commands that correspond to the fields on the __System, Modem Connection, Fax Settings__ dialog in Setup will fail for all "direct connect" and __.dlc__ connections like Telnet.

All __set fax__ commands will fail if Procomm Plus is currently dialing, if a file transfer is in progress, or if another instance of Procomm Plus exists.

fax adaptive19200 OFF | ON

Specifies whether or not to change the baud rate to 19200 when performing *Adaptive Answer*. This command corresponds to the ***Automatically switch baud rate to 19,200...*** check box in the **Setup, System, Modem Connection, Fax Settings, Adaptive Answer** dialog. **fetch** returns a 0 if the check box is not enabled or a 1 if it is.

fax adaptiveans CLASS1 | CLASS2 string

Specifies the modem commands used to enable adaptive answer for Class 1 and Class 2 fax. This command corresponds to the ***Enable command:*** fields in the ***Class 1 Adaptive Answer*** and ***Class 2 Adaptive Answer*** groups of the **Setup, System, Modem Connection, Fax Settings, Adaptive Answer** dialog. **fetch** returns the contents of the specified adaptive answer command's edit field in a string.

fax baudrate long

Specifies the DTE speed to use when faxing. This command corresponds to the ***Fax baud rate (DTE)*** field in the **Setup, System, Modem Connection, Fax Settings, General** dialog. **fetch** returns the current value in a *long*. Possible values include 19200, 28800, 38400, 57600, or 115200.

fax class CLASS1 | CLASS2 | SIERRA

Specifies the standard used to transmit faxes. This command corresponds to the ***Fax class:*** list in the **Setup, System, Modem Connection, Fax Settings, General** dialog. **fetch** returns 1 for CLASS1, 2 for CLASS2, or 4 for SIERRA.

fax cnctmsg CLASS1 | CLASS2 string

Specifies the connect message for fax adaptive answer connections. This command corresponds to the ***Fax connect message*** edit fields in the ***Class 1 Adaptive Answer*** and ***Class 2 Adaptive Answer*** groups of the **Setup, System, Modem Connection, Fax Settings, Adaptive Answer** dialog. **fetch** returns the contents of the ***Adaptive answer connect message*** field in a string.

fax company string

Specifies the contents of the ***Company:*** edit field in the **Setup, Fax, User Info, User Info** dialog. **fetch** returns contents of the ***Company*** field in a string.

fax coversheet filename | NONE

Specifies the default fax cover sheet used in transmitted faxes. This command corresponds to the ***Default cover sheet:*** list in the **Setup, Fax, User Info, User Info** dialog. **fetch** returns the contents of the ***Default cover sheet*** list in a string. If no default cover sheet is defined, **fetch** returns a null string.

fax delpages OFF | ON

Determines whether fax pages are deleted after a fax is sent. This command corresponds to the ***Delete pages after send*** check box in the **Setup, Fax, Fax Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

fax ecm receive NONE | SOFTWARE | HARDWARE

Specifies the type of error correcting method Procomm Plus uses when receiving faxes in Class 1 mode. This command corresponds to the ***Fax ECM during receive...*** list in the **Setup, System, Modem Connection, Fax Settings, Advanced** dialog. **fetch** returns a 0 for NONE, a 1 for SOFTWARE, and a 2 for HARDWARE.

fax ecm send NONE | SOFTWARE | HARDWARE

Specifies the type of error correcting method Procomm Plus uses when sending faxes in Class 1 mode. This command corresponds to the ***Fax ECM during send...*** list in the **Setup, System, Modem Connection, Fax Settings, Advanced** dialog. **fetch** returns a 0 for NONE, a 1 for SOFTWARE, and a 2 for HARDWARE.

fax faxnumber string

Specifies the fax number displayed on fax cover sheets and in the header at the top of fax documents. This command corresponds to the ***Fax number*** field in the **Setup, Fax, User Info, User Info** dialog. **fetch** returns the contents of the ***Fax number*** field in a string.

fax flowcontrol SOFTWARE | HARDWARE

Specifies the type of flow control used when sending faxes. This command corresponds to the ***Flow control*** list in the **Setup, System, Modem Connection, Fax Settings, General** dialog. **fetch** returns 0 for SOFTWARE or 1 for HARDWARE.

fax header PAGENUMBERS | TIMESTAMP | {RECEIVER USERNAME | FAXNUMBER} | {SENDER USERNAME | COMPANY | FAXNUMBER | VOICENUMBER | STATIONID} OFF | ON

Specifies the user information included in the header of transmitted faxes. This corresponds to the check box options found in the ***Include in Fax Header*** groupbox in the **Setup, Fax, User Info, Fax Header** dialog**.** These include ***Page numbers***, ***Date and time***, ***Receiver's fax number***, ***Receiver's name***, ***Sender's name***, ***Sender's company***, ***Sender's fax number***, ***Sender's voice number***, and ***Sender's station ID***.

fetch returns 0 if the specified option is OFF or 1 if it is ON.

fax init string

Specifies the command string sent to the modem before a fax is transmitted. This command corresponds to the ***Fax initialization string*** edit field in the **Setup, System, Modem Connection, Fax Settings, General** dialog. **fetch** returns the contents of the ***Fax initialization string*** field in a string.

fax linetype CONVENTIONAL | CELLULAR

Specifies the type of phone line the fax is being sent over. This command corresponds to the ***Telephone line type:*** list in the **Setup, System, Modem Connection, Fax Settings, General** dialog. **fetch** returns 0 for CONVENTIONAL and 1 for CELLULAR.

fax manager OFF | ON

Specifies whether *Fax Manager* is run after a fax is received. This command corresponds to the ***Run Fax Manager*** check box in the **Setup, Fax, Fax Options** dialog. **fetch** returns 0 if the check box is not enabled, or 1 if it is enabled.

fax options index | string

Specifies the set of saved options used by Procomm Plus. A zero-based index value or a string specifying the name of the options set can be used. This command corresponds to the ***Current User Info*** list in the **Setup, Fax, User Info** dialog. **fetch** returns the name of the options set selected in a string.

fax path coversheet pathname

Specifies the path Procomm Plus will search for fax cover sheets. This command corresponds to the ***Fax cover sheet files:*** edit field in the **Setup, Fax, Fax Paths** dialog. **fetch** returns the contents of the ***Fax cover sheet files*** field in a string.

fax path inbox pathname

Specifies the path where incoming fax documents are stored by Procomm Plus. This command corresponds to ***Received faxes:*** field in the **Setup, Fax, Fax Paths** dialog. **fetch** returns the contents of the ***Received faxes*** field in a string.

fax path sent pathname

Specifies the path where faxes are stored after they've been sent. This command corresponds to the ***Sent faxes directory:*** field in the **Setup, Fax, Fax Paths** dialog. **fetch** returns the contents of the ***Sent faxes directory:*** field in a string.

fax path outbox pathname

Specifies the path where faxes are stored before they're transmitted. This command corresponds to the ***Scheduled faxes*** edit field in the **Setup, Fax, Fax Paths** dialog. **fetch** returns the contents of the ***Scheduled faxes*** field in a string.

fax recvbaud long

Specifies the maximum baud rate used when receiving faxes. This command corresponds to the ***Maximum receive rate*** field in the **Setup, System, Modem Connection, Fax Settings, General** dialog. **fetch** returns a long value representing the current maximum receive rate. Possible values are 2400, 4800, 7200, 9600, 12000, or 14400.

fax recvprint OFF | ON

Determines whether faxes are printed upon receipt. This command corresponds to the ***Print fax*** check box in the **Setup, Fax, Fax Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

fax recvview OFF | ON

Determines whether faxes are displayed upon receipt. This command corresponds to the ***View fax*** check box in the **Setup, Fax, Fax Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

fax retries integer

Determines the number of send fax dialing attempts Procomm Plus will make before aborting; valid values range from 0 to 99. This command corresponds to ***Make up to x attempts*** field in the **Setup, Fax, Fax Options** dialog. **fetch** returns an integer reflecting the current number of dialing attempts.

fax retrydelay integer

Determines the number of seconds Procomm Plus pauses between send fax attempts. This command corresponds to the ***Retry every x minutes*** field in the **Setup, Fax, Fax Options** dialog. **fetch** returns an integer reflecting the current ***Retry interval*** value.

fax reversebit {PORT | FAX} {SEND | RECEIVE} {OFF | ON | DEFAULT}

Determines whether Procomm Plus uses a reverse bit order when sending or receiving faxes. This command corresponds to the four ***Reverse...*** check boxes in the **Setup, System, Modem Connection, Fax Settings, Advanced** dialog. Depending on the first two parameters, **fetch** will return a 0, 1, or 2, representing OFF, ON, and DEFAULT respectively.

fax rmvpolled OFF | ON

Determines whether files set for polling are removed after they're successfully transmitted. This command corresponds to the ***Remove from polling list once polled*** check box in the **Files for Polling by a Fax Caller** dialog, which is available from the ***Fax | Select Files for Polling...*** menu in *Fax Status*. **fetch** returns 0 for OFF or 1 for ON.

fax scheduled RESUME | HOLD

Determines whether Procomm Plus will send or hold all faxes. This command corresponds to the ***Hold/Resume all faxes*** menu item on the ***Fax*** menu in *Fax Status*. **fetch** will return a 0 if all faxes are being sent or a 1 if faxes are on HOLD.

fax sendpolled OFF | ON | AUTO

Determines whether faxes in the polling list are sent when Procomm Plus is polled by a remote fax system. This command corresponds to the ***Send Faxes When Polled*** radio buttons in the **Files for Polling by a Fax Caller** dialog, available from the ***Fax | Files for Polling...*** menu in *Fax Status*. **fetch** returns 0 for OFF, 1 for ON, or 2 for AUTO.

fax showfaxstatus {SEND | RECEIVE} {BEGIN | COMPLETE} {OFF | ON}

Determines when *Fax Status* should be brought to the foreground. This command corresponds to the ***Fax Receive begins, Fax Receive is complete, Fax Send begins,*** and ***Fax Send is complete*** check boxes available in the **Setup, Fax, Fax Options** dialog.

fax stationid string

Specifies the station ID displayed on fax cover sheets and fax document headers. This command corresponds to the ***Station ID*** edit field in the **Setup, Fax, User Info, User Info** dialog. **fetch** returns the contents of the ***Station ID*** field in a string.

fax username string

Specifies the user name displayed on fax cover sheets and fax document headers. This command corresponds to the ***Name*** edit field in the **Setup, Fax, User Info, User Info** dialog. **fetch** returns the contents of the ***Name*** field in a string.

fax voicenumber string

Specifies the voice number displayed on fax cover sheets fax document headers. This command corresponds to the ***Voice number*** edit field in **Setup, Fax, User Info, User Info**. **fetch** returns the contents of the ***Voice number*** field in a string.

fax xmitbaud long

Specifies the maximum baud rate used to send faxes. This command corresponds to the ***Maximum transmit rate*** field in the **Setup, System, Modem Connection, Fax Settings, General** dialog. **fetch** returns a long value reflecting the current maximum transmit rate. Possible values are 2400, 4800, 7200, 9600, 12000, or 14400.

FTP Set and Fetch Statements

The following **set**/**fetch** commands control FTP settings within Procomm Plus. They correspond to options in the **Internet, FTP Options** dialog in *Setup*.

ftp alarm OFF | ON

Determines whether Procomm Plus sounds the alarm when a transfer is complete. This command corresponds to the ***Sound alarm at end of transfer*** check box in the **Setup, Internet, FTP Options, Transfer Options** dialog. **fetch** will return a 0 if the check box is unchecked or a 1 if it is active.

ftp crashrecover OFF | ON

Determines whether to allow crash recovery on aborted downloads in *FTP* mode. This command corresponds to the ***Allow file transfer crash recovery*** check box in the **Setup, Internet, FTP Options, Transfer Options** dialog. **fetch** returns a 0 if the check box is unchecked, or a 1 if it is active.

ftp dblclick fileview | filexfer

Determines the action Procomm Plus takes when you double-click on a filename in FTP mode. This command corresponds to the ***Double Click on File Name*** radio group in the **Setup, Internet, FTP Options, General Options** dialog. **fetch** will return a 0 if ***To view file*** is selected or a 1 if ***To transfer file*** is selected.

ftp filter local | remote filename

Determines what file types appear in the directories on both the local and the remote machine. This command corresponds to the ***Default File Filters*** groupbox in the **Setup, Internet, FTP Options, Folder Options** dialog. **fetch** will return a string containing the filter for either the local or remote machine depending on which parameter you use.

ftp localdir pathname

Determines which directory is displayed by default when *FTP* mode is started. This command corresponds to the ***Initial local directory*** field in the **Setup, Internet, FTP Options, Folder Options** dialog. **fetch** will return a string containing the currently specified path.

ftp options index | string

Specifies the active FTP options set, corresponding to the ***Current FTP Options*** in **Setup, Internet, FTP Options**. Either the zero-based *index* or the name of an option set as it appears in the ***Current FTP Options*** list may be used. **fetch** returns the contents of the ***Current FTP Options*** field in a string.

If the **set dialentry access** command has been used to specify a ***FTP-***class *Connection Directory* entry, **set/fetch ftp options** only affects that entry.

ftp passivemode OFF | ON | CURRENT

Specifies whether to use passive mode. This command corresponds to the ***Use passive mode (for local firewall)*** check box in the **Setup, Internet, FTP Options, General Options** dialog. **fetch** returns a 0 is the check box is unchecked, or a 1 if it is active.

If the **set dialentry access** command has been used to specify a ***FTP-***class *Connection Directory* entry, **set/fetch ftp passivemode** only affects that entry. CURRENT specifies the current *Setup* setting, and is only valid for a *Connection Directory* entry.

ftp password string

Specifies the password to use with an anonymous logon. This command corresponds to the ***Anonymous logon password:*** field in the **Setup, Internet, FTP Options, General Options** dialog. **fetch** return the contents of the ***password*** in a string.

ftp prompt overwrite | longfilename OFF | ON

Specifies whether Procomm Plus will prompt you before converting long filenames or overwriting existing files. This command corresponds to the ***Confirm overwriting existing files*** and ***Prompt for long file name conversion*** check boxes in the **Setup, Internet, FTP Options, Folder Options** dialog. For each of the check boxes, **fetch** will return a 0 if the check box is unchecked, or a 1 if it is active.

ftp retainfiles OFF | ON

Determines whether to save files you've viewed in *FTP* mode. This command corresponds to the ***Retain viewed files*** check box in the **Setup, Internet, FTP Options, Folder Options** dialog. **fetch** returns a 0 if the check box is unchecked, or a 1 if it is active.

ftp viewgif OFF | ON

Determines whether Procomm Plus will automatically display GIF images as they are downloaded while in *FTP* mode. This command corresponds to the ***View GIF file while receiving*** check box in the **Setup, Internet, FTP Options, Transfer Options** dialog. **fetch** returns a 0 if the check box is unchecked or a 1 if it is active.

ftp xfermode BINARY | ASCII | L8 | CURRENT

Specifies the file transfer mode used by Procomm Plus in *FTP* mode. This command corresponds to the ***File transfer mode*** radio group in the **Setup, Internet, FTP Options, Transfer Options** dialog. **fetch** returns a 0 if ***Binary*** is selected, a 1 if ***ASCII*** is selected, or a 2 if ***L8*** is selected.

If the **set dialentry access** command has been used to specify a ***FTP-***class *Connection Directory* entry, **set/fetch ftp xfermode** only affects that entry. CURRENT specifies the current *Setup* setting, and is only valid for a *Connection Directory* entry.

INTERNET Set and Fetch Statements

The following **set/fetch** commands control TCP/IP settings within Procomm Plus. They correspond to options in the **Internet Connection** dialog in *Setup*.

With the exception of ***Internet Connection***, these commands only affect the current option set specified in the ***Current Internet Connection*** list in the **Setup, Internet, Internet Connection** dialog.

internet connection string

Specifies the active Internet connection. The name of a connection in the ***Current Internet Connection*** list in **Setup, Internet, Internet Connection** dialog must be used.

fetch returns the contents of the ***Current Internet Connection*** field in a string variable.

This command, unlike the user interface, does not automatically set the ***Mail Options*** and ***News Options*** to match what is specified for the ***Internet Connection***. If desired, you must manually set these options with the **set mail options** and **set news options** commands.

internet mail options index | string | CURRENT

Specifies the active *mail* options set, corresponding to the ***Current Mail Options*** in **Setup, Internet, Internet Connection**. Either the zero-based *index* or the name of an option set as it appears in the ***Current Mail Options*** list, or the keyword ***CURRENT*** may be used. **fetch** returns the contents of the ***Current Mail Options*** field in a string.

internet news options index | string | CURRENT

Specifies the active *news* options set, corresponding to the ***Current News Options*** in **Setup, Internet, Internet Connection**. Either the zero-based *index* or the name of an option set as it appears in the ***Current News Options*** list, or the keyword ***CURRENT*** may be used. **fetch** returns the contents of the ***Current News Options*** field in a string.

MAIL Set and Fetch Statements

mail address string

Specifies the Internet mail address used in the header of outgoing mail. This command corresponds to the ***Email Address*** field in the ***User Info*** panel of the **Setup, Internet, Internet Mail** dialog. **fetch** returns the contents of the ***Email Address*** field in a string.

mail forwardaddress OFF | ON | string

Specifies the Internet mail address used to send mail. This entry is only used if outgoing mail must be sent through a different server than incoming mail. This command corresponds to the ***Forward outgoing...*** field in the ***Mail Server*** panel of the **Setup, Internet, Internet Mail** dialog. **fetch** returns the OFF / ON setting in an integer or the server address in a string.

mail logonname string

Specifies the username to be used when connecting to the mail server. This command corresponds to the ***Logon Name*** field in the ***Mail Server*** panel of the **Setup, Internet, Internet Mail** dialog. **fetch** returns the contents of the ***Logon name*** field in a string.

mail options index | string

Specifies the active *Mail Client* options set, corresponding to the ***Current Mail Options*** in **Setup, Internet, Internet Mail**. Either the zero-based index or the name of an option set as it appears in the ***Current Mail Options*** list may be used. **fetch** returns the contents of the ***Current Mail Options*** field in a string.

mail password string

Specifies the password to be used when connecting to the mail server. This command corresponds to the ***Password*** field in the ***Mail Server*** panel of the **Setup, Internet, Internet Mail** dialog. **fetch** returns the contents of the ***Password*** field in a string.

mail replyaddress string

Specifies the Internet mail address on which you would like to receive replies. When the person you are sending mail to receives your message, this is the address they will see on the ***Reply to*** line of their mail reader. This command corresponds to the ***Reply Address*** field in the ***User Info*** panel of the **Setup, Internet, Internet Mail** dialog. **fetch** returns the contents of the ***Reply Address*** field in a string.

mail serveraddress string

Specifies the IP Address of the mail server from which you receive mail. This command corresponds to the ***Incoming mail server address*** field in the ***User Info*** panel of the **Setup, Internet, Internet Mail** dialog. **fetch** returns the contents of the ***Incoming mail server address*** field in a string.

mail signaturefile filespec

Specifies a text file that contains text to be appended to the end of all mail messages you send. This command corresponds to the ***Signature file*** field in the ***User Info*** panel of the **Setup, Internet, Internet Mail** dialog. **fetch** returns the contents of the ***Signature file*** field in a string.

mail username string

Specifies the username to be used when sending Internet mail. This command corresponds to the ***Full name*** field in the ***User Info*** panel of the **Setup, Internet, Internet Mail** dialog. **fetch** returns the contents of the ***Full name*** field in a string.

MODEM Set and Fetch Statements

The **set modem** statements affect the communication port connection and initialization, and the operation of your modem. They correspond to the options in the **Setup, System, Modem Connection** dialog. If you intend to change a setting associated with a connection other than the currently selected ***Current Modem/Connection***, you must first select it.

➥ *All **set modem** commands will fail if a file transfer is in progress.*

modem acceptcall DATA |FAX OFF | ON

Enables or disables auto-answer for ***Data*** and/or ***Fax*** calls. This command corresponds to the ***Accept x calls*** check boxes in the **Setup, System, Modem Connection** dialog. For both DATA and FAX, **fetch** returns 0 for OFF or 1 for ON.

This command will fail for all “direct connect” connections and **.dlc** connections like Telnet.

modem ansrings integer

Specifies the number of telephone rings after which Procomm Plus answers a data or fax call, with a valid range of 1 to 20. This command corresponds to the ***Procomm Plus should answer incoming calls after x rings*** field in **Setup, System, Modem Connection**. Using **fetch** returns the current number of rings as an integer.

Setting this value alone does not enable auto-answer. To enable auto-answer for data, fax, or both, you must use the **set modem autoanswer** command.

This command will fail for all “direct connect” connections and **.dlc** connections like Telnet.

modem autoanswer OFF | PW |OTHER

Enables or disables auto-answer for data and fax connections. This command will always return OFF for all “direct connect” connections and **.dlc** connections like Telnet. This command corresponds to the ***Incoming call handling...*** option button group in the **Setup, System, Modem Connection** dialog, where OFF specifies the ***Do not accept...*** option, PW specifies the ***Procomm Plus should answer...*** option, and OTHER specifies the ***Another program will answer...*** option. **fetch** returns 0 for OFF, 1 for PW, or 2 for OTHER.

modem connection index | string | CURRENT | FIRST

Specifies the active data modem connection. Either a zero-based *index*, or the name of a data modem connection in the ***Current Modem/Connection*** list in **Setup, System, Modem Connection** can be used. This command will fail if an attempt is made to set a *Connection Directory* entry to “direct connect-none”, “direct connect-Telnet”, or a modem/connection that has **incnctlist** set to OFF. This command will also fail if you specify a modem/connection that has **incnctlist** set to OFF. **fetch** returns the contents of the ***System Connection*** field in a string.

Please note that the index value for TAPI connections begins at 10, even though TAPI connections may be displayed at the top of the ***Current Modem/Connection*** list. Other connections have zero-based *indexes*. For example, “direct connect-none” is always *index* 0, followed in order by the other connections in the list.

The CURRENT and FIRST keywords are valid only for a specified *Connection Directory* entry. Additionally, the FIRST keyword is only valid when accessing a ***Fax*** class entry.

modem faxxmit OFF | ON

Specifies whether or not a modem can be used for sending faxes. This command corresponds to the ***Use this modem/connection to send faxes*** check box in the **Setup, System, Modem Connection** dialog. **fetch** returns 0 for OFF, 1 for ON.

This command will fail for all “direct connect” connections and **.dlc** connections like Telnet.

modem incntlist index | string OFF | ON

Disables or hides the connection name referred to by the index of string in the *Quick Select* connection list. This command fails when a **set modem incntlist** is attempted on the “direct connect-none” connection, and a fetch on that connection always returns the value ON. This command corresponds to the ***Make this connection available to Procomm Plus*** check box in the **Setup, System, Modem Connection** dialog.

NEWS Set and Fetch Statements

If *Setup* does not have a *serveraddress*, and *News* is started, the user will be presented with a dialog requiring that they specify a ***News server address***.

news address string

Specifies the Internet mail address being used when posting message from *News*. When the message you are posting is read, this is the address that will appear on the ***From*** line. This command corresponds to the ***Email Address*** field in the ***User Info*** panel of the **Setup, Internet, News** dialog. **fetch** returns the contents of the ***Email Address*** field in a string.

news checkgroups OFF | ON

Enables or disables the *News Client*’s automatic checking for new newsgroups. This command corresponds to the ***Check for new newsgroups...*** check box in the **Setup, Internet, News, Properties** dialog. **fetch** returns 1 if the check box is enabled, 0 if it is not.

news logonname string

Specifies the username to be used when posting messages. This command corresponds to the ***Logon Name*** field in the ***News Server*** panel of the **Setup, Internet, News** dialog. **fetch** returns the contents of the ***Logon name*** field in a string.

news organization string

Specifies the organization name being used when posting messages from *News*. When the message you are posting is read, this is the address that will appear on the ***Organization*** line. This command corresponds to the ***Organization*** field in the ***User Info*** panel of the **Setup, Internet, News** dialog. **fetch** returns the contents of the ***Organization*** field in a string.

news options index | string

Specifies the active *News Reader* options set, corresponding to the ***Current News Options*** in **Setup, Internet, News**. Either the zero-based index or the name of an option set as it appears in the ***Current News Options*** list can be used. **fetch** returns the contents of the ***Current News Options*** field in a string.

news password string

Specifies the password to be used when posting messages. This command corresponds to the ***Password*** field in the ***News Server*** panel of the **Setup, Internet, News** dialog. **fetch** returns the contents of the ***Password*** field in a string.

news promptheaders OFF | ON

Enables or disables Procomm Plus's prompting when a large number of headers is available for retrieving. This command corresponds to the ***Prompt if available headers...*** check box in the **Setup, Internet, News, Properties** dialog. **fetch** returns 1 if the check box is enabled, 0 if it is not.

news serveraddress string

Specifies the address of your news server. This command corresponds to the ***News server address*** field in the ***News Server*** panel of the **Setup, Internet, News** dialog. **fetch** returns the contents of the ***News server address*** field in a string.

Changing the server address when Procomm Plus is not in *News* mode will result in a dialog being displayed on a subsequent switch to *News* mode. The dialog will not be displayed if Procomm Plus is already in *News* mode.

news signaturefile filespec

Specifies a text file that contains text to be appended to the end of all messages posted from *News*. This command corresponds to the ***Signature file*** field in the ***User Info*** panel of the **Setup, Internet, News** dialog. **fetch** returns the contents of the ***Signature file*** field in a string.

news username string

Specifies the name you wish to use when posting news messages. This command corresponds to the ***Name*** field in the ***User Info*** panel of the **Setup, Internet, News** dialog. **fetch** returns the contents of the ***Name*** field in a string.

PORT Set and Fetch Statements

The following **set**/**fetch** statements control the communication port for the current connection.

*FAILURE is set if any **set port** command is used to set/fetch the settings of an unopened TAPI connection. Also, all **set port** commands fail with the "direct connect-none" connection and **.dlc** connections like Telnet.*

*For TAPI connections, the **set port** commands only affect the currently selected connection in the **System** panel of Setup. They cannot change or access settings of a connection that is referenced by a Connection Directory entry. The settings will not be saved as new settings for a TAPI connection.*

*All **set port** commands will fail if a file transfer or dialing attempt is in progress.*

port baudrate baudrate

Specifies the baud rate at which Procomm Plus talks to the current data modem. If the DEFAULT keyword is specified, the string ***Modem Default*** is used. **fetch** returns the current baud rate as a long value.

port databits integer

Specifies the length of a byte sent or received through the current port, either 7 or 8 bits. **fetch** returns 7 for 7 data bits or 8 for 8 data bits.

port dropdtr OFF | ON

Determines whether Procomm Plus will drop the RS-232 DTR signal to disconnect. **fetch** returns 0 for OFF or 1 for ON. This command always fails for TAPI connections.

port hardflow OFF | ON

Enables or disables hardware flow control, which is sometimes referred to as RTS/CTS flow control. fetch returns 0 for OFF or 1 for ON.

port parity NONE | ODD | EVEN | MARK | SPACE

Specifies the parity for the current communication port. **fetch** returns 0 for NONE, 1 for ODD, 2 for EVEN, 3 for MARK or 4 for SPACE.

port softflow OFF | ON

Enables or disable software flow control, which is sometimes referred to as XON/XOFF flow control. **fetch** returns 0 for OFF or 1 for ON.

port stopbits integer

Specifies the number of bits used to signify the end of a byte sent or received through the current port. **fetch** returns 1 for 1 stop bit or 2 for 2 stop bits.

PRINT Set and Fetch Statements

The following **set**/**fetch** statements control the operation of the printer.

*Changes made using the **set print** command family will take effect upon the execution of the next **printer open** command. If the printer is already open, changes made using the **set print** command family will not alter the current printer output.*

print device string

Specifies the print device used by Procomm Plus when the printer is enabled. Only the name of the device is required; port information should not be included in the device name even though the device name appears this way in many of the printer lists in Windows. **fetch** returns the name of the current print device in a string.

print fontname string

Specifies the font used by Procomm Plus when printing. *String* should specify the font name just as it appears in the ***Font*** list box in the **Fonts** dialog. **fetch** returns the name of the current printer font in a string.

print fontsize integer

Specifies the size of the font used by Procomm Plus when printing. **fetch** returns the size of the current printer font in an integer.

print footer string

Specifies the text to print at the bottom of printed pages. The maximum character length of this text is 65 characters. **fetch** returns the contents of the ***Footer*** field for the current print device in a string.

print header string

Specifies the text to print at the top of printed pages. The maximum character length of this text is 65 characters. **fetch** returns the contents of the ***Header*** field for the current print device in a string.

print margins left right top bottom

Specifies the margins for printed pages. All values are decimal point values or floats, measured in inches. **fetch** returns four float values, reflecting the values of the four margins.

print orientation PORTRAIT | LANDSCAPE

Determines the orientation of printed pages. When set to PORTRAIT, pages are taller than they are wide. When set to LANDSCAPE, pages are wider than they are tall. This command corresponds to the ***Orientation*** radio group in the **Print Setup** dialog. **fetch** returns 1 for PORTRAIT or 2 for LANDSCAPE.

TELNET Set and Fetch Statements

The following **set**/**fetch** commands control Telnet settings within Procomm Plus. They correspond to options in the **Telnet** dialog in *Setup*.

telnet autologon OFF | ON

Determines whether Procomm Plus allows automatic Telnet logons. This command corresponds to the ***Allow automatic Telnet logon*** check box in the **Setup, Internet, Telnet** dialog. **fetch** returns a 0 if the check box is not active, or a 1 if it is checked.

telnet baudrate baudrate | DISABLED

Determines whether Procomm Plus will force the baud rate to a certain speed or not. The second *baudrate* is a reference to a *long* variable specifying the port speed to use. This command corresponds to the ***Force speed to:*** field in the **Setup, Internet, Telnet** dialog. **fetch** returns an index.

telnet binarymode OFF | ON

Determines whether the session is in binary mode. This command corresponds to the ***Set session to binary mode*** check box in the **Setup, Internet, Telnet** dialog. **fetch** returns a 0 if the check box is not active, or a 1 if it is checked.

telnet negotiation OFF | ON

Specifies whether to disable all types of Telnet negotiation. This command corresponds to the ***Disable all Telnet negotiation:*** check box in the **Setup, Internet, Telnet** dialog. **fetch** returns a 0 if the check box is not active or a 1 if it is checked. If this check box is enabled, it will override the other two ***Negotiate...*** fields.

telnet options index | string

Specifies the active set of Telnet options corresponding to the ***Current Telnet Options:*** list in the **Setup, Internet, Telnet** dialog. Either the zero-based *index* or the name of an option set as it appears in the ***Current Telnet Options*** list may be used. **fetch** returns the contents of the ***Current Telnet Options*** field in a string.

If the **set dialentry access** command has been used to specify a ***Telnet***-class *Connection Directory* entry, **set/fetch telnet options** only affects that entry.

telnet terminal size OFF | ON

Determines whether Procomm Plus will negotiate terminal size. This command corresponds to the ***Negotiate terminal size*** check box in the **Setup, Internet, Telnet** dialog. **fetch** returns a 0 if the check box is not active or a 1 if it is checked.

telnet terminal type OFF | ON

Determines whether Procomm Plus will negotiate the terminal emulation to use. This command corresponds to the ***Negotiate terminal type*** check box in the **Setup, Internet, Telnet** dialog. **fetch** returns a 0 if the check box is not active or a 1 if it is checked.

telnet xwindow string

Specifies the string to use when negotiating with an Xwindow host. This command corresponds to the ***Set Xwindow value to:*** field in the **Setup, Internet, Telnet** dialog. **fetch** returns the string currently specified, or a null string if none is specified.

TERMINAL Set and Fetch Statements

The following **set**/**fetch** commands control emulation settings within Procomm Plus. They correspond to options in the **Terminal Options**, **Terminal Fonts** and **Terminal Colors** dialogs in *Setup*.

terminal autosize WINDOW | FONT | OFF

Determines whether the size of the *Terminal* window is adjusted based on the size of the Procomm Plus main window, or based on the size of the current *Terminal* font. This command corresponds to the ***Terminal autosize*** field in the **Setup, Data, Terminal Fonts** dialog. **fetch** returns 0 for WINDOW, 1 for FONT or 2 for OFF.

terminal backspace NONDEST | DEST

Determines whether Procomm Plus erases or preserves the character under the cursor when a backspace is received. This command corresponds to the ***Destructive backspace*** check box in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for NONDEST or 1 for DEST.

terminal blinkrate integer

Specifies the pace at which blinking characters blink in the *Terminal window*. Values range from 0 to 9, slowest to fastest. This command corresponds to the ***Blink rate*** horizontal scroll bar in **Setup, Data, Terminal Colors**. **fetch** returns an integer reflecting the position of the thumb on the ***Blink rate*** horizontal scroll bar.

terminal blockcursor OFF | ON

Determines the size of the cursor in the *Terminal* window. If **set** to OFF, the cursor is a line. If ON, the cursor is a block. This command corresponds to the ***Block cursor*** check box in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

terminal colors index | string

Specifies the ***Current Terminal Colors*** set. This command corresponds to the ***Current Terminal Colors*** list in the **Setup, Data, Terminal Colors** dialog. The zero-based *index* or the name of an entry in the ***Current Terminal Color***s list can be specified. **fetch** returns the name of the ***Current Terminal Colors*** set in a string.

The **set dialentry access** command can be used to target a ***Data-*** or ***Telnet***-class *Connection Directory* entry for the **set terminal colors** command.

terminal columns integer

Determines whether Procomm Plus uses an 80-column or a 132-column *Terminal* display. This command corresponds to the ***Terminal size X Columns*** list in the **Setup, Data, Terminal Options** dialog. **fetch** returns 80 for 80 columns or 132 for 132 columns.

Only the ADM31, TTY, TVI-955, DEC family (VT-xxx), AT&T family, and WYSE family terminals support 132 columns.

terminal enquiry OFF | ON | CISB

Determines Procomm Plus's response to a received ENQ, ASCII 5, character. When **set** to *ON*, the string specified by the **set terminal enquirystr** command is sent. When **set** to CISB, the ENQ character initiates a CompuServe B+ file transfer. This command corresponds to the ***Enquiry type*** field in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for OFF, 1 for ON or 2 for CISB.

terminal enquirystr string

Specifies the string to be sent in response to a received ENQ, ASCII 5, character when ***Terminal enquiry*** is set ON. This command corresponds to the ***Enquiry response*** edit field in the **Setup, Data, Terminal Options** dialog. **fetch** returns the current contents of the field in a string.

terminal font index | string

Specifies the ***Current Terminal Font***. This command corresponds to the ***Current Terminal Font*** list in the **Setup, Data, Terminal Fonts** dialog. The zero-based *index* or the name of an entry in the ***Current Terminal Font*** list can be specified. **fetch** returns the name of the ***Current Terminal Font*** list entry in a string.

The **set dialentry access** command can be used to target a ***Data-*** or ***Telnet***-class *Connection Directory* entry for the **set terminal font** command.

terminal fontname string

Specifies the font used to display information in the *Terminal* window. This command corresponds to the ***Font*** list box in the **Setup, Data, Terminal Fonts** dialog. Specify the name of the font just as it appears in the ***Font*** list box. **fetch** returns the name of the current font in a string.

terminal fontsize integer

Specifies the size of the current font. This command corresponds to the ***Size*** list box in the **Setup, Data, Terminal Fonts** dialog. **fetch** returns an integer reflecting the size of the current *Terminal* font.

terminal frame OFF | ON

Determines whether Procomm Plus displays a frame around the *Terminal* window. This command corresponds to the ***Display terminal frame*** check box in the **Setup, Data, Window Colors** dialog. **fetch** returns 0 for OFF or 1 for ON.

terminal hostprint ENABLE | DISABLE

Specifies whether Procomm Plus responds to print commands sent by a host system. With **set terminal hostprint** DISABLE, Procomm Plus ignores host print commands. This command corresponds to the ***Disable host printing*** check box in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for ENABLE or 1 for DISABLE.

terminal keyboardfile terminal | string | filename

Specifies the keyboard mapping file to use with the current Terminal emulation. The filename extension, if specified, must be **.kbd**. If no extension is specified, the string must match either a base terminal name, such as ANSIBBS, or a name that was imported from a **.dlt** file.

You can also specify a *Terminal* keyword name such as ANSIBBS, WYSE50, or TVI950. This command corresponds to the ***Terminal keyboard file*** located in the **Setup, Data, Terminal Options** dialog. **fetch** returns the name of the keyboard file for the current emulation in a string.

terminal linewrap OFF | ON

Determines whether Procomm Plus wraps incoming characters to the following row when the end of a row is reached. This command corresponds to the ***Line wrap*** check box in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

terminal pattern string

Specifies the graphic pattern surrounding the *Terminal* window. This command corresponds to the ***Patterns*** list in the **Setup, Data, Window Colors** dialog. **fetch** returns the name of the current pattern in a string.

terminal rawprint OFF | ON

Determines whether Procomm Plus uses raw print mode for print captures. This command corresponds to the ***Raw print mode*** check box in the **Setup, Data, Terminal Options** dialog.

terminal rows integer

Determines the number of text rows in the *Terminal* window for the current emulation. Any value from 2 to 99 is acceptable, but some emulations may not be able to address cursor locations beyond row 25. This command corresponds to ***Terminal Size X Rows*** list in the **Setup, Data, Terminal Options** dialog. **fetch** returns the number of rows available for the current terminal.

terminal rxcr CR | CR_LF

Determines how Procomm Plus reacts when a Carriage Return, ASCII 13, is received. With CR_LF, Carriage Returns are translated to Carriage Return/Linefeed, ASCII 13/ASCII 10 pairs. With CR, Carriage Returns are not translated. This command corresponds to the ***Incoming CR to CR/LF*** check box in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for CR or 1 for CR_LF.

terminal sbpages integer

Determines the number of pages stored by the *Scrollback Buffer*. Valid values range from 0 to 1300. This command corresponds to the ***Scrollback pages:*** field in the **Setup, Data, Terminal Options** dialog. **fetch** returns the current number of *Scrollback* pages.

terminal scroll OFF | ON

Determines the action taken when a character is written to the very last row and column in the *Terminal* display. This command corresponds to the ***Screen scroll*** check box in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

set terminal scroll has no affect when the ***Line wrap*** check box in **Setup, Terminal Options** is unchecked.

terminal scrollmethod REPAINT | NORMAL

Determines how the *Terminal* display is repainted when information is scrolled. This command corresponds to the ***Terminal scroll method***: field in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for REPAINT or 1 for NORMAL.

terminal stripbit8 OFF | ON

Determines whether the 8th or high bit is stripped from received data before it is displayed. Setting this value ON prevents emulations from displaying characters with ASCII values greater than 127, the DEL character. This command corresponds to the ***Strip bit 8*** check box in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

terminal tabstops integer

Specifies the positions for fixed tab stops in the current emulation. The default is 8 characters; values range from 0 to 254. This command corresponds to the ***Tab stops every X positions*** field in the **Setup, Data, Terminal Options** dialog. **fetch** returns an integer equal to the tab spacing for the current terminal.

terminal type terminal | index | string

Specifies the current Terminal emulation. This command corresponds to the ***Current Terminal:*** list in the **Setup, Data, Terminal Options** dialog. An emulation keyword, the name of an emulation shown in the ***Current Terminal*** list, or the zero-based *index* of a listed emulation may be specified. **fetch** returns a string containing the name of the ***Current Terminal***.

For a complete list of keywords and indices for emulations, see "*Emulation Names and Indices*" on page 60.

The **set dialentry access** command can be used to target a ***Data-*** or ***Telnet***-class *Connection Directory* entry for the **set terminal type** command.

terminal update INCREMENTAL | BLOCK | FAST

Determines how the *Terminal window* is updated when data is received. This command corresponds to the ***Terminal update:*** field in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for INCREMENTAL, 1 for BLOCK or 2 for FAST.

terminal viewcursor OFF | ON

Determines whether the emulation cursor is kept in view while the *Terminal window* is being updated. This command corresponds to the ***Keep cursor in view*** check box in the **Setup, Data, Terminal Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

Advanced Emulation Commands Set and Fetch Statements

The following advanced **set**/**fetch** emulation statements are not available for all *Terminal* types; a base type must be specified. For example, to disable the status line for the TVI950 emulation, use **set tvi950 statusline OFF**. To turn on block mode for the WYSE50 emulation, use **set wyse50 blockmode ON**.

Each group of **set**/**fetch** emulation statements is preceded by a list of terminal emulations for which the command group is valid.

For all of the advanced terminal emulation **set/fetch** commands, the specified *emulation* argument is a *base type* and is required to validate the syntax at compile time. At run-time, however, if *Setup*'s currently selected emulation has the same base type, the **set/fetch** statement will only affect the currently selected terminal. If not, the base terminal definition will be effected.

ADM31, IBM3270, TVI950, TVI955, WYSE50, WYSE60, WYSE100

emulation statusattr NORMAL | REVERSE | UNDERLINE

Determines how the terminal status line is displayed for the specified emulation. This command corresponds to the ***Terminal Status Line*** options in the **Setup, Data, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for NORMAL, 1 for REVERSE, or 2 for UNDERLINE.

emulation statusline OFF | ON

Enables or disables the terminal status line for the specified emulation. This command corresponds to the ***Terminal Status Line Display*** check box in the **Setup, Data, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for OFF or 1 for ON.

ADM31, ESPRIT3, IBM3270, TVI910, TVI912, TVI920, TVI925, TVI950, TVI955, WYSE50, WYSE60, WYSE100

emulation protectattr DIM | REVERSE | UNDERLINE OFF | ON

Determines how protected text is displayed in the Terminal window for the specified emulation. This command corresponds to the ***Protected Attribute*** options in the **Setup, Data, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for OFF or 1 for ON.

ADM31, IBM3101, IBM3161, WYSE50, WYSE60

emulation blockmode OFF | ON

Enables or disables block mode for the specified emulation. This command corresponds to the ***Block mode*** check box in the **Setup, Data, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for OFF or 1 for ON.

ADM31, WYSE50, WYSE60

emulation endsequence CRLF_ETX | US_CR

Determines the characters which signify the end of a line or block for the specified emulation. If CRLF_ETX is specified, Carriage Return/Linefeed, ASCII 13/ASCII 10 pairs and end transmission characters, ASCII 3, mark the end of a line or block. If US_CR is used, US characters, ASCII 31 and Carriage Returns mark the end of a line or block. This command corresponds to the ***End of line/block sequence*** field in the **Setup, Data, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for CRLF_ETX or 1 for US_CR.

ANSIBBS, IBMPC

emulation declinewrap OFF | ON

Determines how the ANSI and IBMPC emulations wrap lines that extend beyond the end of an emulation row. If ON, the emulations use the DEC Terminal method, which means Carriage Returns and the first Linefeed received are ignored when incoming data is wrapped to the following line. This command corresponds to the ***Use DEC line wrap mode*** check box in the **Setup, Data, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for OFF or 1 for ON.

emulation escapem MUSIC | DELLINE

Determines how the ANSI and IBMPC emulations respond to the ANSI Escape-M (^[M) sequence. If DELLINE is specified, the sequence is treated as a line delete command. If MUSIC is specified, the emulation "eats" or ignores ANSI music sequences. This command corresponds to the ***Treat Music sequence as Delete Line*** check box in the **Setup, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for MUSIC or 1 for DELLINE.

TVI922, VT220, VT320

emulation bit8mode OFF | ON

Determines how the listed emulations respond to the 8 bit DEC sequences. If ON, the emulation expects command sequences to be prefixed by the cent character, ASCII 155, rather than ESC [, ASCII 27, and ASCII 91. This command corresponds to the ***8 bit mode*** check box in the **Setup, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for OFF or 1 for ON.

emulation charset DEC | ISO

Determines the emulation's startup character set. If DEC is specified, the DEC multi-national character set is used on startup. If ISO is specified, the ISO Latin-1 character set is used. This command corresponds to the ***Extended Character Set*** check box in the **Setup, Terminal Options, Advanced** Terminal Setup dialog for the emulations listed. **fetch** returns 0 for DEC or 1 for ISO.

emulation cursorkeyapp OFF | ON

Determines whether emulation cursor keys generate normal DEC emulation sequences or special application sequences. This command corresponds to the ***Cursor Key Application Mode*** check box in the **Setup, Terminal Options, Advanced** Terminal Setup dialog for the emulations listed. **fetch** returns 0 for OFF or 1 for ON.

emulation keypadapp OFF | ON

Determines whether emulation numeric keypad keys generate normal DEC emulation sequences or special application sequences. This command corresponds to the ***Keypad Application Mode*** check box in the **Setup, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for OFF or 1 for ON.

IBM3101, IBM 3161

emulation entercrlf OFF | ON

Specifies whether an **Enter** keypress generates the characters mapped to the return key in the keyboard mapping utility or a Carriage Return and Linefeed, ASCII 13 and ASCII 10 pair. This command corresponds to the ***Enter key sends CR/LF*** check box in the **Setup, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for OFF or 1 for ON.

emulation nullsuppress OFF | ON

Determines whether processing of the NULL character, ASCII 0, is suppressed for the specified emulation. This command corresponds to the ***Null suppression*** check box in the **Setup, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for OFF or 1 for ON.

emulation turnchar CR | EOT | ETX | XOFF

Determines the end of a line or block character for the specified emulation. If set to CR, a Carriage Return, ASCII 13, is appended to the end of a line or block when the send key is pressed. If set to EOT, an end of text character, ASCII 4, is appended. If set to ETX, an end transmission character, ASCII 3, is appended. If set to XOFF, a pause transmission character, ASCII 19, is appended. This command corresponds to the ***Turn-around character*** field in the **Setup, Terminal Options, Advanced** Terminal Setup dialog for the listed emulations. **fetch** returns 0 for CR, 1 for EOT, 2 for ETX or 3 for XOFF.

RIP

rip filetype BITMAP | TEXT

Controls the operation of ***Edit Screen to Clipboard*** and ***Edit Screen to File*** menu items. When set to *BITMAP* the entire current terminal screen will be copied as a bitmap. If *TEXT* is selected, the text portions of the screen will be copied as text. This command corresponds to the ***Screen to File/Clipboard options*** radio group in the **Ripscrip Advanced Setup** dialog.

rip font index

Specifies which font Procomm Plus will display when using the RIP emulation. *Index* is an integer from 0 to 4 which corresponds to the option buttons in the ***Initial Text Window Font*** radio group in the **Setup, Data, Terminal Options, RIPscrip Advanced Setup** dialog. **fetch** will return the currently selected font.

rip hotkeys OFF | ON

Specifies the initial setting for button hotkeys. This command corresponds to the ***Enable button Hotkeys*** check box in the **Ripscrip Advanced Setup** dialog. Button hotkeys must still be set by the host, and the host may enable or disable this item at any time. **fetch** will return a 0 if it is disabled, or a 1 if it is enabled.

rip iconpath pathname

Specifies the path Procomm Plus will use to locate the RIP icons. This command corresponds to the ***RIP Icons path*** field in the **Setup, Data, Terminal Options, RIPscrip Advanced Setup** dialog. **fetch** will return the current path.

rip print BITMAP | TEXT

Controls printing method used in the RIP emulation. This command corresponds to the ***Screen to Printer printing options*** radio group in the **Ripscrip Advanced Setup** dialog. When set to *BITMAP* the entire current terminal screen will be printed as a graphical image. If *TEXT* is selected, only the text portions of the screen will be sent to the printer.

rip security OFF | ON

Enables or disables Data Security for the RIP emulation. This command corresponds to the ***Enable Data Security*** check box in the **Setup, Data, Terminal Options, RIPscrip Advanced Setup** dialog. **fetch** will return a 0 if it is disabled, or a 1 if it is enabled.

rip tabkey OFF | ON

Specifies the action of the <Tab> key within the RIP emulation. This command corresponds to the ***Tab key controls button focus*** check box in the **Ripscrip Advanced Setup** dialog. When set to OFF, the <Tab> key sends a tab character (ASCII 0x9). When set to ON, the tab key moves the on screen highlight from control to control. **fetch** will return a 0 if it is OFF, or a 1 if it is ON.

rip variable string string | DELETE

Specifies additional text variables for the Text Variable Database. The first *string* specifies the name for the Text Variable and the second *string* specifies the contents. This command corresponds to the ***Name:*** and ***Contents:*** fields in the **Setup, Data, Terminal Options, RIPscrip Advanced Setup, Text Variable Database, Create Text Variable** dialog. DELETE will remove a variable from the database. **fetch** will return the contents of the variable.

PROTOCOLS

For all of the protocol **set/fetch** commands, the specified protocol argument is a *base type* identifier required to validate the syntax at compile time. At run-time, if the currently specified protocol in *Setup* has the same base type, only the currently specified protocol is effected. If the currently specified protocol is of a different base type, the command will affect the base protocol definition.

ASCII Set and Fetch Statements

The following **set**/**fetch** statements apply to only the ASCII transfer protocol. These commands correspond to the options in **Setup, Data, Transfer Protocol**, with the ***Current Protocol*** field set to ***ASCII***.

ascii abortdnld KEEP | DELETE

Determines whether files aborted during an ASCII file download are kept or deleted. This command corresponds to the ***Delete aborted downloads*** check box. **fetch** returns 0 for KEEP or 1 for DELETE.

ascii blankexpand OFF | ON

Determines whether blank lines are expanded to spaces during ASCII file uploads. This command corresponds to the ***Expand blank lines*** check box. **fetch** returns 0 for OFF or 1 for ON.

ascii charpace integer

Specifies the length of a pause after a character is transmitted from a file during an ASCII file upload. This command corresponds to the ***Millisecond delay between characters*** field. **fetch** returns the current character pacing value, in the range of 0 to 9999.

ascii dnld cr STRIP | CR_LF | CR

Determines how a Carriage Return, ASCII 13, is interpreted in an ASCII file download. If STRIP is specified, incoming Carriage Returns are removed before the data is saved. If CR_LF is specified, incoming Carriage Returns are translated into Carriage Return/Linefeed, ASCII 13/ASCII 10 pairs before the data is saved. If CR is specified, no translation is performed. This command corresponds to the ***CR options - download*** list. **fetch** returns 0 for STRIP, 1 for CR_LF or 2 for CR.

ascii dnld display OFF | ON

Determines whether incoming data is displayed as it is received during an ASCII file download. This command corresponds to the ***Display text*** check box. **fetch** returns 0 for OFF or 1 for ON.

ascii dnld lf STRIP | CR_LF | LF

Determines how a Linefeed, ASCII 10, is interpreted during an ASCII file download. If STRIP is specified, incoming Linefeeds are removed before the data is saved. If CR_LF is specified, incoming Linefeeds are translated into Carriage Return/Linefeed, ASCII 13/ASCII 10 pairs before the data is saved. If LF is specified, no translation is performed. This command corresponds to the ***LF options - download*** field. **fetch** returns 0 for STRIP, 1 for CR_LF or 2 for LF.

ascii linepace integer

Specifies length of a pause after a line is sent during an ASCII file upload. This command corresponds to the ***Millisecond delay between lines*** field. **fetch** returns the current line pacing, in the range of 0 to 9999.

ascii overwrite OFF | ON

Determines whether a file is overwritten when another file using the same name is received with the ASCII file transfer protocol. This command corresponds to the ***Overwrite existing file*** check box. **fetch** returns 0 for OFF or 1 for ON.

ascii pacechar character

Specifies the character used to pace ASCII file uploads. This command corresponds to the ***Use x for Line pace character*** field. **fetch** returns the ASCII value of the upload pacing character, ranging from 0 to 255.

ascii stripbit8 OFF | ON

Determines whether the 8th or high bit is stripped from received data before it is saved to a file during an ASCII file download. This command corresponds to the ***Strip bit 8*** check box. **fetch** returns 0 for OFF or 1 for ON.

ascii tabexpand OFF | ON

Determines whether Tab characters, ASCII 9, are expanded into spaces as they are transmitted during an ASCII file upload. This command corresponds to the ***Expand tabs*** check box. **fetch** returns 0 for OFF or 1 for ON.

ascii tabstops integer

Determines the number of spaces that Tab characters are expanded to during an ASCII file upload. This command corresponds to the ***Expand tabs x spaces*** field. **fetch** returns the integer value of this field, ranging from 0 to 255.

ascii timeout integer

Specifies the number of seconds that Procomm Plus waits for incoming information during an ASCII file download before closing the transfer as completed. This command corresponds to the ***Seconds until download time-out*** field. **fetch** returns the current download time-out value.

ascii upld cr STRIP | CR_LF | CR

Determines how Carriage Returns, ASCII 13, are handled during an ASCII file upload. If STRIP is specified, Carriage Returns are not transmitted. CR_LF causes Carriage Returns to be translated into Carriage Return/Linefeed, ASCII 13/ASCII 10 pairs as they are sent. If CR is specified, Carriage Returns are transmitted without translation. This command corresponds to the ***CR options - upload*** field. **fetch** returns 0 for STRIP, 1 for CR_LF, or 2 for CR.

ascii upld display OFF | ON

Determines whether a file is displayed as it is sent during an ASCII file upload. This command corresponds to the ***Display text*** check box. **fetch** returns 0 for OFF or 1 for ON.

ascii upld lf STRIP | CR_LF | LF

Determines how a Linefeed, ASCII 10, is transmitted in an ASCII file upload. If STRIP is specified, Linefeeds are not sent. CR_LF causes Linefeeds to be translated as Carriage Return, ASCII 13, and Linefeed pairs as they are transmitted. If LF is specified, no translation is performed. This command corresponds to the ***LF options - upload*** field. **fetch** returns 0 for STRIP, 1 for CR_LF, or 2 for CR.

ascii usepacechar OFF| ON

Determines whether ASCII file uploads are paced using the pace character specified by **set ascii pacechar**. This command corresponds to the ***Use X for Line pace character*** check box. **fetch** returns 0 for NO or 1 for YES.

CISB Set and Fetch Statements

The following **set**/**fetch** commands correspond to the options in **Setup, Data, Transfer Protocol**, with the ***Current Protocol*** field set to ***CIS-B+***.

cisb abortdnld KEEP | DELETE

Determines whether files aborted during a CompuServe B+ file download are kept or deleted. This command corresponds to the ***Delete aborted downloads*** check box. **fetch** returns 0 for KEEP or 1 for DELETE.

cisb overwrite RESUME | ALWAYS | RENAME

Determines how filename collisions are handled in the CompuServe B+ file transfer protocol. If RESUME is specified, the download is treated as a resumption of a previous download, meaning that the remainder of the downloaded file downloaded is appended to the existing file. If ALWAYS is specified, the existing file is always overwritten by the file being downloaded. If RENAME is specified, the downloaded file is renamed, preserving the existing file. This command corresponds to the ***Existing File Action*** check box. **fetch** returns 0 for RESUME, 1 for ALWAYS, or 2 for RENAME.

IND$FILE Set and Fetch Statements

The following **set** and **fetch** commands correspond to the options found in **Setup, Data, Transfer Protocol**, with the ***Current Protocol*** field set to ***IND$FILE***.

ind$file abortdnld KEEP | DELETE

Determines whether aborted Ind$file downloads are kept or deleted. This command corresponds to the ***Delete aborted downloads*** check box. **fetch** returns 0 for KEEP or 1 for DELETE.

ind$file asciixlat OFF | ON

Determines whether EBCDIC characters are translated into ASCII characters when a text file is downloaded using the Ind$file protocol. This command corresponds to the ***ASCII translate*** check box. **fetch** returns 0 for OFF or 1 for ON.

ind$file converter index string

Specifies protocol converter options for the Ind$file transfer protocol. *Index* corresponds to edit fields listed in the **Protocol Converter Setup** dialog (available within **Setup, Data, Transfer Protocol,** by clicking on ***Protocol Converter...***) ranging from 0 to 9 for ***Enter, Reset, Refresh, Clear, PF1, PF2, Erase input, Erase EOF, File transfer on*** and ***File Transfer off***, respectively. **fetch** returns the contents of the field specified by *index* in a string.

For example, if you want to change the sequence that represents ***Enter***, use **set ind$file converter** 0 *string*, where *string* is the desired value. It is important to note that these settings are only available with certain ***Environment:*** settings.

ind$file crlfxlat OFF | ON

Determines whether end-of-line characters are translated based upon the capabilities of the receiving system in an Ind$file file transfer. This command corresponds to the ***CR-LF translate*** check box. **fetch** returns 0 for OFF or 1 for ON.

ind$file display OFF | ON

Determines whether a text file is displayed as it is uploaded with the Ind$file protocol. This command corresponds to the ***Display data during transfer*** check box**. fetch** returns 0 for OFF or 1 for ON.

ind$file environment index

Specifies the protocol converter and terminal emulation combination used when transferring files. *Index* is a zero-based integer corresponding to an item listed in the ***Environment***: list. **fetch** returns an integer equal to the current item's *index* value.

ind$file host VM_CMS | MVS_TSO | OTHER

Specifies the host's operating system. This command corresponds to the ***Host OS***: list. **fetch** returns 0 for VM_CMS, 1 for MVS_TSO, or 2 for OTHER.

ind$file lrecl integer

Specifies the logical record length for a file uploaded with the Ind$file protocol. This command corresponds to the ***LRECL*** edit field. Valid values range from 1 to 65535, treated as an unsigned integer. **fetch** returns an unsigned integer equal to the current logical record length. This field has no effect if ***Use LRECL:*** is not enabled.

ind$file overwrite OFF | ON

Determines how filename collisions are handled by the IND$FILE protocol. This command corresponds to the ***Overwrite existing file*** check box. **fetch** returns 0 for OFF or 1 for ON.

ind$file pacelines integer

Specifies the number of lines Procomm Plus will receive before pausing during an Ind$file file upload. This command corresponds to the ***Pace uploads: X Lines*** edit field. **fetch** returns an integer equal to the current number of pace lines. If ***Pace uploads:*** is not enabled, this command will have no effect.

ind$file recfm FIXED | VARIABLE

Specifies the record format used for storing files uploaded with the Ind$file protocol. This command corresponds to the ***Use RECFM*** edit field. **fetch** returns 0 for FIXED or 1 for VARIABLE. This field has no effect if ***Use RECFM:*** is not enabled.

ind$file recvcmd string

Specifies the command sent to the host system to initiate an Ind$file download. This command corresponds to the ***Receive command*** edit field. **fetch** returns the contents of the field in a string.

ind$file sendcmd string

Specifies the command sent to the host system to initiate an Ind$file upload. This command corresponds to the ***Send command*** edit field. **fetch** returns the contents of the field in a string.

ind$file timing RELAXED | NORMAL | TIGHT

Determines the Ind$file protocol's sensitivity to timing problems. This command corresponds to the ***Timing:*** list. **fetch** returns 0 for RELAXED, 1 for NORMAL, or 2 for TIGHT.

ind$file upldpace OFF | ON

Determines whether the Ind$file protocol pauses after sending lines from a text file. If ON, the upload pauses after sending the number of lines specified in the ***Pace uploads: X Lines*** edit field before resuming the transfer. This command corresponds to the ***Pace uploads: X Lines*** check box. **fetch** returns 0 for OFF or 1 for ON.

ind$file uselrecl OFF | ON

Determines whether files uploaded with the Ind$file protocol are sent using the logical record length specified in the ***Use LRECL: edit*** field. This command corresponds to the ***Use LRECL:*** check box. **fetch** returns 0 for OFF or 1 for ON.

ind$file userecfm OFF | ON

Determines whether files uploaded with the Ind$file protocol are sent using a specific record format. If ON, Ind$file uses the record format specified in the ***Use RECFM:*** edit field. This command corresponds to the ***Use RECFM:*** check box. **fetch** returns 0 for OFF or 1 for ON.

KERMIT Set and Fetch Statements

The following **set**/**fetch** statements control the operation of the Kermit protocol. They correspond to options in **Setup, Data, Transfer Protocol**, with the ***Current Protocol*** field set to ***KERMIT***.

kermit abortdnld KEEP | DELETE

Determines whether files aborted during Kermit downloads are kept or deleted. This command corresponds to the ***Delete aborted downloads*** check box. **fetch** returns 0 for KEEP or 1 for DELETE.

kermit bit8quote character

Specifies the character used to "quote" high-bit ASCII characters during Kermit transfers. Valid values range from 33 to 126. This command corresponds to the ***8th bit quote*** field. **fetch** returns the ASCII value of the current 8th bit quote character.

kermit blockcheck integer

Specifies the type of error-detection used during a Kermit transfer. Valid values for *integer* are 1, 2, and 3. This command corresponds to the ***Block Check Type*** radio button. **fetch** returns 1, 2, or 3, corresponding to ***1 byte checksum***, ***2 byte checksum***, or ***3 byte CRC***, respectively.

kermit blockstart character

Specifies the start block character for a Kermit transfer. Most systems expect this value to be 1, CTRL-A, but acceptable values range from 0 to 127. This command corresponds to the ***Block start*** edit field. **fetch** returns the ASCII value of the current block start character.

kermit ctrlquote character

Specifies the character used to “quote” control characters during a Kermit transfer. Valid values range from 32 to 127. This command corresponds to the ***Control quote*** field. **fetch** returns the value of the current control quote character.

kermit eolchar character

Specifies the end of line character used during a Kermit transfer. Valid values range from 0 to 127. This command corresponds to the ***End of line*** field. **fetch** returns the value of the current end of line character.

kermit filetype BINARY | TEXT

Specifies the type of files being transferred with the Kermit protocol. This command corresponds to the ***File type*** option button. **fetch** returns 0 for BINARY or 1 for TEXT.

kermit overwrite OFF | ON

Determines how filename collisions are handled during a Kermit file transfer. This command corresponds to the ***Overwrite existing file*** check box. **fetch** returns 0 for OFF and 1 for ON.

kermit packetsize integer

Specifies the maximum size of data packets used in Kermit file transfers. Valid values range from 20 to 1024. This command corresponds to the ***Maximum packet size*** edit field. **fetch** returns the current maximum packet size.

kermit padchar character

Specifies the character used to pad empty data packets in a Kermit file transfer. Valid values range from 0 to 127. This command corresponds to the ***Pad*** edit field. **fetch** returns the value of the current pad character.

kermit padnum integer

Specifies the number of pad characters used to fill empty data packets in a Kermit file transfer. Valid values range from 0 to 127. This command corresponds to the ***Number of pad chars*** field. **fetch** returns the current number of pad characters.

RAWASCII Set and Fetch Statements

The following **set**/**fetch** statements apply to the Raw ASCII transfer protocol. These commands correspond to options in **Setup, Data, Transfer Protocol,** with the ***Current Protocol*** set to ***RAWASCII.***

rawascii abortdnld KEEP | DELETE

Determines whether files aborted during a Raw ASCII download are kept or deleted. This command corresponds to the ***Delete aborted downloads*** check box. **fetch** returns 0 for KEEP or 1 for DELETE.

rawascii charpace integer

Specifies the length of the pause between characters during a Raw ASCII upload. This command corresponds to the ***Millisecond delay between characters*** field. **fetch** returns the current character pacing.

rawascii dnld display OFF | ON

Determines whether data is displayed as it is received during a Raw ASCII download. This command corresponds to the ***Download Options Display text*** check box. **fetch** returns 0 for OFF or 1 for ON.

rawascii linepace integer

Specifies the length of the pause between lines during a Raw ASCII upload. This command corresponds to the ***Millisecond delay between lines*** field. **fetch** returns the current line pacing.

rawascii overwrite OFF | ON

Determines how filename collisions are handled during Raw ASCII file transfers. This command corresponds to the ***Overwrite existing file*** check box. **fetch** returns 0 for OFF or 1 for ON.

rawascii pacechar character

Specifies the pace character for Raw ASCII uploads. This command corresponds to the ***Use X for Line pace character*** edit field. **fetch** returns the value of the Raw ASCII upload pacing character.

rawascii timeout integer

Specifies the number of seconds that Procomm Plus waits for incoming information during a Raw ASCII file download before closing the transfer as completed. This command corresponds to the ***Seconds until download time-out*** edit field. **fetch** returns the current download time-out value.

rawascii upld display OFF | ON

Specifies whether data is displayed as it is sent during a Raw ASCII upload. This command corresponds to the ***Upload Options, Display text*** check box. **fetch** returns 0 for OFF or 1 for ON.

rawascii usepacechar OFF| ON

Specifies whether the Raw ASCII protocol uses the specified pace character to control a Raw ASCII upload. This command corresponds to the ***Use x for Line pace character*** check box. **fetch** returns 0 for NO or 1 for YES.

ZMODEM Set and Fetch Statements

The following **set**/**fetch zmodem** statements control the operation of the Zmodem transfer protocol. They correspond to options in **Setup, Data, Transfer Protocol**, with ***Zmodem*** selected as the ***Current Protocol***.

zmodem eolconvert OFF | ON

Determines whether Linefeeds, ASCII 10, are changed according to the End of Line convention for the system receiving the file. If ON is specified, Linefeeds in a received file are translated to Carriage Return/Linefeed, ASCII 13/ASCII 10 pairs. This command corresponds to the ***Use local EOL convention*** check box. **fetch** returns 0 for OFF or 1 for ON.

zmodem errordetect CRC16 | CRC32

Determines the error-detection method used in Zmodem file transfers. This command corresponds to the ***Error detection*** list. **fetch** returns 0 for CRC16 or 1 for CRC32.

zmodem origtime OFF | ON

Determines whether files received with the Zmodem protocol preserve their original time and date stamps or the time and date stamps are changed to the time the file is saved. This command corresponds to the ***Original file time stamp*** check box. **fetch** returns 0 for OFF or 1 for ON.

zmodem receiver crashrecover OFF | ALWAYS | CRC | TIMESTAMP | SENDER

Determines how aborted Zmodem downloads are restarted. This command corresponds to the ***Crash Recovery Options*** in the **Zmodem Receive Options** dialog which is available after clicking ***Change Settings...*** in the ***Receiver Crash Recovery Settings*** groupbox. **fetch** returns 0 for OFF, 1 for ALWAYS, 2 for CRC, 3 for TIMESTAMP, or 4 for SENDER.

zmodem receiver overwrite NEWER | ALWAYS | SKIP | RENAME

Suggests how filename collisions should be handled during a Zmodem download. This command corresponds to the ***Overwrite Options*** in the **Zmodem Receive Options** dialog which is available after clicking ***Change Settings...*** in the ***Receiver Crash Recovery Settings*** groupbox. **fetch** returns 0 for NEWER, 1 for ALWAYS, 2 for SKIP, and 3 for RENAME.

zmodem sender crashrecover OFF | ALWAYS | CRC

Determines how aborted Zmodem uploads are restarted by the host. This command corresponds to the ***Crash Recovery Options*** in the **Zmodem Send Options** dialog which is available after clicking ***Change Settings...*** in the ***Sender Crash Recovery Settings*** groupbox. **fetch** returns 0 for OFF, 1 for ALWAYS or 2 for CRC.

zmodem sender overwrite NEWER | ALWAYS | SKIP

Suggests how the remote system should handle filename collisions during Zmodem transfers. This command corresponds to the ***Overwrite Options*** in the **Zmodem Send Options** dialog which is available after clicking ***Change Settings...*** in the ***Sender Crash Recovery Settings*** groupbox. **fetch** returns 0 for NEVER, 1 for ALWAYS or 2 for SKIP.

zmodem txmethod STREAMING | 2KWINDOW | 4KWINDOW

Determines the transmit method used by the Zmodem protocol. This command corresponds to the ***Transmit method*** list. **fetch** returns 0 for STREAMING, 1 for 2KWINDOW, or 2 for 4KWINDOW.

Xmodem, 1K-Xmodem, 1K-Xmodem-G, Ymodem, and Ymodem-G Set and Fetch Statements

The following **set**/**fetch** statements are general commands for the Xmodem, 1K-Xmodem, 1K-Xmodem-G, Ymodem and Ymodem-G protocols. They correspond to options in **Setup, Transfer Protocol**, with the ***Current Protocol*** set to the transfer protocol specified by the *protocol* parameter. For example, to enable the overwrite option for Xmodem, use **set xmodem overwrite ON**.

protocol abortdnld KEEP | DELETE

Determines whether aborted file transfers are kept or deleted. This command corresponds to the ***Delete aborted downloads*** check box for the listed protocols. **fetch** returns 0 for KEEP or 1 for DELETE.

protocol overwrite OFF | ON

Determines how filename collisions are handled for the specified protocol. This command corresponds to the ***Overwrite existing file*** check box for the listed protocols. **fetch** returns 0 for OFF or 1 for ON.

protocol relaxed OFF | ON

Enables or disables relaxed timing for the specified protocol. This command corresponds to the ***Use relaxed protocol timing*** check box for the listed protocols. **fetch** returns 0 for OFF or 1 for ON.

WWW Set and Fetch Statements

These statements control the operation of the Procomm Plus *Web Browser*. They correspond to the Web Options in the ***Setup, Internet, Web Options*** dialog.

www animate OFF | ON

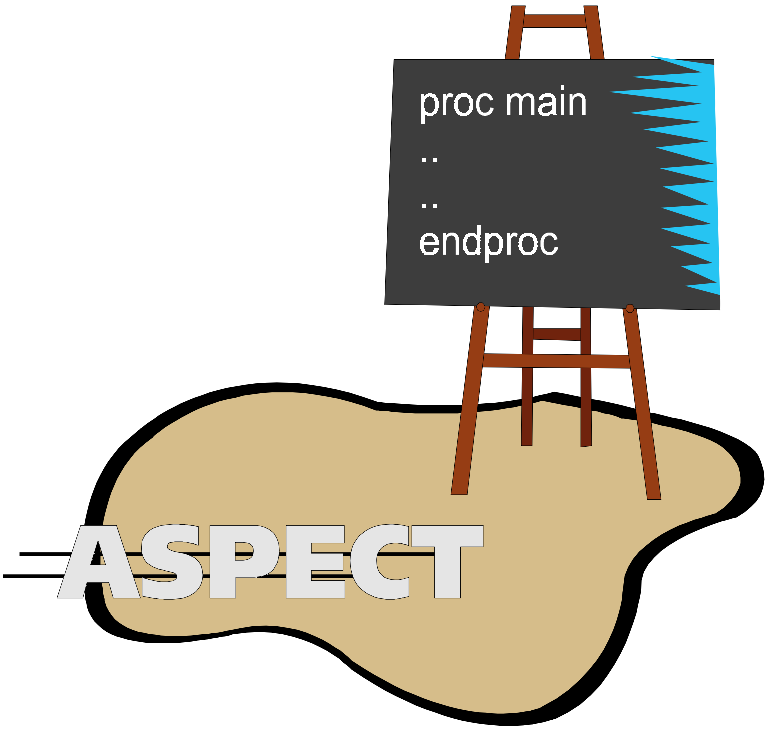
Enables or disables the animated graphics in the *Web Browser* window. This command corresponds to the ***Animate stop sign on Action Bar*** check box in the **Setup, Internet, Web Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

www multiplewindows OFF | ON

Enables or disables the spawning of additional *Web Browser* windows corresponding to the ***Use multiple windows*** check box in the **Setup, Internet, Web Options** dialog. **fetch** returns 0 for OFF or 1 for ON.

Chapter 5

System Variables



Introduction

ASPECT provides a number of variables that are used to report the status of operations and events in Procomm Plus. These variables have values that are maintained internally by Procomm Plus and ASPECT. A script cannot directly alter the value of a system variable, but many ASPECT commands can cause a system variable's value to change.

A system variable is "read only," meaning it can only be used in commands where a constant is permitted. A system variable cannot be used in place of a data variable, since that would imply that the system variable is to be set to a result value upon termination of a command. System variables cannot be passed by reference to a user-defined procedure or function, but they can be passed by value. Other usage restrictions may apply on a per-command basis. For example, a system variable cannot be used within a **dialogbox** command group.

Certain system variables reset their values to a default value after they are accessed. Others have values that remain unchanged for the duration of script execution. Still other system variables change value dynamically based on current program activities and operations.

The following system variables return window ids, and can be used in any ASPECT command where a window id is required: $ACTIVEWIN, $CHATWIN, $DIALDIR, $FOCUSWIN, $LMOUSEWIN, $MAINWIN, $MONITORWIN, $MMOUSEWIN, $POINTERWIN, $PWMAINWIN, $RMOUSEWIN, $SETUP, and $USERWIN. You can use them in any ASPECT command where a window ID is required.

$POINTERTASK, $PWTASK, and $TASK return task IDs. Their values may be used in any ASPECT command where a task ID is required.

Option set item strings are returned by $DATAOPTIONS, $FAXOPTIONS, $FTPOPTIONS, $MAILOPTIONS, $NEWSOPTIONS, $PROTOCOL, $TELNETOPTIONS, $TERMFONT, $TERMCOLORS, $TERMINAL, and $WINCOLORS.

The following integer system variables reset their values to 0 after they are accessed: $ACTIONBAR, $ASPMENU, $DDEADVISE, $ERRORNUM, $LMOUSEEVENT, $MCINOTIFY, $OBJECT, $PKRECV,$PKSEND, and $RMOUSEEVENT. $XFERSTATUS also resets to 0 after being accessed, but only when its value does not indicate an ongoing transfer.

SUCCESS and FAILURE are two special global system variables. For more information, see "*A SUCCESS and FAILURE*" on page 451.

The System Variables

The system variables are listed below alphabetically with their descriptions and other information.

$ACTIONBAR

An integer value reflecting an event from an *Action Bar* configured with a script notification code. $ACTIONBAR resets to 0 after it is accessed.

$ACTIONBARS

Returns a set of bit flags, indicating which *Action Bars* are currently displayed. Values are 0x01 (top), 0x02 (bottom), 0x04 (left), 0x08 (right) and 0x10 (floating).

$ACTIVEWIN

An integer value which is the window ID of the currently-active window. Usually, this window has the current input focus. This value can be used in any command that accepts a window ID.

$ASPECTPATH

A string value containing the currently-defined Aspect Path. This value may be altered by a **set aspect path** command. When a script is launched, $ASPECTPATH is initialized to the drive and directory that contains the script's **.wax** file, unless the script inherited its environment from another script. For more information, see "*The ASPECT Script Environment*" on page 54.

$ASPMENU

An integer value, identifying the last command selected by the user from a custom menu created with **menuitem**. $ASPMENU is set to 0 after it's accessed.

when $ASPMENU will "fire" whenever $ASPMENU's value is non-zero. The procedure handling the **when** event should access the available menu value. Otherwise, the value will be automatically cleared when the procedure returns and the script will lose the value.

$CALLERID

A string containing the number when a Caller ID is recognized. $CALLERID is reset to null after it is read.

$CAPTURE

An integer value that returns non-zero when capturing incoming data to a file and 0 otherwise.

$CARRIER

An integer value reflecting the current status of the RS-232 Carrier Detect (CD) signal. $CARRIER is 0 if no carrier is detected (CD is low), or 1 if a carrier is present (CD is high). If used as a **when** variable, the **when** will “fire” whenever CD changes. $CARRIER is only available in *Terminal* and *Telnet* modes.

$CHAINEDFILE

If the current script was launched from another script using the **chain** command, this string variable will contain the name of the calling script. If the current script was not **chain**ed, this string will be null.

The $CHAINEDFILE variable could be used to restart the original script, or to branch based on the calling script name. The $SCRIPTMODE variable can be tested to determine how the current script was launched.

$CHATWIN

An integer value, equal to the window ID of the *Chat* window or to 0 if the *Chat* window does not exist. This value can be used in any command accepting a window ID.

$CNCTMSG

A string containing the modem result message matched upon a successful connection. The message should contain the baudrate for the connection or a connection error message—these are the same messages you would see when a connection is established in the Dialing Progress dialog.

$CNCTMSG is reset to null after it has been accessed.

If used as a **when** variable, the procedure handling the when event should access the available $CNCTMSG value. Otherwise, it will be automatically reset when the procedure returns and the script will lose the value.

$COL

An integer value equal to the current column cursor position in the *Terminal* display.

$COMPANY

A string variable containing the company name entered during the installation of Procomm Plus.

$CONNECTOPEN

This integer variable returns 1 if a connection is open and zero otherwise. When a connection is open, you can read and write to the communications port provided no other activity (such as a file transfer) prevents it.

$CTS

An integer value reflecting the state of the RS-232 Clear to Send (CTS) line. $CTS equals 1 if CTS is high or 0 if CTS is low. $CTS is only available in *Terminal* and *Telnet* modes. 0 is also returned if the port is closed or invalid.

$DATAOPTIONS

A string value representing the name of the current option selected in the ***Current Data Options*** field in the **Setup, Data, Data Options** dialog. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$DATE

A string value containing the current system date in the format specified for ***Short date style*** in the ***Date*** tab of the **Regional Settings Properties** dialog. (This dialog is available from the Control Panel.)

$DDEADVISE

The ID number identifying the last **ddeadvise** statement whose *gdatavar* received a new value. If an ID value was not specified for the **ddeadvise** statement, $DDEADVISE has a value of (-1).

$DDIRFNAME

A string containing the full path and filename of the current *Connection Directory*.

$DIALCHANGED

An integer value which returns the number of changes which have occurred in the *Connection Directory* that have nor been saved to disk. A successful **dialadd**, **dialinsert**, and **dialdelete** will increment the count, as will any **set** command that alters the *Connection Directory* data.

$DIALCHANGED is reset to 0 when a **dialsave**, **dialload**, or a **dialcreate** is executed, or if the user opens a *Connection Directory* or creates a new one.

$DIALCONNECT

Identifies the name of the *Connection Directory* entry that established the current connection. $DIALCONNECT is set to null if the connection was made via manual dialing, or if no connection exists, or if a *Connection Directory* entry using a direct connection was selected and carrier is not detected.

$DIALCONNECT can be used in the **dialclass** command to determine which class the *Connection Directory* entry came from. A single class will be returned, unlike the general case where **dialclass** returns a value indicating all classes supported by the entry.

$DIALDIR

Returns the window ID of the *Connection Directory* window. It returns 0 if the *Connection Directory* window does not exist. $DIALDIR can be used in any command accepting a window ID.

$DIALENTRY

A string containing the name of the *Connection Directory* entry that launched the currently-executing script. $DIALENTRY is null if the script was not launched by a *Connection Directory* entry.

$DIALENTRY can be used in the **dialclass** command to determine which class (DATA, TELNET, FTP, or WWW) the *Connection Directory* entry came from. A single class will be returned, unlike the general case where **dialclass** returns a value indicating all classes supported by the entry.

The $SCRIPTMODE variable can also be tested to determine how the current script was launched.

$DIALING

Returns the type of the *Connection Directory* entry currently being dialed. Its value can be 0 for not dialing, 1 for DATA, 2 for FAX, or 4 for VOICE. $FAXSTATUS can be used to determine whether fax processes are on-going.

$DIALQUEUE

An integer value equal to the number of entries remaining in the dialing queue. This value is typically non-zero whenever $DIALING is non-zero.

$DIALSELECT

A string containing the name of the last *Connection Directory* entry dialed. It is null when no dial attempt has been made in the current Procomm Plus session.

$DIALSELECT can be used to determine which entry is being dialed when $DIALING returns a non-zero value, or to determine which entry is responsible for the current Procomm Plus settings since these settings change whenever a *Connection Directory* entry is dialed.

$FAXSTATUS can be used to determine whether fax processes are active, since $DIALING will not reflect fax operations.

$DIALSELECT can also be used in the **dialclass** command to determine which class the *Connection Directory* entry came from. A single class will be returned, unlike the general case where **dialclass** returns a value indicating all classes supported by the entry.

$DTR

An integer value reflecting the state of the RS-232 Data Terminal Ready (DTR) line. $DTR equals 1 if DTR is high or 0 if DTR is low. $DTR is only available in *Terminal* and *Telnet* modes. 0 is also returned if the port is closed or invalid.

$ERRORNUM

An integer value, reflecting the last ASPECT run-time error.

when $ERRORNUM will “fire” whenever $ERRORNUM’s value is non-zero. The procedure handling the **when** event should read $ERRORNUM to reset it to 0. Otherwise, it will be automatically cleared when the procedure returns and the script will lose the value.

$ERRORNUM, when used with a **when** condition, can specify a procedure to handle non-critical run-time ASPECT errors. For example, the script could perform clean-up tasks before exiting, or choose to ignore the error altogether!

To prevent the display of error messages, use the **set aspect errormsg** command.

$EXITCODE

Contains the return value from a child script’s **exit** command. If no value was specified in the child script, this will equal 0.

$FATTR

A string containing the attributes of the last file found by a **findfirst** command or **findnext** command. It returns an ‘N’ to indicate a “normal” file with no attributes set.

$FAXFILE

A string, containing the name of the **.fax** or **.bft** file currently being sent or received.

$FAXMESSAGE

A string value containing the last fax activity message. This is the message that appears in the ***Activity*** field in *Fax Status*.

$FAXOPTIONS

A string value representing the name of the current fax option set as it appears in the ***Current User Info*** list in the **Setup, Fax, User Info** dialog. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$FAXRECVCNT

An integer, indicating the number of faxes in the received-fax log.

$FAXSENDCNT

An integer, indicating the number of faxes remaining to be sent.

$FAXSTATUS

An integer, reflecting the state of the current modem/connection within *Setup*. Possible values are:

Value	Description
0	Not busy
1	Busy, sending
2	Busy, receiving
3	Successfully sent or received
4	Unsuccessfully sent or received

$FAXSTATUS will be set to 3 (Successfully sent or received) or 4 (Unsuccessfully sent or received) after an entire send or receive operation has been completed, even if the operation sent or received multiple faxes. $FAXFILE will update as necessary throughout the operation.

Similar to $XFERSTATUS, $FAXSTATUS will be reset to 0 after a value of either 3 or 4 has been read.

$FAXSTATUS behaves differently when a **faxpoll** command is executed. $FAXSTATUS will be set to 1 (Busy, sending), when a **faxpoll** command is being executed, but $FAXFILE will return null, since no file is being sent. $FAXSTATUS can then change from 1 to 2 (Busy, receiving), at which time $FAXFILE has the name of the file being received.

$FDATE

A string containing the creation date of the last file found by a **findfirst** or **findnext** command in the format specified for ***Short date style*** in the ***Date*** tab of the **Regional Settings Properties** dialog. (This dialog is available from the Control Panel.)

$FEXT

A string containing the extension, up to three characters following the period in a filename, of the last file found by a **findfirst** or **findnext** command.

$FILENAME

A string containing the full filename, with its extension, of the last file found by a **findfirst** or **findnext** command. It contains the full filename, but no path or drive information.

$FILESPEC

A string containing the fully-qualified *filespec* of the last file found with a **findfirst** or **findnext** command. The pathname portion of the string will be either the short or long path format specified in the **findfirst** command. Note that **findfirst** does not expand the pathname portion to its long pathname equivalent.

$FLOWSTATE

An integer indicating the current state of software or hardware flow control. $FLOWSTATE equals 1 if either form of flow control has paused the flow of data. Otherwise, it is 0. 0 is also returned if the port is closed or invalid.

$FLTIME

A long value of the type *timeval*, indicating the time and date stamp of the last file found by a **findfirst** or **findnext** command. For more information, see "*ASPECT Conventions*" on page 42.

$FNAME

A string containing the base filename, excluding its extension, of the last file found by a **findfirst** or **findnext** command.

$FOCUSWIN

An integer value, equal to the window ID of the window which currently has the input focus. The window with the input focus will receive data typed at the keyboard. This value can be used in any command accepting a window ID.

$FSIZE

A long value equal to the size, in bytes, of the last file returned by a **findfirst** or **findnext** command.

$FTIME

A string containing the creation time for the last file found by a **findfirst** or **findnext** command. This string uses the format specified for ***Time style*** in the ***Time*** tab of the **Regional Settings Properties** dialog. (This dialog is available from the Control Panel.)

$FTPCONNECT

An integer, reflecting the state of the *current* FTP connection. Possible values are:

Value	Description
0	Idle
1	Connecting
2	Connected
3	Connection failed or shut down.

If $FTPCONNECT equals 3, it resets to 0 after it is accessed.

$FTPOPTIONS

A string value representing the name of the current FTP option set as it appears in the ***Current FTP Options*** list in **Setup, Internet, FTP Options**. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$FTPSTATUS

An integer value reporting the status of remote FTP operations. Possible values are 0 for idle, 1 for busy, 2 for success, and 3 for failure. If the variable is read when its value is 2 or 3, it is automatically reset to 0.

$IPADDRESS

A string value.

In *News* or *Mail* modes, it contains the IP address of the news or email server to which Procomm Plus is connected.

In *Telnet* mode, it contains the host IP address.

In *Web* mode, it contains the document name you're currently viewing, which could be either a filename or URL.

$IPADDRESS is reset whenever Procomm Plus switches communication modes; it will always be null when Procomm Plus is not using an active Internet connection.

$KEYHIT

An integer value equal to the number of keyhits available in the keyboard buffer. If no keyhit is available, $KEYHIT will return 0. $KEYHIT will always return 0 when **set aspect keys** OFF has been issued.

when $KEYHIT will "fire" whenever $KEYHIT’s value is non-zero. The procedure handling the **when** event should read the available key value using the **keyget** command to reset $KEYHIT. Otherwise, the **when** $KEYHIT will process repeatedly for the same value.

$LMOUSEEVENT

An integer value equal to 1 whenever the left mouse button is pressed down while the mouse pointer is within the boundaries of the client area of Procomm Plus in *Data* and *Web* modes. For more information, see the **uwincreate** command.

$LMOUSESTATE

An integer value, reflecting the state of the left mouse button. Equal to 1 if the button is depressed, 0 otherwise.

$LMOUSEWIN

An integer value reflecting the window ID over which the mouse event occurred.

$LMOUSEX

An integer value, set to the X-coordinate of the mouse pointer whenever the left mouse button is pressed down within the boundaries of the client area of Procomm Plus in *Data* and *Web* modes. The value is updated again when the left mouse button is released. Coordinate values are relative to the window specified with the **set aspect mousecoord** command.

$LMOUSEY

An integer value, set to the Y-coordinate of the mouse pointer whenever the left mouse button is pressed down within the boundaries of the client area of Procomm Plus in *Data* and *Web* modes. The value is updated again when the left mouse button is released. Coordinate values are relative to the window specified with the **set aspect mousecoord** command.

$LTIME

A long value of type *timeval* representing the number of seconds between midnight of January 1, 1970 and the current system time and date. For more information on *timevals*, see “*ASPECT Conventions*” on page 42.

$MAILOPTIONS

A string containing the name of the currently selected option set in the ***Current Mail Options*** field of the **Setup, Internet, Internet Mail** dialog. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$MAINWIN

An integer value equal to the window ID of the active application's main (top level) window. This value can be used in any command accepting a window ID.

$MAPIENABLED

An integer value representing the status of MAPI mail services: a 1 is returned if MAPI services are available, a 0 otherwise.

$MCIDEVICEID

An integer value that reports the device ID of the last device to send a notification. This information can be used along with the string returned by a **mcisend** command.

$MCINOTIFY

An integer value that will be set to the result of a **mciexec** command or **mcisend** command if the *notify* parameter was specified. Possible values are:

Value	Description
1	Operation succeeded
2	Operation superseded (by another command notification request)
4	Operation aborted
8	Operation failed

After this variable is accessed, $MCINOTIFY is reset to 0.

$METAKEYS

An integer value, indicating the status of the *Meta Keys* display. The value is 1 if the keys are displayed or 0 if they are hidden.

$MISC

A string equal to the contents of the ***Miscellaneous*** field in the last *Connection Directory* entry dialed. This string is initialized to the corresponding string in a *Connection Directory* entry whenever that entry is dialed. It is null when a dialing attempt has not been made in the current Procomm Plus session.

$MMOUSEEVENT

An integer value equal to 1 whenever the middle mouse button is pressed down while the mouse pointer is within the boundaries of the client area of Procomm Plus in *Data Terminal* and *Web* modes. For more information, see the **uwincreate** command.

$MMOUSESTATE

An integer value, reflecting the state of the middle mouse button. Equal to 1 if the button is depressed, 0 otherwise.

$MMOUSEWIN

An integer value reflecting the window ID over which the middle mouse button event occurred.

$MMOUSEX

An integer value, set to the X-coordinate of the mouse pointer whenever the middle mouse button is pressed down within the boundaries of the client area of Procomm Plus in *Data* and *Web* modes. The value is updated again when the middle mouse button is released. Coordinate values are relative to the window specified with the **set aspect mousecoord** command.

$MMOUSEY

An integer value, set to the Y-coordinate of the mouse pointer whenever the middle mouse button is pressed down within the boundaries of the client area of Procomm Plus in *Data* and *Web* modes. The value is updated again when the middle mouse button is released. Coordinate values are relative to the window specified with the **set aspect mousecoord** command.

$MODEMCONNECT

A string value representing the name of the current system connection as it appears in the ***Current Modem/Connection*** list in the **Setup, System, Modem Connection** dialog.

$MONITORWIN

Returns the window ID of the *Monitor Mode* window. It is 0 if the *Monitor Mode* window does not exist. This value can be used in any command accepting a window ID.

$NEWSOPTIONS

A string containing the name of the currently selected option set in the ***Current News Options*** field of the **Setup, Internet, News Reader** dialog. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$NEWSSTATUS

An integer that returns 1 when news is being retrieved and 0 when the *News Client* is idle.

$NULLSTR

Returns a null or empty string. $NULLSTR can be used to test a string variable for the null condition.

$NUMCOLORS

An long value equal to the number of colors supported by the current Windows display driver. Typical values are 2, 16, and 256.

$NUMTASKS

Returns the number of tasks currently running in Windows. Note that this number may be greater than the number of “standard” Windows tasks returned with the **firsttask** command and the **nexttask** command. An integer value.

$OBJECT

Contains the ID value for the last User window object selected by the user. Selectable objects include **bitmap**s, **iconbutton**s, **pushbutton**s and **hotspot**s. $OBJECT is reset to 0 after it is accessed.

when $OBJECT will "fire" whenever $OBJECT’s value is non-zero. The procedure handling the **when** event should access the available $OBJECT value. Otherwise, it will be automatically cleared when the procedure returns and the script will lose the value.

$OS

A string value identifying the operating system running the current session.

$OSVER

Identifies the version of the operating system. A string value.

$PARENTFILE

A string value, $PARENTFILE contains the name of the script that will resume execution when the current script terminates. If the script was spawned from another script, this variable identifies the script that spawned it. The $SCRIPTMODE variable can be tested to determine how the current script was launched. A **chain**ed script can also have a parent script.

$PASSWORD

The contents of the ***Password*** field in the last *Connection Directory* entry dialed. This string is initialized to the corresponding string in a *Connection Directory* entry whenever that entry is dialed. It is null when no connect attempt has been made in the current Procomm Plus session.

$PKRECV

An integer, reflecting the status of a **pkrecv** command. Possible values are:

0	Packet not received or idle (default state)
1	Packet received successfully
2	Timeout occurred, packet not received
3	Errors occurred, packet not received

$PKRECV is reset to 0 after it's accessed.

when $PKRECV will "fire" whenever $PKRECV's value is non-zero. The procedure handling the **when** event should access the available $PKRECV value. Otherwise, it will be automatically cleared when the procedure returns and the script will lose the value.

$PKSEND

An integer, reflecting the status of a **pksend** command. Possible values are:

0	Packet send in progress or idle (default state)
1	Packet sent successfully
2	Time-out occurred, packet not sent
3	Errors occurred, packet not sent
4	Packet send canceled by receiver, packet not sent

$PKSEND is reset to 0 after it's accessed.

when $PKSEND will "fire" whenever $PKSEND’s value is non-zero. The procedure handling the **when** event should access the available $PKSEND value. Otherwise, it will be automatically cleared when the procedure returns and the script will lose the value.

$PLAYBACK

An integer value, equal to 1 if a file is currently being played back. Otherwise, it is 0.

$POINTERTASK

An integer value, equal to the ID of the task over which the mouse pointer is currently positioned. This value can be used in any command which accepts a task ID.

$POINTERWIN

An integer, equal to the ID of the window over which the mouse pointer is currently positioned. This value can be used in any command accepting a window ID.

$POINTERX

An integer, containing the absolute X screen coordinate of the current mouse pointer position.

$POINTERY

An integer, containing the absolute Y screen coordinate of the current mouse pointer position.

$PROTOCOL

A string value representing the name of the current protocol option set as it appears in the ***Current Transfer Protocol*** list in **Setup, Data, Transfer Protocol**. The **itemfind** and **itemname** commands can obtain the indices and name strings for all available protocols.

$PWACTIVE

An integer value, equal to 1 if the Procomm Plus session that launched the script is currently the active application. Equal to 0 otherwise.

$PWLOCALPATH

A string, containing the path where Procomm Plus files are stored. The **diaload**, **dialsave**, and **dialcreate** commands all use the local task path. *Meta Key*, *Translate Table*, *Action Bar,* and keyboard files are also stored there. Also see “*$PWTASKPATH*” on page 443.

In a network installation, certain files are installed local to the user, such as *Connection Directories* and saved setup values. Others, such as executable files, are not.

$PWMAINWIN

An integer value, equal to the window ID of the Procomm Plus main or top-level application window. This value can be used in any command accepting a window ID.

$PWMENUBAR

Contains the integer ID value of the current Procomm Plus menu bar. This value is used with the **menupopup**, **menuitem**, and **menushow** commands. This value may be 0 if there is no menu displayed.

$PWMENUDEF

An integer value that returns the ID of the default menu bar for Procomm Plus’s current mode of operation.

$PWMODE

An integer value that contains the current execution mode of Procomm Plus. Possible values are:

0	Procomm Plus is in Terminal mode
1	Procomm Plus is in FTP mode
2	Procomm Plus is in WEB mode
3	Procomm Plus is in MAIL mode
4	Procomm Plus is in NEWS mode
5	Procomm Plus is in TELNET mode

$PWTASK

An integer value, equal to the Procomm Plus task ID. This value can be used in any command which accepts a task ID.

$PWTASKPATH

Returns Procomm Plus’s executable directory path. A string value. Also, see “*$PWLOCALPATH*” on page 442.

$PWTITLEBAR

The current contents of the Procomm Plus application title bar. A string value.

$PWVER

A string value containing the version number of Procomm Plus.

$PWWINSTATE

An integer value, reflecting the current status of Procomm Plus. Possible values are:

0	Procomm Plus is minimized
1	Procomm Plus is restored
2	Procomm Plus is maximized

$QUICKSELECT

An integer value, reflecting the state of the Procomm Plus *Quick Select Line*. Equals 1 if it is displayed, 0 otherwise.

$RMOUSEEVENT

An integer value equal to 1 whenever the right mouse button is pressed down while the mouse pointer is within the boundaries of the client area of Procomm Plus in *Data* and *Web* modes. For more information, see the **uwincreate** command.

$RMOUSESTATE

An integer value, reflecting the state of the right mouse button. Equal to 1 if the button is depressed, 0 otherwise.

$RMOUSEWIN

An integer value reflecting the window ID over which the mouse event occurred.

$RMOUSEX

An integer value, set to the X-coordinate of the mouse pointer whenever the right mouse button is pressed down within the boundaries of the client area of Procomm Plus in *Data* and *Web* modes. The value is updated again when the right mouse button is released. Coordinate values are relative to the window specified with the **set aspect mousecoord** command.

$RMOUSEY

An integer value, set to the Y-coordinate of the mouse pointer whenever the right mouse button is pressed down within the boundaries of the client area of Procomm Plus in *Data Terminal* and *Web* modes. The value is updated again when the right mouse button is released. Coordinate values are relative to the window specified with the **set aspect mousecoord** command.

$ROW

The current row cursor position within the *Terminal* display. An integer value.

$RTS

An integer value reflecting the state of the RS-232 Ready to Send (RTS) line. Will equal 1 if RTS is high or 0 if RTS is low. $RTS is only available in *Terminal* and *Telnet* modes. 0 is also returned if the port is closed or invalid.

$RXCOUNT

An integer value, equal to the number of characters remaining in the receive data buffer. These characters may or may not be available to ASPECT depending on which process currently “owns” the buffer. $RXCOUNT is only available in *Terminal* and *Telnet* modes.

when $RXCOUNT will "fire" whenever $RXCOUNT’s value is non-zero. The procedure handling the **when** event should perform some action to clear the buffer and reset $RXCOUNT. Otherwise, the **when** $RXCOUNT will process repeatedly for the same data.

$RXDATA

An integer value, equal to the number of characters the script can retrieve in the received data buffer. $RXDATA is only available in *Terminal* and *Telnet* modes.

when $RXDATA will "fire" whenever $RXDATA’s value is non-zero. The procedure handling the **when** event should perform some action to clear the buffer and reset $RXDATA. Otherwise, the **when** $RXDATA will process repeatedly for the same data.

$SCRIPTENV

An integer value, equal to 1 if the ASPECT environment the current script has is being shared with the script that launched it; otherwise 0. For further information, see the **execute** command, and “*Script Spawning and Chaining*” on page 55.

$SCRIPTFILE

A string, containing the fully-qualified path and filename of the currently-executing script file.

$SCRIPTMODE

An integer value reflecting how the currently-executing script was launched. Possible values are:

0	Normal execution.

1	Spawned script (**execute** issued by another script). Note that $PARENTFILE contains the name of the script which spawned the currently-executing script.
2	Chained script (**chain** command issued by another script). Note that $CHAINEDFILE contains the name of the script which chained the currently-executing script.
3	Executed from the command line at the start-up of Procomm Plus. Note that the pre-defined global variable I0 contains the number of arguments passed to the script from the command line, and S0 through S9 contain the values of those arguments. For more information about these variables, please see "*Predefined Global Variables*" on page 462.
4	Remote script command
5	Executed from a *Connection Directory* entry. Note that $DIALENTRY contains the name of the Connection Directory entry.
6	Executed from distinctive ring.
7	Executed from DDE execute.
8	Executed from DDE aspectcmd command.
9	Executed from a *Caller ID Directory* match. Note that $CALLERID contains the caller ID number associated with the matching *Caller ID Directory* entry.

$SCROLLBACK

An integer value representing the status of scrollback mode: 0 is returned if it is inactive, 1 if it is active.

$SERIALNUM

The serial number of this copy of Procomm Plus. A string variable.

$SETUP

Returns the window ID of the Setup window, or 0 if the Setup window does not exist. This value can be used in any command accepting a window ID.

$SFILENAME

A string containing the full filename, with its extension, of the last file found by a **findfirst** or **findnext** command. The filename is in the MS-DOS standard 8.3 format, containing the filename and extension, but no path.

$STATIONID

A string value representing the station ID of the remote side of the current fax connection. The value will be null if the current fax connection is not sending or receiving.

$STATMSG

A string value containing the current status line message, if any.

$STATUSLINE

An integer value, equal to 1 if the Procomm Plus status line is displayed; 0 otherwise.

$TASK

An integer value, equal to the task ID of the active Windows application. This value can be used in any command which accepts a task ID.

$TELNETOPTIONS

A string value representing the name of the current telnet option set as it appears in the ***Current Telnet Options*** list in the **Setup, Internet, Telnet** dialog. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$TERMCOLS

An integer value, equal to the number of columns supported by the current emulation. Note that this value does not change if the *Terminal* window is re-sized.

$TERMCOLORS

A string value representing the name of the current terminal colors option set as it appears in the ***Current Terminal Colors*** list in the **Setup, Data, Terminal Colors** dialog. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$TERMFONT

A string value representing the name of the current terminal font option set as it appears in the ***Current Terminal Font*** list in the **Setup, Data, Terminal Fonts** dialog. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$TERMINAL

A string value equal to the name of the current emulation options set as it appears in the ***Current Terminal*** list in the **Setup, Data, Terminal Options** dialog. The **itemfind** and **itemname** commands can obtain the indices and name strings for all available emulations.

$TERMROWS

An integer value equal to the number of rows supported by the current emulation. Note that this value does not change if the *Terminal* window is re-sized.

$TIME

The current system time in a string in the format specified for ***Time style*** in the ***Time*** tab of the **Regional Settings Properties** dialog. (This dialog is available from the Control Panel.)

$TIME24

The current system time in 24-hour format. A string, with the time formatted using information from the ***Time style*** field in the ***Time*** tab of the **Regional Settings Properties** dialog. (This dialog is available from the Control Panel.) A leading 0 is prefixed to an hour less than 10, and the AM/PM/24-hour string is omitted.

$TITLEBAR

A string value, containing the caption or title text of the active application.

$TXCOUNT

An integer value, equal to the number of characters remaining to be transmitted. $TXCOUNT is only available in *Terminal* and *Telnet* modes.

$TXDATA

An integer value, equal to the number of characters that ASPECT can send to the transmit data buffer without causing an overflow. This value is set to 0 when further data cannot be transmitted. $TXDATA is only available in *Terminal* and *Telnet* modes. The value returned is not necessarily the limit of available space that ASPECT has to transmit data. Provided that the additional memory is available, Procomm Plus will dynamically allocate more space to accommodate the transmitted data.

$USERDISK

An integer value, reflecting the disk drive specified in the current User Path. $USERDISK equals 1 for drive A, 2 for drive B, up to 26 for drive Z.

$USERDISKSTR

A string value, containing the “name” of the currently active drive. For example, “**c:**”.

$USERID

A string containing the ***User ID*** field in the last *Connection Directory* entry dialed. This string is initialized to the corresponding string in a *Connection Directory* entry whenever that entry is dialed. It is null when no dial attempt has been made in the current Procomm Plus session.

$USERNAME

A string containing the name of the registered user of the copy of Procomm Plus.

$USERPATH

Contains the script’s current working directory path (for example, “**c:windows**”). When a script is launched, $USERPATH is initialized to the drive and directory that contains the script’s **.wax** file, unless the script inherited its environment from another script. For more information, see “*The ASPECT Script Environment*” on page 54.

The string does not end with a trailing backslash unless the path is the root directory of the current drive. $USERPATH can be changed using the **chdir** command.

$USERWIN

An integer value, equal to the window ID of the *User window*. This value can be used in any command accepting a window ID.

$UWINACTIVE

An integer value that returns the status of the *User window*: 0 for inactive, 1 for active.

$VOLUME

A string containing the disk volume label of the current disk drive specified in the ASPECT User Path. For example, “COMPRESSED”.

$WINCOLORS

A string value representing the name of the current window colors option set as it appears in the ***Current Window Colors*** list in the **Setup, Data, Window Colors** dialog. The **itemname** command and the **itemfind** command can obtain the indices and name strings for all option entries.

$WINPATH

A string value containing the pathname of the Windows directory, which usually contains applications, initialization files, and help files.

$WWWSTATUS

An integer value, reflecting the current status of the Procomm Plus *Web Browser*. Possible values are:

0	Idle
1	Loading page
2	Page successfully loaded
3	Page failed to load

If the value is 2 or 3, it will automatically reset to 0 when accessed.

$XFERFILE

A string, reflecting the name of the file currently being transferred.

$XFERSTATUS

Indicates the status of the current file transfer. If no file transfer is in progress, $XFERSTATUS is set to 0, but it can also be 1 for file transfer in progress, 2 for file transfer completed successfully, or 3 for file transfer aborted.

The **when** $XFERSTATUS command will “fire” when the value of $XFERSTATUS changes to a value other than 0. Values of 2 or 3 are reset to 0 after $XFERSTATUS has been accessed. The procedure handling the **when** event should access the available $XFERSTATUS value. Otherwise, it will be automatically cleared when the procedure returns and the script will lose the value.

$XOFFRECV

An integer value, equal to 1 if an XOFF (ASCII 17 or <Ctrl><Q>**)** has been received by Procomm Plus. 0 is also returned if the port is closed or invalid.

$XOFFSENT

An integer value, equal to 1 if an XOFF (ASCII 17 or <Ctrl><Q>**)** has been sent by the Windows communication driver to halt the flow of incoming data. It cannot be set true by pressing <Ctrl><Q> at the keyboard. 0 is also returned if the port is closed or invalid.

$XPIXELS

An integer equal to the horizontal resolution of the current display screen in pixels.

$YPIXELS

An integer equal to the vertical resolution of the current display screen in pixels.

A *SUCCESS and FAILURE*

ASPECT commands which set SUCCESS and FAILURE are indicated by an ***"S/F"*** bullet in their command descriptions.

FAILURE

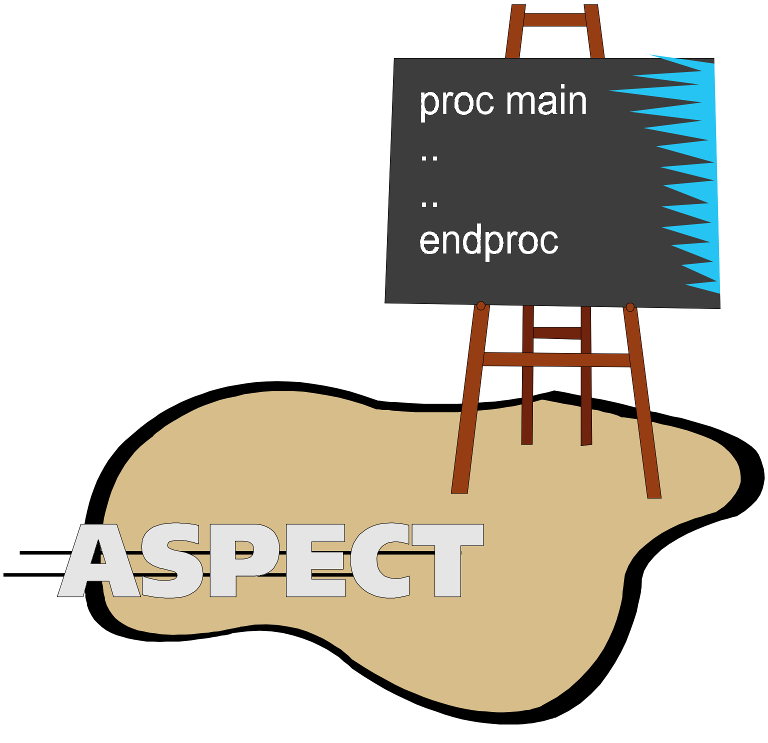
Indicates whether the last testable ASPECT command did not complete successfully. FAILURE evaluates to 1 or true if the last operation failed, 0 otherwise. The FAILURE variable always contains the opposite value of SUCCESS.

SUCCESS

Indicates whether the last testable ASPECT command completed successfully. SUCCESS contains the value 1 for true or 0 for false. The SUCCESS variable always contains the opposite value of FAILURE.

Chapter 6

A Brief ASPECT Tutorial



Introduction

Other areas of the ASPECT documentation introduce you to the features of ASPECT and its command syntax. This tutorial takes a less abstract approach and walks you through real-world examples of ASPECT programming, starting with the most common use of script files and ending with more advanced topics.

Throughout this tutorial there are sample scripts and script fragments. We have included a file named **samples.txt** in your ASPECT directory (by default **...\aspect**) that contains these samples. Use the *ASPECT Editor*, or another text editor to copy script fragments from **samples.txt** into your own scripts, or into new files that you can compile with Procomm Plus.

The tutorial also includes exercises that are designed to help you become comfortable with ASPECT. The exercises begin simply, but advance with the tutorial discussions. Exercises are highlighted with this special graphic to help you locate them quickly.

This Tutorial is designed to be worked through serially, but is actually made up of several different tasks and ideas:

- "*A simple logon script*" on page 455
- "*Dialing without the Connection Directory*" on page 458
- "*Variables: the basics*" on page 460
- "*Predefined Global Variables*" on page 462
- "*Storing Information in Array Variables*" on page 462
- "*Procedures: the Basics*" on page 466
- "*Controlling Program Flow*" on page 469
- "*Functions*" on page 472
- "*Passing Variables by Reference*" on page 475
- "*Calling Conventions*" on page 474
- "*SUCCESS and FAILURE*" on page 476
- "*Getting User Input*" on page 478
- "*A Dialog Box Example*" on page 479
- "*A User Window Example*" on page 484
- "*The when ... call Command*" on page 486

- "*System Variables and the when Command*" on page 489
- "*Sending Keys with ASPECT*" on page 490
- "*Spawning and Chaining ASPECT scripts*" on page 492
- "*Some Notes on Design*" on page 493
- "*Advanced Topics*" on page 496
- "*What's been covered?*" on page 506

As the tutorial progresses, we'll move from simple scripts that introduce concepts to complex scripts that accomplish various tasks, ending with a summary in "*What's been covered?*" For now, let's look at "*A simple logon script*."

A simple logon script

The most common use of ASPECT is the automation of routine communication tasks, such as signing on to a BBS or an information service like CompuServe, Delphi, or MCI. Since this is a logon script, you can easily create it using the Procomm Plus *Script Recorder*.

Using the Script Recorder

You'll need to switch to the *Data Terminal* or *Telnet* window, in order to use the *Script Recorder*. Once you are in one of these two modes, recording a script is quite simple:

1. Start the *Script Recorder* by choosing ***Start Recorder*** from the ***Tools* | *Scripts*** menu, or clicking the ***Record*** button on the *Action Bar.*
2. Perform the actions you want the script to automate.
3. Stop the *Script Recorder* by selecting ***Stop Recorder*** from the ***Tools* | *Scripts*** menu or clicking the ***Record*** icon button again
4. Save the script as a **.was** file.

Once you've saved the script, you can proceed with"*Examining the Source*" on page 456. We have also provided a detailed example of "*Recording a Logon Script*" on page 17.

Within the ***Save Recorded Script*** *dialog, you can attach the script to a Connection Directory entry. Simply select the appropriate entry from* ***Attach to Connection Directory entry:*** *list box.*

Examining the Source

Select ***Tools | Scripts | Compile/Edit...*** from the menu, or click the ***Compile/Edit*** *Action Bar* button to open the **Compile/Edit ASPECT File** dialog. Select your script and click ***Edit*** to open the script.

The script created by the *Script Recorder* may need some minor before you use it. For instance, many of the **waitfor** commands can be shortened to the last few words. Once you're finished editing it, it should look something like this:

```
;Logon to XYZ  Technical Support BBS.
proc Main
  waitfor "your FULL name:  "
  transmit "B.J. Johnson^M"
  waitfor "Is this you? "
  transmit "y"
  waitfor "Your Password: "
  transmit "secret^M"
  waitfor "Any Key-"
  transmit "^M"
endproc
```

The **transmit** statements represent what you normally type when logging on. The **waitfor** statements represent prompts sent to you by the host system. Notice the "**^M**" characters at the end of the **transmit** statements. "**^M**" represents a carriage return and appears in the script file each time you pressed <Enter>.

Note: *The above script is a sample and actual prompts may vary between different BBS systems.*

ASPECT scripts are organized or structured by procedure and function blocks. Every script file requires at least one procedure called **main**. A **proc main** and **endproc** statement pair represent the beginning and end of the **main** procedure, respectively. Later in this tutorial we'll see how a procedure can **call** other procedures or functions to accomplish specialized tasks within your script.

Compiling the Script

Creating an ASPECT source file is only the first step. You must compile the source file before Procomm Plus can use it. If you are using the *ASPECT Editor*, you can simply click the ***Save and Compile*** button from the *Action Bar*, or select ***Tools | Compile...*** from its menu. This opens the *ASPECT Compiler* as described in "*The ASPECT Compiler*" on page 567.

If you have already closed the *ASPECT Editor*, or you are not using it as your editor, you can compile from within Procomm Plus. Choose ***Compile/Edit*** from the ***Tools | Scripts*** menu or click the same button on the *Action Bar*. This opens the **Compile/Edit ASPECT File** dialog.

Select the file you created either by clicking on it, or by tabbing into the list and moving the focus with the cursor keys. After you select the filename, notice that the "***Notes***" field in the dialog displays the first line of your recorded script.

If the first line of your script is a comment, marked with a semicolon in the first character on the first line, it will be displayed in the ***Notes*** field. If you have a variety of scripts, you can use the ***Notes*** field to help you distinguish between them.

The ASPECT *Script Compiler* automatically processes your script when you click on ***Compile...*** or ***Compile and Run...*** with a file selected. It reports its progress, and any errors or warnings in the ***Message List*** area.

If the compile is successful, the ASPECT *Script Compiler* creates a file with a **.wax** extension that Procomm Plus can run.

Notice the ***Edit source file*** button in the compiler dialog. If the compiler detects a syntax error in your source file, you can click on this button to load the source file into your editor to resolve the problem.

Click on ***OK*** to close the ASPECT *Script Compiler*.

To use your new script file with a *Connection Directory* entry, select it from the ***Script*** list box when you edit the entry. Then, whenever you connect to the host system listed in the entry, Procomm Plus automatically launches the new script. You won't have to enter your name and password anymore!

Remember, if you make any changes to the script source **.was** file, it must be compiled again before Procomm Plus is aware of those changes. You should also be careful to maintain backups of your source files: compiled scripts cannot be uncompiled!

Write and compile a logon script for a BBS or information service you call.

Hint: *Try using the Script Recorder.*

Now, let’s look at how we can communicate directly with the modem in “*Dialing without the Connection Directory*.”

Dialing without the Connection Directory

The **transmit** command can be used to send commands to your modem, as well as to send text such as your name and password as shown in the previous section. Look at the following script file:

```
#define BBS_NUMBER "555-1234" ; Define phone number macro
#define PASSWD "Rosebud^M"    ; Define password macro
proc Main                   ; Start of Main procedure.
  commandmode ON            ; Enter command mode.
  transmit "ATDT"           ; The modem dial command
  transmit BBS_NUMBER       ; .. the number to dial.
  transmit "^M"             ; ... a carriage return.
  waitfor "Name: "          ; Wait for name prompt.
  transmit "Orson Welles^M"; Send your name.
  waitfor "Password:  "     ; Now the password prompt
  transmit PASSWD           ; Send password.
  waitfor "again: "         ; This system confirms passwords
  transmit PASSWD           ; So let’s send it again.
  alarm 2                   ; Wake up!
  usermsg "OK, you’re in"  ; You’re on.
  while $CARRIER            ; Loop as long as the modem is
    yield                   ; connected.
  endwhile                  ;
  commandmode OFF           ; release the modem to TAPI.
endproc                     ; End of Main procedure
```

Using Macros

In this script we use macros or **#define** statements to make our script easier to maintain. Wherever the word ***BBS_NUMBER*** appears it is replaced with the string "***555-1234.***" Similarly, whenever ***PASSWD*** appears, it is replaced with the string "***Rosebud***" followed by a carriage return.

On first glance, it may seem as though our macros were a waste of time. After all, the phone number and password could just as easily have been placed with the **transmit** statements. But what if the script were longer, or you needed to change the password? With the critical strings **#define**d at the top of the script, you'll find them quickly and easily, and you'll be able to change ***every*** instance of your password simply by editing the appropriate **#define** statement! We'll learn more advanced uses for macros in "*Advanced Topics*" on page 496.

Notice the detailed comments for each line of the script file. Comments are optional notes that help you understand your programming later. They're always preceded by a semicolon and are therefore ignored by the compiler, so they don't become part of the compiled script.

We also added two commands: a two-second **alarm** and a simple dialog message with the **usermsg** command to alert you when the connection is made.

Since this script actually dials the phone number, you run it by itself instead of from the *Connection Directory*. After you've compiled the script, you can execute it by choosing ***Run*** from the ***Scripts*** menu, or by selecting it from the ***Script File*** list box on the *Action Bar*.

*There are other ways to automate calling without attaching a script file to a Connection Directory entry. For example, the **dialnumber** command places a call just as though you had initiated a manual dial from the Connection Directory.*

Set and Fetch

ASPECT has a **set** and **fetch** command for almost every option available in Procomm Plus. "*Set and Fetch Statements*" contains a complete list.

For instance, suppose that you have several Internet Mail accounts, including a separate mail account for your business. If you configure each account as an option set within *Setup*, you could specify a particular option set within the script. By creating an option set called "Business Mail" you could simply add this line to a script to switch your option set:

set internet mail options "Business Mail"

Switching option sets is as easy as using a **set** command along with the option set's name included in quotes, or as a string.

As an alternative to using the option set's name, you can reference it numerically using a zero based index. For example:

set internet mail options 1

would assign the second option set as the ***Current Mail Options***.

Write a script that dials a phone number without using the Connection Directory. Use one ***#define*** *statement for the dialing command and another for the phone number.*

Hint: *Don't forget to* ***transmit*** *a Carriage Return after the number! This is the modem's "execute" command, which causes it to process the commands you've sent.*

Now that we have used **#define** statements, let's look at data we can change in "*Variables: the basics*."

Variables: the basics

The rest of the examples in the tutorial use an important feature of the ASPECT language: *global* and *local variables*.

A *variable* is a symbol that stands for a changeable value. For example, in the expression "x = 1," x is a variable. Later in your script you could change the value of the variable with the statement "x = y + 1." First x has the value 1, and after the second expression is evaluated, x takes on the value of another variable, y, plus one. Since x is a variable, its value can change in the course of a script.

Variables can represent various data types. The variable types used most often are **string**s for text, **integer**s for numeric integer values, and **float**s for numeric decimal point values. For a more detailed discussion of ASPECT's data types, see "*Data Types*" on page 22.

You must define a variable before your script can use it. Defining a variable involves specifying a name and data type for the variable. This allows the ASPECT *Script Compiler* to allocate resources for the variable, and to properly interpret the ASPECT commands which use it.

Variables are normally defined at the beginning of a procedure or function, although a variable definition can occur almost anywhere within an ASPECT script. First we'll discuss "*Variable Names*," then we'll look at "*Local Variables*," "*Global Variables*," "*Predefined Global*

Variables,"and finally "*Storing Information in Array Variables*." Once variables have been discussed, we'll continue with "*Procedures: the Basics*."

Variable Names

Variable names in ASPECT can have up to thirty distinct characters. You can use either upper or lower case characters. The compiler treats names in a case-insensitive manner. Thus, "***MyValue***" and "***MYVALUE***" each refer to the same variable.

In our examples, we've used mixed case characters for variable names because they are easier to read. Whether they are "*Local Variables*," or "*Global Variables*," the names must be unique.

Local Variables

This script example shows how variables are defined:

```
proc Main          ; Beginning of Main procedure
string Name        ; Define string variable called "name"
  Name = "Fred^M"    ; Assign the value of name
  transmit Name     ; and send "Fred^M" out the port
endproc
```

In this example, "**string Name**" tells the compiler to create a variable called "***Name***," which will contain character data. Since the "**string Name**" variable definition appears inside the procedure after the "**proc main**" statement, it is known as a *local variable*. Once this line is encountered in the file, ***Name*** can be used anywhere inside the procedure called **main**. It is not available to other procedures. "*Global Variables*," on the other hand, are.

Global Variables

Using the example from "*Local Variables*," if you move the "**string Name**" line outside of the **main** procedure, the variable becomes available to all procedures in the script and is known as a *global variable*:

```
string Name        ; Define the global string variable "Name"
proc Main          ; Beginning of Main procedure
  Name = "Fred^M"   ; Assign text to the variable
  call SendData     ; Call another procedure
endproc            ; End of Main procedure
```

```
proc SendData       ; Procedure called by another procedure.
  transmit Name     ; Send value of the global variable "Name"
  usermsg "Done!"   ; Put up a message box
endproc           ; End of SendData procedure, return
                ; to calling procedure
```

In this case, the variable ***Name*** maintains the value "***Fred^M***" in all procedures, and the word "***Fred***" is transmitted, followed by a carriage return, in ***SendData***. But if we change the value of ***Name*** in the **main** procedure, it changes in ***SendData*** as well, because it was defined globally.

Notice how the "=" operator is used to assign a value to the ***Name*** variable. You can also assign values to variables when they're defined, as shown in the following script segment:

```
string Name = "Fred^M"; Define and assign the variable.
```

Predefined Global Variables

In addition to allowing you to define your own variables, ASPECT provides 40 predefined global variables that you can use "right out of the box." These are the string variables S0 through S9, the integer variables I0 through I9, the long integer variables L0 through L9 and the float variables F0 through F9. Like user-defined variables, you can assign these variables any values you want in the course of your script. For most purposes, it's better to define your own variables. It's very hard to tell from a variable name like ***I5***, for example, what role it plays in a script. Variable names like "***NumberOfApples***," "***CostOfGoods***", or "***NameOfClient***" are much more descriptive.

Although they can be used like any global user-defined variable, the predefined global variables can be used in very special ways in ASPECT, such as sharing values with other ASPECT scripts by spawning and chaining and using DDE. For more information, see "*Spawning and Chaining ASPECT scripts*" on page 492, "*DDE server operation*" on page 498, and "*DDE client commands*" on page 500.

Storing Information in Array Variables

Along with local and global variables, you can also define local and global *array* variables. *Arrays* are best thought of as collections of information that you can reference using a single name. They are used to group together variables of the same type and relation. Individual parts of an array are known as *elements of the array,* and are referenced using subscripts or indices corresponding to their locations. We are going to talk about how to access information stored in arrays a little later. First, let's see how an array is defined.

Defining an Array

Remember that in defining a normal variable, we had to tell the compiler the name of the variable, and its data type:

```
string Mauve
```

defining an array is similar, but we have to tell the compiler the dimensions of the array. We do this by adding a *subscript* expression:

```
string MauveArray[12]
```

The definition for ***MauveArray*** is very similar to the definition of ***Mauve***, above. Other than the variable names, the only real difference is its subscript expression, [12]. The subscript expression tells the compiler how many strings are contained in the array. In ***MauveArray***, we have defined a single-dimensioned array containing 12 string elements.

ASPECT also allows you to create multi-dimensioned arrays. For example, the definition:

```
integer FiveByFive[5][5]
```

creates an array of integers with 25 elements, arranged in five rows of five elements each. In our examples, we have used both integer and string constants for subscript expressions. You can, however, use any integer or long for a subscript expression.

Accessing Array Elements

Array names are handled by the compiler in the same way as names for non-array variables. To access the array elements, however, you must supply the element number you're referencing:

```
Mauve[0] = "What is this color?"
Mauve[1] = "Who owns this car?"
```

The statements above assign the first and second array elements in **Mauve** to "***What is this color?***" and "***Who owns this car?***," respectively.

The subscript expression for the first element in ***Mauve*** *is 0. This is because array indices are zero-based. The first element is addressed using a subscript expression of zero, and the last element in a row or column is addressed using a subscript of one less than the definition subscript value.*

Using a value stored in an array is just like using any other variable:

```
transmit Mauve[1]
usermsg Mauve[0]
```

Global and Local Arrays

Like other variables, arrays can be defined as either local or global. The following example shows how to define both local and global array variables:

```
string Names[10]; Define global string array with 10 elements.
proc Main
integer Ages[10]; Define local integer array with 10 elements.
.
.
endproc
```

Notice how the definition for the array called ***Names*** appears outside of the procedure block and the definition for the array called ***Ages*** appears inside of the procedure block. ***Names*** is a globally defined array variable and ***Ages*** is a locally defined array variable. Both of the array variables in the above example can be used by commands within the procedure called **main**, but only ***Names*** can be used by other procedures.

Now, let's add some assignments to our script file:

```
string Names[10]   ; Define string array with 10 elements.
proc Main
integer Ages[10]   ; Define integer array with 10 elements.
   Names[0] = "Johnny B. G."  ; Assign a value to the first
                              ; element of our array of names.
   Ages[0] = 89    ; Assign a value to the first element of our
                   ; array of ages.
endproc
```

In the script example above, ***Names[0]*** is used to reference the first element in the array called ***Names***, and ***Ages[0]*** is used to reference the first element in the array called ***Ages***. Assigning and retrieving information from arrays is made easy with the use of variable subscripts.

As a practical example, let's make our script file read a list of names and ages from a file and put them in our ***Names*** and ***Ages*** arrays. You'll need to create a text file called "**names.lst**" in a text editor like the *ASPECT Editor* and fill it with 10 names. Follow each name by a comma and an age. Your newly created file should look something like:

```
Bob Jennings,35
Frosty T. Snowman,1
Joseph Risingstar,18
E. L. Capitan,51
.
.
```

Now we can change our script file to open the **names.lst** file, read names and ages from the file, separate the names from the ages, and put the names into the ***Names*** array and the ages into the ***Ages*** array. The resulting script file looks like this:

```
string Names[10]         ; Define 10 element string array.
proc Main
string szLine            ; Variable to store line from file.
string szTemp            ; Temporary variable used for Age.
integer Ages[10]         ; Define 10 element integer array.
integer Count = 0        ; Variable subscript used to
                   ; reference our arrays.
  if fopen 0 "names.lst" READ;Open name file for read only.
    while not feof 0     ; Loop while we’re not at end of
                   ; the file.
      fgets 0 szLine     ;Get a line from the file and
                   ; extract the name and age.
      strtok Names[Count] szLine "," 1
      strtok szTemp szLine "," 1
      atoi szTemp Ages[Count]
                   ; Convert the age to an integer and
                   ; put it into our Ages array.
      if (Count += 1) == 10; Increment array subscript and
        exitwhile        ; make sure that we don’t
                   ; increment past the last elements
      endif              ; in the two arrays.
    endwhile
    fclose 0             ; Close our file of names and ages.
  else            ; Display a message if an error occurred.
    errormsg "Couldn't open NAMES.LST for input!"
  endif
endproc
```

In this example, the variable called ***Count*** is used to reference the elements in the ***Names*** and ***Ages*** arrays. You could expand this example to include displaying the list of names and ages, pumping them into a spreadsheet or word processing program with DDE, or many other tasks.

ASPECT accepts subscript expressions that exceed the number of elements in an array but a run-time error is displayed if such a case occurs. For more information about using arrays, including memory requirements and addressing, see “*Arrays*” on page 30.

Write a script that reads a list of software products and their prices from a file, stores the product names and prices in arrays and then calculates and displays the average price of all of the products. Make your script file support at least 50 products.

Procedures: the Basics

The example in “*Global Variables*” on page 461 was our first look at a script using more than one procedure. When another procedure is called, script execution passes to it. When the called procedure is finished, execution automatically returns to the calling procedure at the line after the **call** statement.

The use of separate procedures may seem odd to you at first. Why not just put everything inside the **main** procedure? Breaking up your scripts into separate procedures gives you several advantages:

- You can break up larger tasks into smaller, more easily-designed steps. This is especially critical in large script projects.
- You can call isolated procedures from other procedures in your script.

  In the previous example, **main** is the only procedure calling ***SendData***, but you could add other procedures that called the ***SendData*** routine.
- Re-using code helps make for smaller, faster running applications.
- Since procedures are reusable, you avoid redundant lines of code.

  If you have broken down a task into several logical procedures or functions, there’s a good chance that at least some of those functions can be useful in other scripts, possibly without requiring adjustment.

- Modular, or procedure-oriented, code is more readable and therefore easier to maintain.

  Changing the way ***SendData*** works is much easier than searching through dozens, hundreds, or even thousands of lines of code to make selective changes.

The ***SendData*** procedure shown in “*Global Variables*” on page 461 requires ***Name*** to be a global variable, accessible to any procedure in the program. Be careful when you use global variables in this manner: any procedure can modify a global variable’s value too!

The **call** *Command*

Unless you are very careful using global variables, they can increase the chance for confusion among different procedures. For instance, in the script sample of “*Global Variables*” on page 461, another procedure can set the value of the variable to your last name rather than your first, and the **main** procedure has no way of detecting it. Let’s rewrite the script so that it uses a local variable rather than a global variable:

```
proc Main               ; Beginning of Main procedure.
string Name = "Fred^M"  ; Define local string variable "Name"
  call SendData with Name; Call another procedure, passing it
                        ; the value of name.
endproc                 ; End of Main procedure.

proc SendData           ; Procedure called by another proc.
param string szText     ; Define a local variable to contain
                        ; the value passed from another
                        ; procedure. In this case a string.
  transmit szText       ; Send the value of the local variable
                        ; "szText", which is equal to the
                        ; value of name
  usermsg "Done!"       ; Message box.
endproc                 ; End of SendData procedure,
                        ; execution returns to calling proc.
```

In this example we use the **call** command not only to invoke another procedure, but to pass it a value as well. This demonstrates a very powerful feature of ASPECT. The **call...with** construct only passes a variable’s value, and not the variable itself. The **call**ed procedure accepts that value into its own local variable and can manipulate it in any way without changing the value of the original variable in the **main** procedure.

The ability to pass arguments greatly enhances the usefulness and modularity of **call**ed procedures. Now we can **call SendData** several times for different purposes:

```
;***********************************************************
; Main procedure.
; Calls: SendData.
; Called by:
; Modifies the value of the following globals:
;***********************************************************
proc Main            ; Beginning of Main procedure.
string FirstName = "Fred^M" ; Define local string variable
string LastName = "Garvin^M"; Define local string variable
  call SendData with FirstName; Call SendData procedure,
                  ; passing it the value of FirstName.
  call SendData with LastName; After execution returns to
                  ; Main, call SendData again passing
                  ; a new value.
endproc              ; End of Main procedure.

;***********************************************************
; SendData procedure.
; Calls:
; Called by: Main.
; Modifies the value of the following globals:
;***********************************************************
proc SendData        ; Procedure called by another proc.
param string szText  ; Define a local variable to
                ; contain the value passed from another
                ; procedure.
  transmit szText    ; Send the value of the local variable
                ; "szText", which is equal to the value
                ; passed in from calling procedure.
  usermsg "Done!"     ; Message box.
endproc               ; End of SendData procedure, execution
                      ; returns to caller.
```

Notice the command clause "**param string**," instead of "**string**" in **SendData**'s variable definition. **param** tells ASPECT that a procedure requires the calling procedure to pass it a variable or constant of the specified type. In this case, **SendData** expects a **string** because the **param** command is followed by "**string**." Similarly, we can also pass **integer**s, **long**s and **float**s by changing the type definition that follows the **param** command.

As you can see, the comments at the head of each procedure make it clear "who is calling whom." They can also document global variables that are affected by a procedure or explain what a procedure does. This form of documentation, while not required, is excellent programming practice.

Write a procedure called ***DelaySend*** *that can be passed a* ***string*** *from any other procedure.* ***DelaySend*** *should use the command "****pause*** *1" to delay a second before transmitting the string. You can test your procedure by creating a* ***main*** *procedure that calls* ***DelaySend****.*

Now that you've learned about procedures and user defined variables, let's look at "*Controlling Program Flow*."

Controlling Program Flow

A program in any structured programming language relies heavily on special keywords that control the flow of events. While many logon scripts are linear, more complex scripts perform activities based on conditions.

To understand conditionals that control program flow, it's necessary to understand the meaning of the terms "true" and "false." For the purposes of logic within ASPECT, "true" means "non-zero" while "false" represents a value of zero. The expression "5 == 4" is, logically, the same as zero, or false in ASPECT. On the other hand, "5 == 5" is logically true, or the same as a non-zero value.

Simple Conditionals

The simplest type of conditional expression uses the **if ... endif** construct:

```
proc Main
```

```
integer Test        ; Define variable named "Test"
                    ; Code that manipulates value of Test.
  if Test == 5      ; See if the variable now equals 5.
    call DoThis     ; If true, call a particular procedure.
  endif             ; End of the conditional.
endproc
```

Although the **if ... endif** construct can be used in complex decision trees, at its basic level it is a simple question:

```
  if Test == 5      ; If Test equals value 5, THEN
    call DoThis     ; ... call our procedure.
  endif             ; End of question
```

More complex variants on the **if ... endif** construct include **if ... else ... endif** and **if ... elseif ... endif**. **switch** ... **case** is a related command structure. **switch** ... **case** constructs are also used in several examples in other areas of this tutorial.

Loops

Many of the sample scripts in this tutorial include some variation of this script fragment:

```
#define TRUE 1
proc Main
  .
  .         ; Other code here.
  .
  while TRUE    ; Loop while true.
  endwhile
endproc
```

These lines cause the program to process indefinitely, until it is halted or until Procomm Plus is terminated. Why? **while** is a loop command. As long as the expression after the **while** command evaluates to non-zero, or true, everything between the **while** and **endwhile** executes over and over. When the expression after **while** evaluates to false, or 0, execution jumps to the line after the **while** ... **endwhile** construct. Since the expression “1” is true, the **while** TRUE / **endwhile** lines of code above loop endlessly. If you sandwich a **transmit** command between the lines, the script will transmit the same string over and over again:

```
  while TRUE
```

```
    transmit "hi"; This would look like hihihihihihihihihi...
  endwhile
```

Often the code in a **while** ... **endwhile** loop changes the value of the expression from true to false, causing the loop to execute a specific number of times:

```
proc Main
integer Counter = 0       ; Initialize "Counter" to 0
  while Counter != 5      ; This means while counter does
                  ; *not* equal 5".
    Counter++             ; Increment the value of counter
                  ; (add 1 to it)
    transmit "hi"         ; and transmit "hi".
  endwhile
  transmit "^M^J"         ; Send a CR/LF combination
  transmit "bye"          ; and then the word "bye".
endproc
```

In this example, the starting value of ***Counter*** is 0. The **while** expression evaluates to true, since ***Counter*** does not equal 5, and so the code inside the loop is executed. ***Counter*** is incremented to 1 and the word "***hi***" is transmitted. Execution loops back to the **while** statement. The variable ***Counter*** equals 1 now, which is still not 5, so the code inside the loop is executed again. This looping process continues until the value of ***Counter*** is 5. When the value of ***Counter*** is 5, the expression after the **while** keyword becomes false and execution jumps past the loop to the next line of the script. Thus, what gets transmitted by this fragment of code looks like this:

```
hihihihihi
bye
```

You can also nest loops, making loops within loops:

```
proc Main
integer Test1 = 1,Test2    ; Define and initialize "Test1".
  while test1 != 5         ; Create outer loop.
    Test2 = 1              ; initialize another variable
    while test2 < 10       ; Create inner loop.
                  ; Notice use of less than (<).
      test2 ++             ; Increment test2
```

```
                    ;  This line is processed 36 times.
    endwhile            ; End of inner loop.
                    ; This line is processed 4 times.
    test1 ++          ; Increment test1
  endwhile              ; End of outer loop.
endproc
```

The above example, as noted in the comments, causes the inner loop to be accessed thirty-six times, and the outer loop four. For more information on conditional expressions and looping, see the **for** command and the **while** command.

*Write a script that sends "AT^M" to your modem ten times, with a one second pause between each transmission. Try to make your script a maximum of eight lines, including the **proc main** and **endproc** statements.*

Functions

ASPECT supports user-defined functions as well as procedures. Functions and procedures are almost identical. They are isolated pieces of code that perform specific operations, sometimes changing the value of data that is passed to them. Unlike procedures however, functions return values. For instance, a function can return the square of a number:

```
integer Num = 2
  Num = Square(Num)     ; Assuming the user-defined function
                ; "Square" works to square an integer,
                ; it returns the value 4, so Num now
                ; has the value 4.
```

As you can see, you don't need to explicitly **call** and pass arguments to a function using the **call** ... **with** statement. Instead, a list of parameters can be passed to a function inside of parentheses. As we'll see in "*Calling Conventions*" on page 474, this is true of procedures as well.

Functions not only behave differently from procedures, but they also look different. They start with the keyword **func** rather than **proc** and, as you might expect, they end with an **endfunc** rather than an **endproc**. Also, the first line of a function must define the type of value it returns. Here's the square function we referred to earlier:

```
func Square: integer     ; This func returns an integer value.
```

```
param integer Number     ; Variable for value passed to Square.
  return Number * Number; Return the new value of number.
endfunc                  ; End of func Square.
```

And here's a script that includes a function to add a carriage return to a string:

```
;****************** MAIN PROCEDURE *******************
proc Main
string FirstName = "Ebenezer"  ; Define string for first name.
  FirstName = AddCR(FirstName); Get "Ebenezer^M" from AddCR.
  waitfor "Name: "          ; Now you can wait for prompt
  transmit FirstName        ; and send the string out.
endproc

;****************** ADDCR FUNCTION ******************
func AddCR : string         ; Must include return type!
param string szText         ; Define variable "szText"for
                            ; passed string.
  strcat szText "^M"        ; Add a CR to received string
  return szText             ; and return it to calling proc.
endfunc
```

This **AddCR** function is general enough that you can use it in a variety of situations, so you may wish to add it to a "library" file of commonly-used procedures and functions that can be **#include**d in your other scripts. Save the **AddCR** function to a file called **library.inc**.

To create a library, or **#include** file, simply open a new text file, and enter the functions, procedures, macros or variable definitions you want it to contain. Save the file to the directory containing the source files that will reference the **#include** file. Any file name and extension can be used for an **#include** file. To help maintain consistency, however, it is common practice to use the **.inc** extension.

Let's say that you are working on a bulletin board logon script called **callbbs.was**. At the beginning of the script, before any procedures, insert the line:

```
#include "library.inc"
```

Now you can use the ***AddCR*** function in your **callbbs.was** file even though the function itself is in **library.inc**! You can include **library.inc** in any script you want to write.

There are two important considerations to keep in mind when working with functions:

- Functions are always user-defined. There are no ASPECT commands that can be called in the manner that functions are called.
- Functions can be used in command expressions anywhere that a constant value can be used, including arguments to other procedures and functions.

For example, the lines:

```
Answer = Num + Square(Num); OK to use functions in expressions.
```

and

```
transmit AddCR(Name)     ; OK to use function as argument
```

are perfectly acceptable, but the following line is not:

```
strcat AddCR(Name) "^L"   ; WRONG!
```

The third example doesn't work because the ASPECT **strcat** command expects a string variable as the first parameter, but functions can only be used where constants are allowed.

A. *Write a script that prompts you for an input string, adds an exclamation point to the string and displays it.*

Hint: *Use the* ***sdlginput*** *command to get the string, and the* ***usermsg*** *command to display it.*

B. *Write a script that prompts you for two numbers, adds them, and returns the result. Display the result using* ***usermsg*** *with the "%d" format string.*

C. *Write a function that prompts for a string and reverses the contents of the string.*

Hint: *You may find the* ***strrev*** *command helpful!*

Now that you're familiar with the use of multiple procedures and functions, let's examine "*Calling Conventions*."

Calling Conventions

"*Procedures: the Basics*" on page 466 and "*Functions*" on page 472 may have given you the impression that functions are called differently from procedures. ASPECT supports two methods of calling procedures and functions that are interchangeable. You can use the **call** ... **with** syntax, or drop both keywords and include the argument list in parentheses. For example, suppose we had a **call** to a procedure as:

call SomeProcedure with FirstVar, SecondVar, ThirdVar

We can use the shortened syntax and enhance the readability of our script with:

SomeProcedure(FirstVar, SecondVar, ThirdVar)

The shortened function call is similar. Instead of:

**call SomeFunction with FirstVar, SecondVar, ThirdVar INTO **

Results

We can use:

Results = SomeFunction(FirstVar, SecondVar, ThirdVar)

You can use either calling form. However, we recommend the shortened form because it's simpler and more readable. This is the calling method we will use in the more complex sections of the tutorial.

Passing Variables by Reference

As stated in "*Local Variables*" on page 461, a local variable can only change within the procedure that defines it. Globals are accessible to all procedures in a script file and can, therefore, be modified at any point in a script. It is possible to pass a local variable from one procedure to another, have the second procedure change its value and pass back the changed value. This is accomplished by prefixing the ampersand character (&) to the name of the passed variable, a process known as passing by reference:

```
proc Main
integer Number = 0       ; Define a number variable.
OtherStuff( &Number )    ; Call OtherStuff passing the
                         ; variable Number "by reference".
  usermsg "%d" Number    ; Display new value. Should be 1 now.
endproc

proc OtherStuff
param integer Num        ; Variable for passed value.
  .
  .                      ; The procedure has other commands...
  .
  Num++                  ; But one command increments the
```

```
                  ; value passed by the calling proc.
endproc
```

This is a trivial example, and it's possible to achieve the same results with a function or a global variable. However, if you think of the above fragment in the context of a longer script with many procedures, a global variable might be risky. You might lose track of the various situations in which you've accessed a global variable, and have a procedure change its value in a way that adversely affects another part of the script. Additionally, a function may not be appropriate if it performs more than one task, and any of its tasks inconveniently change the values of the passed arguments. In the case above, a variable passed by reference is the safest and most elegant solution.

Notice the use of "%d" in the ***usermsg*** *command to display the value of an integer variable in the formatted string.*

Rewrite the "Carriage Return" script in the section on functions so that it uses a variable passed by reference, instead of a function.

SUCCESS and FAILURE

A number of ASPECT script commands set the value of two special system variables, SUCCESS and FAILURE. They can have the value of either 1 for TRUE or 0 for FALSE. If a command executes successfully, the value of SUCCESS is 1 and FAILURE is 0. If a command fails, the value of SUCCESS is 0 and FAILURE is 1. SUCCESS and FAILURE are used in conditional expressions:

```
  if SUCCESS        ; Meaning "if SUCCESS is set to 1".
    .
    .            ; Do something.
    .
  endif
```

A typical use of SUCCESS and FAILURE is after **waitfor** commands:

```
proc Main
  waitfor "CONNECT 28800" 60; Wait 60 seconds for a connect
                    ; message to appear.
  if FAILURE             ; If the waitfor doesn't succeed
    usermsg "Try again later!"; let the user know it failed.
  endif
endproc
```

Another interesting use of SUCCESS and FAILURE is after a file transfer command, either **sendfile** or **getfile**, to see if the transfer started:

```
proc Xfer
  sendfile zmodem "abc.zip" ; Send the file abc.zip by zmodem.
  if SUCCESS             ; If upload started successfully,
    usermsg "Transmitting file..." ;say so
  else
    usermsg "Sorry - try again!"
                     ; Otherwise, put up error message.
  endif
endproc
```

ASPECT even allows an abbreviation of this status-checking method. Any command that sets SUCCESS and FAILURE can be used directly in a conditional expression. The value that is returned to SUCCESS and FAILURE is returned to the conditional; you don't have to explicitly check the variable. For example:

```
proc Main
  if not waitfor "CONNECT 2400" 60 ; Wait for connect message.
    usermsg "Try again later!"    ; If one isn't received,
  endif                      ; display an error message.
endproc
```

A. *Look through "The ASPECT Commands", starting on page 65, for commands that set the SUCCESS and FAILURE system variables. Commands that set SUCCESS and FAILURE have their descriptions specially marked with an "****S/F****" bullet:*

A A command description.

B. *Write a simple logon script in such a way that it provides some error-checking by using these system variables or by using the commands in conditional expressions.*

Getting User Input

Suppose you want to automate your logons with ASPECT, but for security reasons you don't want the script to transmit your password automatically. ASPECT has many ways to prompt you for input with a "dialog box," the simplest of which is an input dialog box created with the **sdlginput** command. The ASPECT commands for a standard dialog box customized for the input of a user password looks like this:

```
;************************************************************
; Main procedure.
; Calls: SendData.
; Called by:
; Modifies the value of the following globals:
;************************************************************
proc Main
string szPassword     ; string variable the user enters in
                      ; input box. Its value is passed to the
                      ; SendData routine.
  ; The following line creates the box for user input. The
  ; first string appears in the title bar of the box; the
  ; second is the prompt string that appears right before the
  ; text entry field itself. The variable at the end takes on
  ; the value of the string the user enters.
  sdlginput "Password Entry" "Enter your password:" szPassword
  waitfor "Password:"          ; Wait for password prompt.
  SendData( szPassword )
  ; Call the SendData routine with the value of the local
  ; string variable "szPassword". SendData adds a carriage
  ; return and sends user's password out the port.
endproc

;************************************************************
; SendData procedure. Adds a carriage return to passed string
; and then transmits it.
; Calls:
; Called by: Main.
; Modifies the value of the following globals:
;************************************************************
proc SendData
param string szText     ; Local variable to receive
                        ; value passed from another procedure.
  strcat szText "^M"    ; Strcat appends a string onto the
                        ; end of another. In this case a "^M"
                        ; is added to the end of the password.
  transmit szText       ; And send the concatenation.
endproc
```

Write a script that prompts you for your first name, your last name, and concatenates the two strings with the ***strcat*** *command and displays your full name in a single message box. Be careful to divide your script into sensible procedures.*

Hint: *You may end up with procedures named something like* ***GetFirstName*** *and* ***GetLastName****. Don't forget to embed a space between the first and last names!*

A Dialog Box Example

In the last example we created a very simple dialog box for string input using the **sdlginput** command. You can also create more complex dialog boxes that allow you to input information by means of edit boxes, buttons, check boxes, radio buttons, and other controls. If you've used many Windows programs, including Procomm Plus, you've entered information using several kinds of dialog boxes and controls.

The easiest way to create complex dialog boxes in ASPECT is to use the *Dialog Editor*. By default, you can access the *Dialog Editor* from ***Start | Programs | Procomm Plus | Procomm Plus Utilities***, or if you're already in Procomm Plus, you can select it from the ***Tools | Scripts*** menu. "*The Dialog Editor*" on page 510 covers this utility in detail. Of course, you can also "hand code" dialog boxes and achieve the same results, but that makes the process much more tedious!

The following example, if pasted into the *Dialog Editor*, creates a dialog captioned “**Dialog Example**” that contains three **radiobutton**s, labeled ***First***, ***Second***, and ***Third***, and a **pushbutton** labeled ***OK***.

```
dialogbox 0 22 118 99 103 11 "Dialog Example"
    radiogroup 100 WhichButton
      radiobutton 1 25 22 42 10 "&First"
      radiobutton 2 25 39 42 10 "&Second"
      radiobutton 3 25 56 42 10 "&Third"
    endgroup
    groupbox 4 21 8 51 65 "Button Choices"
    pushbutton 5 27 80 40 14 "E&xit" default
enddialog
```

This fragment of code, called from a script, paints the dialog box on the screen. The various numeric arguments control the sizing and placement of their corresponding objects. They can be adjusted manually with the *ASPECT Editor*, or the *Dialog Editor*. For details on these arguments, please refer to “*The ASPECT Commands*” on page 65.

Now that we have the code for a dialog box, let’s create a procedure that will display it:

```
proc Main            ; Beginning of Main procedure
  MakeDialog()         ; Call the dialog procedure. When
  pause 5            ; execution returns, pause for 5 sec
endproc              ; and exit the script file.


proc MakeDialog
  integer WhichButton    ; This variable’s value is based
                   ; on which radio button is selected.
  dialogbox 0 22 118 99 103 11 "Test"
    radiogroup 100 WhichButton
      radiobutton 1 25 22 42 10 "&First"
      radiobutton 2 25 39 42 10 "&Second"
      radiobutton 3 25 56 42 10 "&Third"
    endgroup
    groupbox 4 21 8 51 65 "Button Choices"
    pushbutton 5 27 80 40 14 "&Exit" default
  enddialog
endproc                                ; Return to proc Main.
```

The call to **MakeDialog** causes script execution to jump to that procedure. **MakeDialog** paints the dialog you designed, then returns control to the **main** procedure.

Currently, our program does nothing except briefly display the dialog box on the screen. In order to be useful, let's modify the script so that, when run, it displays a dialog and waits for the user to select one of the **radiobutton**s, and click ***OK***. Then, let's have the script display a message box reporting the button the user selected.

Of course, this script still isn't very useful, but it's a good building block that you can use to perform a variety of tasks.

Each dialog control has its own unique *ID* that is returned by the **dlgevent** command when you select that control. If you click on the **pushbutton** in the example above and read the result with the **dlgevent** command, ASPECT returns the ID of the ***OK*** **pushbutton**, which is 5. Let's take a look at our script file and add some commands to handle the dialog events and display the results.

```
#define TRUE 1
integer WhichButton   ; Global variable used for radiobuttons.
;*************************************************************
; MAIN
; The Main procedure calls MakeDialog to construct and display
; a dialog. It waits in an endless loop for a dialog event.
; When a dialog event occurs (i.e., when the user chooses a
; radio button or presses the pushbutton) CheckIt is called.
; Calls: MakeDialog, CheckIt, ShowIt
; Modifies global variables: None
;*************************************************************
proc Main              ; Start of Main procedure.
integer Event          ; Variable used for dialog events.
  MakeDialog()         ; Show the dialog box.
  while TRUE           ; Loop and wait for an event.
    dlgevent 0 Event   ; Get any event for our dialog box.
    switch Event       ; Evaluate the dialog event.
      case 5           ; Corresponds to the Exit button.
        exitwhile      ; exit case or the OK pushbutton.
      endcase
    endswitch
  endwhile
  ShowIt()             ; Call routine to display selection.
endproc                ; End of Main procedure.
;************************************************************
; MAKEDIALOG
; MakeDialog constructs and displays the dialog box. It then
; returns to Main.
; Calls:
; Called by: Main
; Modifies global variables: WhichButton (initializes it to 0)
;************************************************************
proc MakeDialog
  WhichButton = 0       ; Initialize WhichButton variable.
    dialogbox 0 22 118 99 103 11 "Test"
      radiogroup 100 WhichButton
        radiobutton 1 25 22 42 10 "&First"
        radiobutton 2 25 39 42 10 "&Second"
        radiobutton 3 25 56 42 10 "&Third"
      endgroup
      groupbox 4 21 8 51 65 "Button Choices"
      pushbutton 5 27 80 40 14 "&Exit" default
    enddialog
endproc                ; Return to proc Main.
;************************************************************
; SHOWIT
; ShowIt checks the value of the global variable WhichButton
; after the pushbutton is pushed. WhichButton is 1 if the first
; radio button was selected, 2 if the second was selected, 3 if
; the third was selected. A different message is displayed
; based on this value.
; Calls:
; Called by: Main
; Modifies global variables:
;**************************************************************
proc ShowIt
string DisplayString      ; Local variable for display in
                          ; message box.
  switch WhichButton      ; Evaluate the WhichButton variable.
    case 1
      DisplayString = "The First radio button"
    endcase
    case 2
      DisplayString = "The Second radio button"
    endcase
    case 3
      DisplayString = "The Third radio button"
    endcase
  endswitch
  usermsg " %s was selected." DisplayString
endproc
```

Notice the use of a format string in the **usermsg** command above. The format specifier “%s” is replaced in the sentence by the value of the variable ***DisplayString***. For more information on the format string, see “*Formatting Text and Data*” on page 49.

***A.** Write a script that displays a dialog box with at least three different controls and displays the value returned by the* **dlgevent** *command when the controls are selected.*

B.** Write a procedure similar to the* **sdlginput** ASPECT *command that displays a dialog box with an* **editbox, title, and* **pushbutton***. To make the job easy, use the Dialog Editor to create your dialog box.*

While dialogs are a common interface for gathering input, you may want to see "*A User Window Example*" for another method of interacting with the script's user.

A User Window Example

The graphical capability of Procomm Plus isn't limited to dialog boxes. You can create *User windows* within the *Terminal* workspace. *User windows* can contain **metafile**s, **bitmap**s, **icon**s, **iconbutton**s, **pushbutton**s, **hotspot**s, and various other graphical elements. Some of these objects can be created interactively in the *User Window Editor* utility, which operates similar to the *Dialog Editor*. Others can already exist in other graphic or executable files, and only their source and placement needs to be specified in the script. For additional details, see "*The User Window Editor*" on page 535.

By positioning graphic objects in a *User window* you define, you can create custom applications that use the Procomm Plus communications engine, yet have your own look and feel. Since you're rapidly becoming a master at reading and writing ASPECT scripts, let's use what we've learned and the various *User window* commands to create an icon bar on the left side of the *Terminal window*. While this is similar to an *Action Bar*, it is created wholly by ASPECT, and is not a **.pwb** file.

```
#define TRUE 1
string ABARLeft            ; Name of the left icon bar.
proc Main
  fetch actionbar left ABARLeft; Get current left Action Bar.
  set actionbar left NONE     ; Turn off the left Action Bar.
  when USEREXIT call ResetAbar ; Wait for a user exit event.
    set aspect path $PWTASKPATH
            ; Create a user window on which to display our
            ; icons for our left icon bar. Procomm Plus
    ; stores its icons in pw4icons.DLL.
uwincreate LEFT PIXELS 42 128 128 128 BITMAP
  iconbutton 1 0 0 "" "pw4icons.DLL" 5 BACKGROUND
  iconbutton 2 0 390 "" "pw4icons.DLL" 28 BACKGROUND
  iconbutton 3 0 780 "" "pw4icons.DLL" 27 BACKGROUND
  iconbutton 4 0 1170 "" "pw4icon2.DLL" 36 BACKGROUND
  iconbutton 5 0 1560 "" "pw4icons.DLL" 83 BACKGROUND
  iconbutton 6 0 1950 "" "pw4icons.DLL" 84 BACKGROUND
uwinpaint           ; Draw the new user window.
while TRUE          ; Loop forever.
  switch $OBJECT      ; Evaluate the uwin event.
    case 1          ; Connection Directory.
      sendkey ALT 'D'
    endcase
    case 2          ; Setup selected.
      sendkey ALT 'S'
      sendkey ALT 'S'
    endcase
    case 3          ; Scrollback selected.
      sendvkey 0x0C26
    endcase
    case 4          ; Hangup selected.
      hangup
    endcase
    case 5          ; Send file selected.
      sendkey ALT 'A'
      sendkey 'S'
    endcase
    case 6          ; Receive file selected.
      sendkey ALT 'A'
      sendkey 'R'
```

```
        endcase
      endswitch
  endwhile              ; Bottom of loop.
endproc                 ; End of proc Main.

proc ResetAbar
  set actionbar left ABARLeft; Restore the left Action Bar.
  exit
endproc                 ; End of ResetAbar procedure.
```

In our example, we use the **uwincreate** command to define our *User window*. This command tells Procomm Plus what the *User window* should look like on the *Terminal* workspace. In this case, we're telling ASPECT that we want a *User window* that's 42 pixels wide with a gray background.

Take a look at the **iconbutton** commands following **uwincreate**. Unlike dialog box commands, *User window* commands are not restricted to code structures or clauses, and can appear anywhere after the **uwincreate** definition.

$OBJECT is a system variable that contains the ID of the last *User window* object selected. If no *User window* object has been selected, the value of $OBJECT is false or zero. We use a **switch** ... **case** construct to evaluate the state of $OBJECT and the **sendkey** and **sendvkey** commands to emulate what happens when an icon is selected from a real *Action Bar*.

Read the descriptions for **bitmap**, **hotspot**, **icon**, **iconbutton**, **metafile**, **pushbutton**, **uwincreate**, **uwinpaint**, and **uwinremove** for more information about *User windows* and their objects.

A. *Modify the sample above so that it displays an icon bar on the bottom of the screen instead of the left side. Use the User Window Editor if you like.*

B. *Write a script that puts a Windows* ***.bmp*** *file in the User window along with a* ***hotspot*** *that, when selected, exits the script.*

The when ... call Command

Microsoft Windows is an event-driven environment. This means that programs perform tasks when specific events occur. For example, Windows tells programs when they should draw graphics on the screen, when someone clicks on a button or icon, and even when they are allowed to use a device on your system. ASPECT takes advantage of this event-driven design with the **when** command.

when allows your script to watch for events while it's performing other activities. For example, suppose that you're connected to a host system that displays a prompt after every 24 lines of information and waits for you to press <Enter>. You want Procomm Plus to press <Enter> for you when this event occurs. We can accomplish this easily with the **when** command. Examine the script example below:

```
proc Main
; Issue a when command that watches for the prompt and calls
; another procedure to send a carriage return.
  when TARGET 0 " Press Enter " call PressEnter
  while $CARRIER          ; Loop while connected.
    yield             ; Yield script processing time
  endwhile
  when TARGET 0 CLEAR      ; Clear the when clause.
endproc


proc PressEnter
  transmit "^M"           ; Send a carriage return.
endproc
```

In our example, we define a **when** command to watch for the prompt from the remote system. We also suspend the script's progress while we're on-line or connected using the **while** ... **endwhile** loop. Notice that we use the TARGET option to tell ASPECT what type of event to watch for. In this case, TARGET specifies that we're expecting to receive a text string. The number following the TARGET option is the index of this **when** clause. Our script file sends a carriage return any time Procomm Plus receives the string "***Press Enter***".

There are many events besides **when** TARGETs that you can watch for in your script files. You can watch for dialog events, key presses, user exits, or even sense when incoming information has stopped. Let's take a look at each of the **when** commands that you can use in your script files:

when DIALOG 0 call ProcessEvent

This is an example of using the **when** command to check for dialog events. Every object within a dialog is identified by an ID. When you click on the dialog's object with an ID of 0, your script calls a procedure named **ProcessEvent**. **ProcessEvent** reads and evaluates the value returned by the **dlgevent** command.

Dialog boxes are discussed in more detail in "*A Dialog Box Example*" on page 479. Let's take a look at a **when** clause that watches for a key press:

when ISKEY 0 'A' call AKeyPressed

This **when** clause tells ASPECT to **call** the procedure named ***AKeyPressed*** each time you press the "***A***" key on your keyboard. As with **when** TARGETs, an index is used to differentiate multiple **when** ISKEY clauses. Use the index to selectively clear your **when** ISKEY clauses. **'*A*'** is a constant that the compiler replaces with the value of the "***A"*** key when the script is compiled. '***A***' can be replaced by other virtual key values.

when QUIET 10 call Done

when QUIET allows a script to **call** a procedure when the communications line is quiet for a specified number of seconds. In the line above, we're telling our script file to **call** a procedure called ***Done*** when the line is quiet for 10 seconds. If data is received during each 10 second period, ***Done*** is never called. If no data is received for over 10 seconds, ***Done*** is called.

On the other hand, we may want to **call** a procedure in timed intervals. Suppose we need a script that calls a procedure every 30 seconds and sends a space character. Our script file might contain the following line:

when ELAPSED 0 30 call SendSpace

In this example, our script will call a procedure called ***SendSpace*** every 30 seconds. The ***SendSpace*** procedure is used to **transmit** the space character.

Finally, **when** clauses can be used to sense when users try to exit our script files. Look at the following line:

when USEREXIT call CleanUp

USEREXIT tells ASPECT that you want to call the specified procedure every time the ***Stop Script*** *Action Bar* button is selected, or when the user selects ***Stop Script*** from the ***Tools | Scripts*** menu. We could use ***CleanUp*** to perform final operations to insure that our script exits properly, restoring *Setup* options that it changed, or closing files. It's important to remember that you must use the **exit** or **quit** command to stop your script if you have created a **when** USEREXIT clause. Otherwise, the script keeps running even though the user tried to stop it!

*Create a script file that prompts you with a confirmation message when you try to **exit** the script.*

***Hint.** Use the **sdlgmsgbox** command to make the message box handling a little easier.*

System Variables and the when Command

In addition to watching for incoming text, key presses, dialog events, and so forth, you can use **when** commands to detect when the values of ASPECT system variables change. For example, suppose we want a script file that displays the position of the mouse pointer on the status line, but we only want to update the status line when the position of the mouse pointer changes. We need two **when** commands that check the status of the $POINTERX and $POINTERY system variables. Our **when** commands would look like this:

```
when $POINTERX call PointerChanged
when $POINTERY call PointerChanged
```

Since we don't care which of the two values changes, we may as well **call** the same procedure in both cases. $POINTERX contains the X location of the mouse cursor and $POINTERY contains the Y location. When $POINTERX or $POINTERY changes, our script **call**s the procedure named ***PointerChanged***. In ***PointerChanged***, we can display the values of $POINTERX and $POINTERY on the status line. Our resulting script file might look like:

```
#define TRUE 1
proc Main
  PointerChanged()      ; Display beginning message on the
                   ; status line and set up our whens.
  when $POINTERX call PointerChanged
                   ; Watch the $POINTERX value.
  when $POINTERY call PointerChanged
                   ; Watch the $POINTERY value.
  while TRUE          ; Loop forever.
    yield
  endwhile
endproc
```

```
proc PointerChanged       ; Procedure to display pointer
                          ; location on the status line.
  statmsg "( %d, %d )" $POINTERX $POINTERY
endproc
```

In our script, ***PointerChanged*** is called at the beginning to display the mouse pointer's location before entering the endless **while** ... **endwhile** loop at the end of the **main** procedure. If we don't make this **call** at the beginning of our script file, the status line message doesn't appear until after the user has moved the mouse. Thereafter, ***PointerChanged*** is called anytime the value of $POINTERX or $POINTERY changes. **statmsg** is used to display the position of the mouse pointer on the status line.

Remember, ASPECT can detect changes to the value of any ASPECT system variable. Be aware that some system variables are reset to their default value when they are accessed. If you don't access the system variable within the called procedure, the system variable will automatically reset to prevent the procedure from being continuously called. "*System Variables*" on page 427 contains more detailed information on each of the system variables.

See "*Packet Send and Receive*" on page 502 for additional example of how to use system variables in conjunction with the **when** command.

Sending Keys with ASPECT

In the script within "*A User Window Example*" on page 484, we used the **sendkey** command. **sendkey** sends a keystroke to the currently active window using that keystroke's keycode. ASPECT also provides a command called **sendkeystr**. As you might expect, **sendkeystr** sends a string. Examine the following script fragments:

```
sendkey 13
sendkeystr "^M"
```

Both fragments send a Carriage Return to the active window. With **sendkeystr**, however, you could also send a string of several characters:

```
sendkeystr "It's coming down the wire!^M"
```

With **sendkey**, on the other hand, you can send not only regular character keys, but **Alt**, **Shift**, and **Control** keys as well. For instance:

```
sendkey ALT 70
```

sends <Alt><F> to the currently active window, just as if you'd typed it at the keyboard. If the *Terminal window* is currently active, the command pulls down the Procomm Plus ***File*** menu. Alternately, you can use the command syntax:

```
sendkey ALT 'F'
```

Notice the use of the single quotes around the character constant F. '***F***' is not a string; it is another way of expressing the integer value 70.

sendkey and **sendkeystr** can be used to send keystrokes to other applications, as long as the appropriate window is first activated. For instance, you may want to switch over to an editor and display a file. These two simple commands allow you not only to create a custom interface for Procomm Plus functions, but to build entire integrated systems! The following script loads Wordpad and searches for a string within the displayed file:

```
#define RUNPATH "C:\windows\write hints.txt"
proc Main
string szAreaCode="573"
  run RUNPATH
  pause 3
  sendkey ALT 'E'
  pause 3
  sendkeystr "f"
  pause 3
  sendkeystr szAreaCode
  pause 3
  sendkey 13
  pause 3
  sendkey 27
  pause 3
  sendkey 27
  pause 3
  sendkey ALT 'F'
  pause 3
  sendkeystr "x"
endproc
```

Note the use of the **pause** command. This command is not required, but included so that the individual commands may be obvious when you run the script.

Write a script file that prompts for a string from the user, launches Wordpad using the ***run*** *command and searches for that string in a particular file.*

Spawning and Chaining ASPECT scripts

ASPECT allows one script to call another in two ways: chaining and spawning. With the **execute** command, you can spawn a child script and return to the point in your parent script where you issued the command. This allows the creation of worker scripts containing procedures or functions that can be used in a variety of applications. When you spawn a script, only the values of the predefined global variables S0 - S9, I0 - I9, L0 - L9, and F0 - F9 are preserved. If you want to pass a value to a child script, use these predefined globals. Consider these sample scripts:

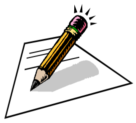
```
; File MAIN.WAS
proc Main
  S0 = "hello"              ; Assign S0 the value "hello"
  usermsg S0                ; and display it.
  execute "change.wax"      ; Call file that changes S0
  usermsg S0                ; and show the changed value.
endproc

; File CHANGE.WAS
proc Main
  S0 = "Goodbye"            ; Reassign S0 to "goodbye"
endproc                     ; and return to parent.
```

In addition to executing scripts, you can link to them with the **chain** command. When you **chain** to a script, control is not automatically passed back to the calling script when the **chain**ed script completes. If you used the **chain** command in the sample above, for instance, the “***Goodbye***” message would never display.

Chaining and executing scripts is useful in building complex ASPECT systems and in creating reusable modules. There are important considerations, however, when working with **chain**ed or **execute**d script files. For example, a parent script’s dialog boxes are destroyed and any active DDE client sessions are terminated when you **execute** or **chain** to another script.

➥ *For more information, see the **chain** and **execute** commands. Also, see "Script Spawning and Chaining" on page 55.*



Write a script file that asks the user for a name and address and calls another script to format and display this information.

Some Notes on Design

If you've read the other sections in this tutorial, you've learned the basic structural elements of the ASPECT language. Before going on to more advanced topics, let's discuss various approaches to designing script programs.

If you sit down and start writing a very complex ASPECT script without any planning at all, you may become frustrated and dissatisfied with the results. ASPECT is a simple language to use, but if you randomly start experimenting with commands it's easy to get lost in its power. Here are some questions to ask yourself before you begin the actual coding:

- What is the purpose of the script?
- What are the necessary procedures to accomplish this purpose?
- What user input does the script need, and what is the best way to gather this input?
- Are there any necessary supporting procedures or functions? Can I reuse functions and procedures, either in this script or in other scripts, by generalizing them?

Suppose you want to write a script that downloads several files in the middle of the night when long distance phone rates are cheaper. Each night, you want to be able to specify exactly which files to download, and to tell the computer what time to place the call. In addition, you want Procomm Plus to hang up the call after a specified amount of on-line time has elapsed. You might come up with the following answers to the above questions:

- Purpose of script: to call a BBS and download user-specified files within a user-specified time period.
- Necessary procedures:
  - ❖ Ask user for files to download, time to call, and maximum length of session.
  - ❖ Wait for right start time.
  - ❖ Dial the call.
  - ❖ Set up some sort of **when** statement so the script can monitor the available time. **call** a procedure to hang up line if time elapses.

  - ❖ Logon to BBS.
  - ❖ Transmit names of files to download.
  - ❖ Download files.
  - ❖ Logoff BBS if files finished in specified time period.
- ◆ The script needs the following user input:
  - ❖ Time to call.
  - ❖ Files to download.
  - ❖ Length of time before ending session.

  Taking advantage of **asptime.dll**, start and end times can be chosen as described in "*Using ASPTIME.DLL*" on page 616 via radio buttons or list boxes. The requested filenames can be entered in edit boxes in another dialog.
- ◆ Support procedures or functions:
  - ❖ Input file names must be error-checked for the number of characters in the filename and extension. This function or procedure is needed for every file name specified by the user. I can generalize it so that it's useful in other scripts.
  - ❖ File names are delimited by spaces or carriage returns when sent to the BBS. Once again, this is a general purpose function. Perhaps I can use the "***AddCr***" function discussed earlier in this section.
  - ❖ The script needs to check whether or not files are properly downloaded, and needs to report the status. This may end up as two procedures.

There are important elements you are leaving out at this point, but at least you have an idea of where to start your script. It is now possible to sketch out what you need to include in a **main** procedure, without writing any code:

```
start of Main procedure
  get user input    ; This procedure can call yet another one
              ; for the dialog box.
  process user input ; Need to process info from dialog,
              ; error check etc.
  show waiting message; While waiting...
  start of loop
    if time == start time
      go on       ; This "go on" procedure will call all
              ; other worker procedures and functions.
      when elapsed time reached hang up
              ; From this point on we wait for the
              ; specified session length to be reached
              ; Script will end in "hang up" procedure.
    endif
  end of loop
end of Main procedure
```

This rough “pseudo-code” is obviously not ASPECT syntax, and you haven’t defined any variables or worked on the procedures, but you have outlined the basic logic, which is often the most difficult task. Of course, you can change some of the basic logic. For instance, you may not want the on-line session to hang up in the middle of a transfer. At this point it is useful to ask yourself several new questions:

- What kinds of variables do I need? Do any have to be global?
- Are there any procedures that may be more efficient as functions? Can I use the return value immediately in an expression?
- Are there variables that should be passed by reference instead of value?
- Can I simplify my functions or procedures by using ASPECT system variables?
- What ASPECT commands are the right ones for the various tasks?

Programming in any language is an iterative process, meaning that you often have to go back to your code many times, refining it a little more each time. These iterations should not be shots in the dark. Each time you visit your script file, the logic should become clearer and clearer rather than more obscure.

In the example above, there is very little user interface. It may be useful to work on the user interface before the underlying foundation of the program. Often the process of designing the interface, called “*prototyping*,” raises new issues or clarifies the direction the script should take.

There is no magic formula for constructing good scripts, but there are some stylistic conventions that keep you from getting lost in your code:

- Comment extensively, more than you think is necessary.

  You may know your code intimately today, but two weeks from now it may make no sense at all without good comments.
- Indent consistently.

  ifs should line up with **endif**s, **while**s with **endwhile**s. As you nest your code, make sure to indent the same number of spaces or tabs for lines or blocks of code that are at the same level. Most programmers indent using 2 or 3 spaces.
- Use global variables only when necessary.

Globals can change in other procedures without you realizing it, leading to unexpected results, known as “bugs.”

- Name your variables descriptively.

  A carefully chosen variable name like ***MyFileNames*** communicates so much more than ***StrVar4***.
- Create reusable functions and procedures whenever possible.

There are several sample ASPECT files in your ASPECT directory that serve as good examples of coding style.

Advanced Topics

This section of the tutorial provides details about some of ASPECT’s lesser known, but very powerful, features. “*Using macro definitions*” on page 496, for example, shows you how to use macros within your scripts. “*DDE server operation*” on page 498 and “*DDE client commands*” on page 500 give examples of how you can exchange information between Procomm Plus and other Windows 95 applications. “*Packet Send and Receive*” on page 502 wraps up this section of the tutorial with commands that allow you to write your own file transfer protocol.

Using macro definitions

Macros are meaningful phrases or constants that make your ASPECT script source code easier to read and maintain. In addition, macros can also represent **call**s to functions, procedures, or internal commands. They are typically defined at the top of script files, before the **main** procedure, but may appear anywhere in scripts. First, let’s talk about simple macro definitions that give loops, **switch** ... **case**, and boolean operations more meaning.

A simple macro definition, sometimes called a manifest constant, is made up of a reference name and constant value. Reference names are used in your script files in place of the values they represent. During compilation, Procomm Plus replaces the reference names with their corresponding values. Macro definitions take on the following form:

#define Name Value

As with variables, try to make your macro names describe the values they represent. For example, we may call a macro that represents the maximum length of a name MAX_NAME_LEN. Use underscores to clarify the meaning of macros or to resolve naming conflicts with reserved words. The following script example shows how to use macro definitions to make a script easier to read:

```
#define MAX_ROWS $TERMROWS-1   ; Define a macro representing
                               ; the max number rows on screen.
```

```
proc Main
integer CurrentRow          ; Define an integer variable
                    ; we can use to count through
                    ; each row on the screen.
  for CurrentRow = 0 upto MAX_ROWS
                    ; Loop through all rows on
    termputs CurrentRow 0 "Awesome!"
                    ; screen and display a string
  endfor
endproc
```

When Procomm Plus compiles the script shown above, all references to MAX_ROWS are replaced with the value of the system variable $TERMROWS, less 1. In this case, we're using a macro to make the operation of the **for** ... **endfor** loop clearer. Let's say that we want the script to treat row 5 as the maximum row on the screen. Instead of changing all references to the system variable $TERMROWS to 5, we can simply change the reference in our MAX_ROWS macro.

ASPECT also allows you to define macros that represent functions, procedures, or ASPECT commands. This feature not only makes a script easier to read, it can save you some programming time. For example, imagine that we want a function that displays a message box with a stop sign icon and an ***OK*** button and we're going to call this function many times within our script file. Instead of using the **sdlgmsgbox** command with all of its required parameters, let's create a macro that calls **sdlgmsgbox** with the arguments we want. The resulting macro definition could look something like this:

```
#define DO_MSG(a) sdlgmsgbox "Script Message" a STOP OK I9
```

The parameter called "a" in the macro name corresponds to parameter "a" in the macro definition. "a" is a token, representing the information that's displayed using the **sdlgmsgbox** command. When a script is compiled with the above macro definition, DO_MSG(string) references are replaced with ASPECT's **sdlgmsgbox** command. In addition, the string used in the DO_MSG reference replaces the "a" token in the macro definition. This process is referred to as *Macro Expansion*. Here is a script example that uses the macro definition above:

```
#define DO_MSG(a) sdlgmsgbox "Script Message" a STOP OK I9
proc Main
  DO_MSG( "Display me!" )    ; Use our macro to display a
                    ; message using sdlgmsgbox
endproc
```

When the script is compiled, the _DO_MSG macro is replaced with:

sdlgmsgbox "Script Message" "Display me!" STOP OK I9

As with the simple macro definitions, using macro expansion provides you with an easy way to make changes. All we need to do to change the icon displayed in all of our message boxes is replace STOP with the icon we want to use: INFORMATION, EXCLAMATION, QUESTION, or NONE.

A macro can be defined with more than one parameter. To create a macro called _PUTCURSOR that locates the mouse pointer at a specific location, we can use the following definition:

#define PUTCURSOR(a, b) setpointer a b

"a" and "b" in the macro definition are replaced with "a" and "b" in the macro name when the macro is expanded. We've replaced references to the **setpointer** command with a more descriptive macro!

A. *Write a script that uses simple macros to reference all text strings used for displaying messages.* ***#define*** *these text macros in a separate file called "***strings.inc***." Use the* ***#include*** *command to include the text macros in your script file.*

B. *Imagine you're writing a large ASPECT script application and your script is going to be used in many other countries. Describe how macros can make translation of text strings displayed by your script file into other languages easier.*

Now, let's look at "*DDE server operation*" to examine how other Windows 95 applications can exchange information with Procomm Plus's DDE server.

DDE server operation

Procomm Plus supports Dynamic Data Exchange (DDE) both as a server and as a client. Using the program as a DDE server is not directly related to the construction of ASPECT script files, but it helps you understand client scripts if you have a good grasp of server operations as well.

Procomm Plus functions automatically as a DDE server. When it receives a DDE request from another Windows application, it processes it automatically. Typically, another program requests that Procomm Plus run a script file and return the value of a predefined global variable. For example, in Microsoft Excel for Windows 95, the formula:

=pw4|test.wax!i0

instructs Excel to establish a DDE link with Procomm Plus (pw4), run **test.wax**, and display the value of the predefined global integer variable i0 in the cell where the formula is entered. Since this formula sets up a **ddeadvise** link to the variable i0, any change made by a script file to the value of i0 is automatically displayed in the cell.

As an example, suppose that you have written an ASPECT script called **dow.wax** that calls a financial service, downloads a particular stock quote and assigns the value to the float variable **f0**. You enter the formula **=pw4|dow.wax!f0** in an Excel cell. Now each time **dow.wax** changes the value of **f0**, the spreadsheet is updated with the latest information!

In addition to running script files and determining the current value of the predefined global variables, a client Windows application may request Procomm Plus to execute a command. Procomm Plus, when acting as a DDE server, responds to the following commands received from a DDE client:

ASPECTCMD string

CAPTURE OFF | ON

**DIAL {DATA | FAX | VOICE | FTP | TELNET | WWW } [GROUP] **
name [DIALINGBOX OFF]

DIALLOAD filespec

EXECUTE filespec

GETFILE string [filespec] [FILEXFERBOX OFF]

HALT

HANGUP

PWEXIT

SENDFILE string [filespec] [FILEXFERBOX OFF]

TRANSMIT string

Normally only applications that include some sort of macro or scripting language are able to send DDE “executes” like those listed above to a DDE server. Read the client application’s documentation for information about its client DDE support.

As a server, Procomm Plus responds to two topics, “System” and the name of a script file. For an explanation of DDE server commands and topics, please see the Procomm Plus help system. To examine the commands Procomm Plus supports while acting as a DDE client, see “*DDE client commands*” on page 500.

DDE client commands

As mentioned in the previous section, many DDE requests and “executes” can be sent only by client programs that support a macro or scripting language. ASPECT is a powerful scripting language, and you can use it to send a variety of DDE requests to other Windows programs that are able to act as DDE servers.

The process of setting up a DDE link with a server program begins with the **ddeinit** command, where you specify an application name and a topic. You will need to read the server application’s documentation to see what topics it supports. The **ddeinit** command returns a unique DDE channel identifier into a long variable. It’s a good idea to test this command to make sure the connection is successful:

```
long DDEChan = 0
ddeinit DDEChan "excel" "sheet1" ; Initiate DDE link with Excel
if FAILURE                     ; If no link established,
  usermsg "Unable to establish link"; warn the user.
endif
```

Let’s say you want to send some DDE commands or “executes” to Excel. To do this you initiate a link using the “system” topic:

```
long SysChan = 0
 ddeinit SysChan "excel" "system"
```

The **long** variable ***Syschan*** assumes the value of a unique link identifier that you can use in **ddeexecute** commands:

```
proc SendCommands
long SysChan                      ; Var. for link handle.
  ddeinit SysChan "excel" "system"   ; Set up a link to Excel.
  if FAILURE                     ; If no link established,
    usermsg "Unable to establish link"; warn the user.
  endif
ddeexecute SysChan "[APP.RESTORE()]"; Commands that
  ddeexecute SysChan "[APP.MOVE(0,0)]"; Excel accepts to size
  ddeexecute SysChan "[APP.SIZE(530,180)]"; its window, format
  ddeexecute SysChan "[FULL(TRUE)]"  ; columns, etc.
  ddeexecute SysChan "[COLUMN.WIDTH(75)]"
```

```
endproc
```

Once again, the syntax of DDE server commands is specified in the documentation for the server application.

Now suppose you want to change the value of one of the cells in an Excel spreadsheet. For this, use the **ddepoke** command. To poke values to a spreadsheet cell, Excel requires that you use a DDE channel established using a topic that is the name of the spreadsheet. Examine the following script fragment:

```
long SysChan = 0              ; Identifier to Excel system.
long DDEChan = 0              ; Identifier to DDE channel.
  ddeinit DDEChan "excel" "test.xls"; Initialize DDE link with
  if FAILURE                  ; spreadsheet. If link fails
    usermsg "Unable to establish link"; tell user and
    exit                      ; leave.
  endif
ddeinit SysChan "excel" "system"  ; Establish execute channel.
ddeexecute SysChan "[SELECT(`"R1C1`")]"; Make A1 active cell.
                              ; Notice the backticks (`)
                              ; for literal quote marks.
ddepoke DDEChan "R1C1" "Test"    ; A1 now contains "Test".
```

At this point, you can send commands referencing the ***SysChan*** handle, and poke values into the spreadsheet using the ***DDEChan*** handle.

Use the **dderequest**, or **ddeadvise** command to request information from the client. The channel for this information in our example is the one established using the spreadsheet. In the line:

```
ddeadvise DDEChan "R1C1" s0
```

The value of the A1 cell is put into the variable s0. When the value of the R1C1 cell is changed, the new value is placed in s0 and the system variable $DDEADVISE takes on the value 1 to indicate that a change has occurred. In addition, the **when** command can trigger when the server reports a change. This is usually referred to as a DDE hot link:

```
#define TRUE 1
long DDEChan                  ; Handle to the DDE channel.
proc Main
  .
  .
  ddeinit DDEChan "excel" "sheet1"; Establish DDE link.
  if FAILURE                ; If no link established,
    usermsg "Unable to establish link"; tell user
    exit                    ; and leave.
  endif
  ddeadvise DDEChan "R1C1" s0   ; Set up hot link to R1, C1.
  when $DDEADVISE call ShowMessage; Call ShowMessage when the
                            ; cell is updated in Excel.
  while TRUE                ; Loop forever.
    .
    .
  endwhile
endproc

proc ShowMessage
  usermsg "The value of cell A1 has changed."
endproc
```

Optionally, you can specify an integer at the end of the **ddeadvise** command. When the server informs the client that the information associated with a **ddeadvise** has changed, the system variable $DDEADVISE returns the value of this integer. If you have several outstanding **ddeadvise** events, you can determine very easily which changes occurred.

When you no longer need to be advised of changes to server data, issue a **ddeunadvise** command. Likewise, when you have completed your DDE session issue a **ddeterminate** for each open channel:

```
ddeunadvise DDEChan "R1C1"
ddeterminate DDEChan
ddeterminate SysChan
```

Packet Send and Receive

Another interesting set of commands is the packet send and receive commands. These commands give you the ability to exchange data and commands using an error-free packet protocol that you define. You can, for example, develop a custom BBS that sends and receives packetized Internet mail messages, commands, and responses for on-line games or “doors,” and a file transfer protocol to send and receive files simultaneously!

Since this protocol is unique to Procomm Plus, both sides of the connection must be running Procomm Plus, typically with two different script files that interact with each other. In order to use the packet send and receive commands, you must first put both machines in packet mode with the command:

```
 pkmode ON txtimeout rxtimeout
```

where *txtimeout* and *rxtimeout* specify the timeout in seconds for transmitted and received packets, respectively. Setting the timeouts with the **pkmode** command sets the defaults for each transmitted or received packet, although it’s still possible to override these defaults by including an optional timeout parameter with packet sends and receives. A typical packet send looks like this:

```
integer UserCommand = 1      ; Command to pass to receiver.
string szText = "This is the first command."
                             ; Text to send to receiver.
integer Len = 26             ; Length of string to send.
 pkmode ON 30 60             ; Turn on packet mode.
 pksend UserCommand szText Len ; Send packet to receiver.
```

After performing a **pksend**, you check to see if the packet is properly transmitted by checking the value of $PKSEND. If the packet is successfully transmitted, you can send another. If not, you can resend the packet or perform some other task:

```
 when $PKSEND call ProcessSend
                        ; When the status of the $PKSEND
                        ; system variable changes, letting
                        ; you know the status of the send,
                          ; call a proc to see what happened.


proc ProcessSend        ; Determines the packet status
 switch $PKSEND         ; by checking $PKSEND
  case 0           ; If $PKSEND==0, send is in
  endcase            ; progress or idle.
  case 1           ; If $PKSEND==1, packet sent,
      SendNext()       ; the receiver has the packet.
    endcase
    case 2             ; If $PKSEND==2, a timeout has
                      ; occurred so warn user.
      DoTimeout()      ; (The receiver is dead or lost)
    endcase
    case 3             ; If $PKSEND==3, there were errors
                      ; during transmission.
      Notify(_COMERROR)  ; Tell us about the error.
    endcase
    case 4             ; If $PKSEND==4, the send was
                      ; cancelled by receiver.
      Notify(_CANCEL)   ; Report the cancel request.
    endcase
  endswitch
endproc
```

$PKSEND is assigned a value of 3 when there are UART errors, buffer over-runs, or when the ASPECT packet protocol is in an indeterminate state. These errors are usually an indication of a poor communications link. $PKSEND is assigned a value of 4 when a cancel sequence is received.

At the receive end, your script processes the incoming packet when it arrives. Take a look at this script fragment:

```
  pkmode ON 30 120             ; Turn on packet mode.
  when $PKRECV call ProcessRecv  ; Watch for incoming packets!
  while TRUE
    .
    .
    yield                      ; Yield processing time
  endwhile
  .
  .
  .
```

```
proc ProcessRecv        ; Determines the status of the packet
  switch $PKRECV        ; checking system variable $PKRECV.
    case 0              ; If $PKRECV==0, packet not received.
    endcase
    case 1              ; If $PKRECV==1, the packet was
      ReadPacket()      ; received OK, extract the
    endcase             ; information from it.
    case 2              ; If $PKRECV==2, a timeout has
      Notify(_TIMEOUT) ; occurred. Warn user about the
    endcase             ; error.
    case 3              ; If $PKRECV==3, errors occurred
      Notify(_COMERROR); during the receive. Warn user
    endcase             ; about the problem.
    endswitch
endproc
```

In this example, if the packet is received error free, $PKRECV = 1, and ASPECT automatically sends an acknowledgment to the sender, setting the sender's $PKSEND variable to 1. Typically, this prompts the other end to send another packet. Meanwhile, the information in the received packet can be processed in the ***ReadPacket*** procedure. The variable parameters to the **pkrecv** command contain the information sent by the other end. A ***ReadPacket*** routine might look something like the following:

```
proc ReadPacket
integer Command              ; Command sent with packet.
string RecvString            ; String sent with packet.
integer PktLength            ; Length of data in packet.
  switch Command             ; Switch case to evaluate the
     .                       ; command passed in packet.
     .
  endswitch
  ProcessData(RecvString)    ; Look at the string sent
                             ; with packet.
  switch PktLength          ; Possibly evaluate the
     .                 ; length of the packet to
     .                 ; perform some other actions.
  endswitch
endproc
```

In this example, you can key on the value of the command variable to display a **bitmap** or **metafile**, write the received string to a file, or perform some other type of processing on the received string. As you can see, this group of commands is very flexible. The packet send and receive support in ASPECT provides the tools necessary to develop your own error-free protocol for a variety of uses. The exact specification of your protocol is determined by how you choose to react to errors, the timing of sends and receives, and how integer and string data in the packets is interpreted.

A. *Write a **#include** file that contains macros representing each character in the ASCII chart. For example, _CR might stand for the carriage return character or ASCII 13. **Hint:** It's a good idea to use an extension other than **.was** for include files. **.inc** or **.h** are commonly used extensions for ASPECT's **#include**d files.*

B. *Write a script that captures a list of user names from a BBS and creates a report in a Windows word processor like Microsoft Word for Windows, WordPerfect for Windows, or Lotus Word Pro 96.*

C. *Write your own file transfer protocol in ASPECT.*

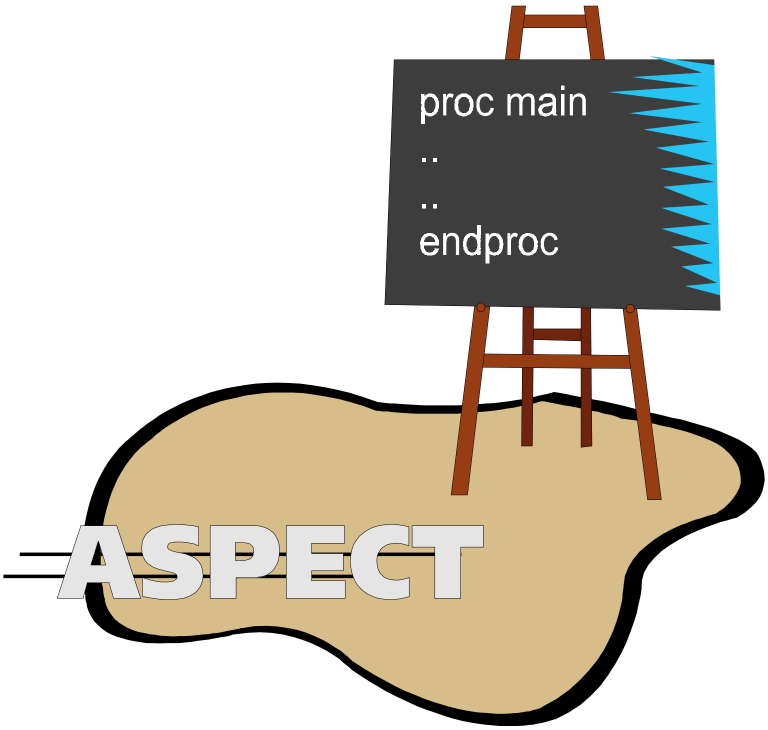
What's been covered?

In this tutorial you've learned the basics of ASPECT and seen some of its real-world applications. If you worked your way through the entire tutorial, you probably noticed that ASPECT can automate on-line sessions and many, more sophisticated tasks! With the added tools provided in the *Script Recorder, Dialog Editor,* and *User Window Editor,* everyone can tap into some of the power of ASPECT.

A tutorial can only scratch the surface of a powerful language like ASPECT. Some of the areas not covered by this tutorial include the string manipulation commands, the file I/O commands, the **set** and **fetch** commands, conditional compilation, and loading DLLs. Now that you've gone through the tutorial, look back at "*The Elements of ASPECT*" on page 19 and "*The ASPECT Commands*" on page 65 and study the commands and explanations in greater detail.

Chapter 7

The ASPECT Utilities



The Dialog Editor

The Procomm Plus *Dialog Editor* allows you to visually create and edit dialog boxes for use with ASPECT script files. Using this approach is much easier than coding a **dialogbox** statement group by hand, compiling and running the script, and verifying the appearance. Dialog boxes can:

- Serve as user input or option selection boxes.
- Report the status of an on-going process.
- Provide information to the user.

As you create a dialog and configure its controls, the *Dialog Editor* maintains the ASPECT script commands that will recreate the dialog within a script: this is a standard **dialogbox** statement group, which allows you to easily modify or even port the dialog between scripts. Once you've designed your dialog, there are several methods for importing and exporting dialogs into scripts.

"*Elements of the Dialog Editor*" on page 510 offers an overview of the components of the *Dialog Editor*'s window as well as general information about opening and saving dialogs. You may, however, wish to jump right in with "*The Dialog Box*" on page 512 or "*Dialog Controls*" on page 515. A customized dialog and its **dialogbox** group can be found in "*A Sample Dialog*" on page 534.

The *Dialog Editor* is never more than a few clicks away. From within Procomm Plus or the *ASPECT Editor*, you can activate the *Dialog Editor* by selecting it from the ***Tools*** menu. Additionally, you can run it from the ***Start*** menu by selecting ***Start* | *Programs* | *Procomm Plus* | *Procomm Plus Utilities* | *Dialog Editor***. The editor's main window opens with an untitled dialog.

Elements of the Dialog Editor

The editor's main window is composed of three primary elements: a menu bar and *Action Bar*, a dialog "work area," and a status line. The menu bar and *Action Bar* contain the tools you need to modify your dialog. The dialog work area is where your dialog box is displayed. "*The Dialog Box*" on page 512 provides further details about the workspace and it use. "*The Status Line*", described on page 511, gives you information about the currently selected control, or the dialog box itself.

The Status Line

The status line appears at the bottom of the *Dialog Editor* window and is comprised of four different fields. If you are running the *Dialog Editor* in a small sized window, or if you are running at a lower resolution, you may not see all four fields.

The first field of the status line always displays the description of the currently selected item. This may be “***Main Dialog***,” if the dialog box itself is selected, or a particular control name such as “***Icon***” or “***Pushbutton***.” After a control has been placed, the field is blank when no item is selected.

The values in the second status line field reflect the location of the currently selected item. Again, this can be either the dialog box itself, or a control. The ***x*** and ***y*** values represent the horizontal and vertical coordinates of the upper left corner of the item in *Dialog Box Units* or DBUs.

If the selected item is a dialog control, its ***x*** and ***y*** values are measured with respect to the dialog box. If the dialog box itself is selected, its ***x*** and ***y*** values are measured with respect to the editor’s main window. The field is blank if no item is selected.

The third status line field describes the size of the currently selected item, with the ***dx*** and ***dy*** values referring to the horizontal and vertical dimensions of the item in DBUs. The field is blank if no item is selected.

The final status line field displays the ID value of the currently selected item. The field is blank if no item is selected.

“Dialog Box Units” on page 48 provides detailed information regarding DBUs. To learn how DBUs are used in ASPECT, see the ***dialogbox*** *command.*

Importing and Exporting Dialogs

If you are using the *Dialog Editor*, you’re probably creating a dialog for a script or modifying an existing dialog. If you’re creating a new dialog, you’ll be concerned with exporting your dialog from the *Dialog Editor*. If you just want to modify an existing dialog’s code, you’ll need to import it, and then export it again.

Importing an ASPECT Dialog

There are two common methods for importing an existing **dialogbox** group from a source file:

- By pasting it directly from the source file.

Within the *ASPECT Editor*, you can copy the dialog from the editor to the Windows Clipboard, then paste it directly into the *Dialog Editor.* Simply highlight the entire **dialogbox** group in its source file, copy it, switch to the *Dialog Editor*, and paste it.

- By opening its *Windows User Dialog* or "**.wud**" file.

  Many script writers embed dialogs in scripts using **.wud** files and the **#include** command. If you use this method, you can open the **.wud** file by selecting the editor's ***File | Open...*** menu, and browsing for the file. The editor loads and displays the dialog.

Exporting an ASPECT Dialog

Once you've completed your dialog, there are two common methods of exporting the **dialogbox** group for use in a script:

- By pasting it directly into the source file.

  Within the *Dialog Editor*, you can copy the entire **dialogbox** group ASPECT commands to the Windows Clipboard, then paste it directly into the *ASPECT Editor* (or other text editor). Simply select ***Edit | Copy Dialog*** from the menu, switch to your script editor, and paste it.

- By saving it as a *Windows User Dialog* or "**.wud**" file.

  You can embed **.wud** files into script files using the **#include** command. This allows you to re-use the dialogs in multiple scripts. Simply select the ***File | Save As...*** from the editor's menu, and save the dialog using an appropriate name.

  While the **.wud** extension is not required, it provides a consistent naming convention.

Regardless of the method you choose for including the **dialogbox** statement group in your script, you'll need to write supporting ASPECT commands to initialize the dialog's variables and to respond to the user's actions.

If you don't already have a dialog to modify, or are starting from scratch, "*The Dialog Box*" on page 512 offers more introductory discussion.

The Dialog Box

Dialogs provide the primary means of providing information to users. Similarly, they also provide an excellent method for gathering information from users. At start up, the *Dialog Editor*. It presents you with a default dialog box. Most likely you'll want to perform some modifications like resizing it, moving it, and changing its properties. Once you've completed all of your changes, you'll be ready to start placing dialog controls, as described on page 515.

Re-sizing the Dialog Box

To re-size the dialog, move the mouse pointer over one of the “drag handles” on its corners or sides. The mouse pointer changes to indicate the directions you can drag each particular handle. If the dialog doesn’t have drag handles, it’s not selected. Click on it once below its title bar to select it.

When you’ve located the handle you want to use, drag it in the desired direction. As you drag the handle, the editor leaves the dialog in its current size, but draws an outline to show you the dialog’s new size. When you’re satisfied with the dialog’s new size, release the mouse button, and the editor will update its display.

For more precise sizing, you can also use the cursor keys. Simply hold the <Shift> key down while you press the cursor keys. The up and down cursor keys move the bottom of the dialog, while the left and right cursor keys move the dialog’s right side one DBU at a time.

If the dialog is selected, the editor’s status line will report the dialog’s size in its third field. The ***dx*** and ***dy*** values represent the its width and height, respectively. These values will update as the dialog is being expanded or contracted.

Moving the Dialog Box

Although the editor places a new dialog within its work space under the *Action Bar*, ASPECT dialogs can normally be placed anywhere on your Windows display.

To move the dialog, select it by clicking on it once below its title bar, then place the mouse pointer inside the dialog’s title bar. Drag the dialog to its new location. The mouse pointer changes to indicate that you are moving the control. If the dialog doesn’t have a title bar, you can move it by dragging any of its borders to the new location.

The editor draws an outline around the mouse pointer to indicate the dialog’s new position. The dialog’s original position is unaffected until you release the mouse button.

You can also use the cursor keys for more precise positioning. Select the dialog, then use the cursor keys to move the control up, down, left, or right, one DBU at a time.

*The second field of the editor’s status line will always report the dialog’s position with respect to the editor’s window. The **x** and **y** values represent the number of DBUs from the **left** and **top** of the editor’s window to the **left top** corner of the dialog, respectively. These values are updated continuously.*

Dialog Box Properties

A dialog's display, and the way it interacts with your script and user input, are determined by the properties you assign to it. Double-click on the dialog box work area or right-click and select ***Properties*** to display the ***General*** settings panel of the **Main Dialog Options** dialog.

Property	Description
Moveable with Caption	If this check box is enabled, its associated edit field will accept either a string constant or variable to be displayed as the caption in the dialog box's title bar. Click on the ***Constant*** or ***Variable*** option buttons to select the edit field's content type.
Dialog ID	This edit field specifies the ID for the dialog. This integer value is used in such commands as **dlgupdate** and **dlgdestroy** to reference and control the dialog box. The editor will assign a default ***Dialog ID***. However, you'll probably want to modify it depending on your script's needs.
Call Procedure Name	If this check box is enabled, its associated edit field accepts the name of the ASPECT procedure to be called when the dialog box is created.
Parent ID	If this check box is enabled, its associated edit field accepts either a numeric constant, or a variable equal to the control ID of the parent or owner dialog of this dialog box. When a parent dialog box is destroyed, its child dialog boxes will automatically be destroyed as well.

Settings in the ***Styles*** panel of the **Main Dialog Options** dialog allow you to specify how the dialog will interact with your script and user input. Click on the ***Styles*** tab to access the following options:

Property	Description
Center Dialog Box	If this check box is enabled, the dialog box will be centered horizontally on the display screen.
Modeless	If this check box is enabled, the dialog box will be modeless. This means that the user will be able to access other features of Procomm Plus, or even other script dialogs, without closing the current dialog.
Hide Dialog Initially	If this check box is enabled, the dialog box will remain hidden until it's displayed by the **dlgshow** command. This is useful if the script needs to update variables within the dialog before showing them to the user.

Property	**Description**
Update Variables on Event	If this check box is enabled, the variables and files associated with the dialog box controls will be updated only as their respective events are read from the dialog event queue with **dlgevent**.
Disable Esc Key and Menu Close	If this check box is enabled, neither the dialog's control menu ***Close*** item nor the <Esc> key will close the dialog box.
Remove Close from System Menu	If this check box is enabled, the dialog's control menu ***Close*** item will be removed from the dialog box's system menu.
Dead End Dialog	If this check box is enabled, script execution will not continue past the **enddialog** command until the dialog box is destroyed.

When you're satisfied with the settings you've selected for your dialog, click on ***OK*** to save them and return to the main editing window. If you want to close the **Main Dialog Options** dialog without saving any changes you may have made, click on ***Cancel***. For more information on the various dialog options, see the **dialogbox** command. At this point, you're ready to start adding dialog controls.

Dialog Controls

Each control you place in a dialog results in an ASPECT command in the **dialogbox** statement group generated by the editor. Options you select for each control select ASPECT keywords that are also included in the generated ASPECT commands. The editor also includes settings and arguments reflecting the control's size, position, and properties within the dialogs you design.

If you'd like further details on how the finished **.wud** statement will appear for a particular control, you may wish to refer to the description for the corresponding command. See "*The ASPECT Commands*.", starting on page 65.

Windows 95 is a graphical interface which takes full advantage of a system's mouse. Great lengths are taken, however, to make sure that people who don't have a mouse can still access controls. This is done through the use of *accelerators*. When you are creating a dialog, you must make sure that you do not duplicate accelerator keys. "C," "O," and "X" are some of the easiest to be accidentally duplicated. This is because nearly every dialog has a ***Cancel***, ***OK***, or ***Exit*** button. If you duplicate an accelerator key, it will not cause compilation or run-time errors, but the duplicated accelerators will not function properly.

Manipulating Dialog Controls

Dialogs are made up of controls that display information or can accept information from a user. There are seventeen different controls which you can put into your dialogs. You can learn more details about each control by reading its corresponding ASPECT command reference or properties descriptions as displayed in Table *1, "ASPECT Dialog Controls."*

When you're designing a dialog, the first step you must take is deciding what type of controls you want to use. The next step is to place a control, as described in "*Placing a Control in the Dialog Editor*" on page 553. Repeat this step, as necessary, until you have all of the controls you'll need.

Once you have placed the controls, you can modify their properties in the *Dialog Editor*, or by directly modifying its ASPECT command in the **.was** or **.wud** file.

Table 1: ASPECT Dialog Controls

ASPECT Command	Dialog Editor Control
text	See "*Text Control Options*" on page 519
ftext	See "*FText Control Options*" on page 520
checkbox	See "*Checkbox Control Options*" on page 521
pushbutton	See "*Push Button Control Options*" on page 521
radiobutton	See "*Radio Button Control Options*" on page 522
iconbutton	See "*Icon Button Control Options*" on page 523
icon	See "*Icon Control Options*" on page 524
listbox	See "*List Box Control Options*" on page 525
flistbox	See "*FListbox Control Options*" on page 526
dirlistbox	See "*Dirpath Control Options*" on page 528
combobox	See "*Combo Box Control Options*" on page 529
fcombobox	See "*FCombobox Control Options*" on page 530
editbox	See "*Edit Box Control Options*" on page 531
feditbox	See "*Feditbox Control Options*" on page 532
groupbox	See "*GroupBox Control Options*" on page 533
metafile	See "*Metafile Control Options*" on page 533
bitmap	See "*Bitmap Control Options*" on page 534

Many of the buttons will be used at their default size, but many of the text fields and edit boxes require that you spend a moment resizing them. Once you have it properly sized, making the dialog look right often requires you to move a control. If you're duplicating a control, you'll find it easier than ever in this editor.

Re-sizing a Control

New dialog controls are initially drawn by the editor in a default size determined by the control's type. To re-size a control, move the mouse pointer over one of the "drag handles" on its corners or sides. The mouse pointer changes to indicate the directions you can drag each particular handle.

If the mouse pointer does not change when placed over a handle, it means that the control is not re-sizable. In particular, **icons** and **iconbuttons** are not re-sizable.

If the control you want to re-size doesn't have drag handles, click on it once with the mouse to "select" it. Once selected, the control can be moved or re-sized.

When you've located the handle you want to use, drag it in the desired direction. As you drag the handle, the editor leaves the control in its current state, but draws an outline to show you the control's new size. When you're satisfied with the control's size, release the mouse button, and the editor will update its display.

For more precise sizing, you can also use the cursor keys. Select the control you want to re-size, and hold the <Shift> key down while you press the cursor keys. The up and down cursor keys move the bottom of the control, while the left and right cursor keys move the control's right side one DBU at a time.

*The editor's status line always reports the size of the selected control in its third field; the **dx** and **dy** values represent the control's width and height, respectively. These values will update as the control is being expanded or contracted.*

If you have selected a group of controls, you can use the ***Layout*** menu items to help you arrange them. The ***Layout*** menu's capabilities are especially useful when you first place a group of controls. You can quickly position, size, and arrange them with only a few clicks of the mouse.

Moving a Control

If the control you want to move doesn't have drag handles, click on it once to select it. If you would like to select a group of controls, or a control of undetermined size, you may click outside of the control(s), drag diagonally to create a border encompassing the control(s), and release the mouse. This selects the group of controls contained within your border. To move the currently selected control, place the mouse pointer inside the area bounded by its drag handles, and drag it to its new location. The mouse pointer changes to indicate that you are moving the control.

The editor draws an outline around the mouse pointer to indicate the control's new position. The control's original position is unaffected until you release the mouse button.

You can also use the cursor keys for more precise positioning. Select the control you want to move, then use the cursor keys to move the control up, down, left, or right, one dialog unit at a time.

The second field of the editor's status line will always report the selected control's position with respect to the dialog. The ***x*** *and* ***y*** *values represent the number of DBUs from the* ***left*** *and* ***top*** *of the dialog's client area to the* ***left top*** *corner of the control, respectively. These values are updated continuously as you move the control.*

Duplicating Controls

You can duplicate a control by holding the <Ctrl> key down, then dragging the selected control. The new control will have the same size and characteristics as the original, except for its control ID.   You can re-size it, and double-click on it to modify its settings.

If you have a group of controls selected, this still holds true: you can quickly copy a group of controls. Simply select the group by "dragging" a border around them. Then, press <Ctrl>, click, and drag the duplicate group away.

Deleting a Control

To delete the currently selected control, press the <Del> or <Delete> key, or click on the ***Delete*** *Action Bar* button. The *Dialog Editor* will remove the control without prompting you to confirm the action.

Control Properties

Within each control's **Options** dialog are the various settings specific to that control. Many controls include similar settings, and the ***ID*** setting is necessary for every control. To view or modify a control's properties, double-click on it, or right click on it and select ***Properties***. This opens the ***General*** settings panel of the controls **Options** dialog.

Each of the controls includes a full description of each of its settings:

- "*Text Control Options*"
- "*Checkbox Control Options*"
- "*Radio Button Control Options*"
- "*Icon Control Options*"
- "*FListbox Control Options*"
- "*Combo Box Control Options*"
- "*Edit Box Control Options*"
- "*GroupBox Control Options*"
- "*Bitmap Control Options*"
- "*FText Control Options*"
- "*Push Button Control Options*"
- "*Icon Button Control Options*"
- "*List Box Control Options*"
- "*Dirpath Control Options*"
- "*FCombobox Control Options*"
- "*Feditbox Control Options*"
- "*Metafile Control Options*"

Text Control Options

Property	**Description**
Label	This field accepts either a string constant or variable to be displayed in the text control. If a variable is specified, a valid ASPECT name must be entered. In the ***Style*** panel, you may also specify ***Left, Center,***or ***Right Alignment*** for the ***text*** in this dialog. To include a keyboard accelerator in the text, place an ampersand (&) in front of the desired character. To display an ampersand in the text, use two ampersands. For example, the label "***&R&&D***" would display as "***R&D***," with the ***R*** being an underlined accelerator.
ID	This field requires a constant in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the control, and can be used with **dlgupdate** to refresh the **text** display. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **text**. However, you may want to change this ID in the actual script to suit your preferences.

When you're satisfied with the settings you've selected for your **text** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Text Options** dialog without saving any changes you may have made, click on ***Cancel***.

FText Control Options

Property	**Description**
File Name	In this field, you can specify either a string constant or variable for the name of the file to be displayed. If a variable is used, you must enter a valid ASPECT name. Otherwise, enter the *filespec* directly into this field.
File Offset	If this check box is enabled, the associated edit field accepts either a variable name or a numeric constant, specifying the starting point within the source file for the text to be displayed. Enabling this check box allows you to enable ***File Length***, and disables ***Dynamic File Update***.
File Length	If this check box is enabled, the associated edit field accepts either a variable name or a numeric constant, specifying the maximum number of bytes to be displayed from the source file. This check box cannot be enabled unless ***File Offset*** is enabled.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **ftext** box, and can be used with **dlgupdate** to refresh the **ftext** display. Each control within a **dialogbox** must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **ftext** box. However, you may want to change this ID in the actual script to suit your preferences.
Dynamic File Update	If this check box is enabled, the dialog can update the file's display with **dlgupdate**. Additionally, it can be reset by changing the filespec to another file or to null and issuing the **dlgupdate** command. Enabling this check box disables ***File Offset***.
Horizontal Scroll	If this check box is enabled, the **ftext** control will be displayed with the appropriate scroll bar, allowing the user to scroll the file display horizontally.

When you're satisfied with the settings you've selected for your **ftext** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **File Text Options** dialog without saving any changes you may have made, click on ***Cancel***.

Checkbox Control Options

Property	**Description**
Label	This field accepts either a string constant or variable to be displayed as the **checkbox** label. To provide a keyboard accelerator for the **checkbox**, simply place an ampersand (&) in front of the desired character within the label text. To display an ampersand in the text, use two ampersands; for example, the label "***&R&&D***" would display as "***R&D***," with the ***R*** appearing as an underlined accelerator.
State Variable	This field requires an integer variable name. The ***State Variable*** can be pre-initialized within the script for a starting value. It will also receive a value based on the status of the **checkbox**. A value of 1 indicates that the box is checked; a 0 indicates that it is unchecked.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The *ID* is used within a **dialogbox** statement group to reference the **checkbox**, and is reported to the script as a **dlgevent** whenever the check box is selected. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **checkbox**. However, you may want to change this ID in the actual script to suit your preferences.

When you're satisfied with the settings you've selected for your **checkbox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Check Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

Push Button Control Options

Property	**Description**
Label	This field accepts either a string constant or variable to be displayed as the **pushbutton** label. To provide a keyboard accelerator for the **pushbutton**, simply place an ampersand (&) in front of the desired character within the label text. To display an ampersand in the text, use two ampersands. For example, the label "***&R&&D***" would display as "***R&D***," with the ***R*** appearing as an underlined accelerator.

Property	Description
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The *ID* is used within a **dialogbox** statement group to reference the **pushbutton**, and is reported to the script as a **dlgevent** whenever the button is selected. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **pushbutton**. However, you may want to change this ID in the actual script to suit your preferences.
Style Selection	There are three styles of **pushbuttons**: ***Normal***, ***OK***, and ***Cancel***. A ***Normal*** button triggers a dialog event, but does not close the dialog. A ***Cancel*** button closes the dialog and signals a cancel event. An ***OK*** button updates an associated variable and closes the dialog.
Default	If enabled, the **pushbutton** will act as the default choice if the user simply presses <Enter>, or double-clicks within certain list controls. A dialog box can have only one default **pushbutton** or **iconbutton**.

When you're satisfied with the settings you've selected for your **pushbutton** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Push Button Options** dialog without saving any changes you may have made, click on ***Cancel***.

Radio Button Control Options

Property	Description
Button Label	This edit field accepts either a string constant or variable to be displayed as the **radiobutton** label. To provide a keyboard accelerator for the **radiobutton**, simply place an ampersand (&) in front of the desired character within the label text. To display an ampersand in the text, use two ampersands. For example, the label "***&R&&D***" would display as "***<u>R</u>&D***," with the ***R*** appearing as an underlined accelerator.
Button ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **radiobutton**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **radiobutton**. However, you may want to change this ID in the actual script to suit your preferences. When a **radiobutton** is selected, its ***Button ID*** value is assigned to the status variable associated with its **radiogroup**.

Property	**Description**
Radio Button Style	**radiobutton**s must be associated with a **radiogroup** command. The **option button group**'s associated variable is assigned the value of the control ID of the **option button** selected by the user. Only one **option button** within a group can be selected at any time. Click on the ***Group ID*** list box to select an **option button group**. If an **option button group** does not exist, it can be created with the ***New Group*** button. Existing **option button group**s can be edited by pressing the ***Edit Group*** button.

When you're satisfied with the settings you've selected for your **radiobutton** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Radiobutton Options** dialog without saving any changes you may have made, click on ***Cancel***.

Icon Button Control Options

Property	**Description**
Label	This field accepts either a string constant or variable to be displayed as the **iconbutton** label. To provide a keyboard accelerator for the **iconbutton**, simply place an ampersand (&) in front of the desired character within the label text. To display an ampersand in the text, use two ampersands. For example, the label "***&R&&D***" would display as "***R&D***," with the ***R*** appearing as an underlined accelerator.
File Name	This field accepts either a string constant or variable specifying the file containing the icon graphic. If you select a string constant, a ***Browse*** button is enabled, allowing you to select an icon file using a standard file open dialog. If you specify a variable, an edit field is displayed, allowing you to select a variable name. You can access icon graphics in executable programs (**.exe**), DLL modules (**.dll**), and icon libraries (**.ico**, **.icl**, and **.nil** files).
Index	This field accepts either a numeric constant or variable, specifying the index of the icon graphic within the target file. Since some icon files can contain multiple icons, this number identifies which icon to use. If you specified a constant for the ***File Name*** and ***Index*** fields, the ***Prev*** and ***Next*** buttons allow you select an icon from a file containing multiple icons. The currently selected icon is displayed in the preview button below the ***ID*** edit field.

Property	Description
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **iconbutton**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **iconbutton**. However, you may want to change this ID in the actual script to suit your preferences.
Iconbutton Style	An **iconbutton**'s style determines how it interacts with your script. There are three styles of **iconbuttons**: ***Normal***, ***OK***, and ***Cancel***. A ***Normal*** button triggers a dialog event, but does not close the dialog. A ***Cancel*** button closes the dialog and triggers a "cancel" event. An ***OK*** button triggers an OK event and closes the dialog.
Default	If this check box is checked, the **iconbutton** will act as the default choice if the user simply presses <Enter>, or double-clicks within certain list controls. A dialog box can have only one default **pushbutton** or **iconbutton**.

When you're satisfied with the settings you've selected for your **iconbutton** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Icon Button Options** dialog without saving any changes you may have made, click on ***Cancel***.

Icon Control Options

Property	Description
File Name	This field accepts either a string constant or variable, specifying the file containing the icon graphic. If you select a string constant, a ***Browse*** button is provided, allowing you to select an icon file using a standard file open dialog. If you select a variable, an edit field is provided, allowing you to specify a variable name. You can access icon graphics in executable programs (**.exe**), DLL modules (**.dll**), and icon libraries (**.ico**, **.icl**, and **.nil** files).
Index	This field accepts either a numeric constant or variable, specifying the index of the icon graphic within the target file. Since some icon files can contain multiple icons, this number identifies which icon to use. If you specified a constant for the ***File Name*** and ***Index*** fields, the ***Prev*** and ***Next*** buttons allow you to select an icon from a file containing multiple icons. The currently selected icon is displayed in the preview button in the lower right corner of the dialog.

Property	**Description**
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **icon**. It can also be used to change the **icon** if variables were specified for either the ***File Name*** or ***Index*** fields. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **icon**. However, you may want to change this ID in the actual script to suit your preferences.

When you're satisfied with the settings you've selected for your **icon** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Icon Options** dialog without saving any changes you may have made, click on ***Cancel***.

List Box Control Options

Property	**Description**
Itemlist	This field accepts either a string constant or variable for the comma-separated items to be displayed in the **listbox**. A vertical scrollbar will be displayed at run-time if the item list contents are too numerous to be displayed simultaneously.
Tab String	If this check box is enabled, its associated edit field accepts either a string constant or a variable. The constant or variable should contain a comma-separated string of tab positions. These positions will be used to space the record fields in the display. This option is useful if the items in the list contain tab-delimited records. Tab positions are measured in DBUs.
Select Variable	This field only accepts a variable name. The named variable will be updated with the contents of the item selected by the user.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the list box. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **listbox**. However, you may want to change this ID in the actual script to suit your preferences.

Property	Description
List Box Style	A **listbox**'s style determines how it interacts with your script. To allow a user only a single selection from the list, select the ***Single*** option button. The ***Multiple*** option button allows one or more items to be selected at once. If the user selects more than one item in the list, the items will be comma-separated in the ***Select Variable*** string.
Sort	If this check box is enabled, the item list will be sorted lexicographically in ascending order when displayed.
Horizontal Scroll	If this check box is enabled, the **listbox** control will be displayed with the appropriate scroll bars, allowing the user to scroll the item display.

When you're satisfied with the settings you've selected for your **listbox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **List Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

FListbox Control Options

Property	Description
Input File	This field accepts either a string constant or variable for the name of the item list file to be displayed in the **flistbox**. A vertical scrollbar will be displayed at run-time if the item list file contents are too large to be displayed in the list box.
File Offset	If this check box is enabled, its associated edit field accepts either a numeric constant or variable specifying the offset into the item list file where the displayed list begins.
File Length	If this check box is enabled, its associated edit field accepts either a numeric constant or variable specifying the maximum number of bytes to display from the offset.
Tab String	If this check box is enabled, its associated edit field accepts either a string constant or a variable. The constant or variable should contain a comma-separated string of tab positions. These positions will be used to space the record fields in the display. This option is useful if the items in the list file contain tab-delimited records. Tab positions are measured in DBUs.
Select Variable	This field only accepts a variable name. If a ***Single*** selection is specified, the named variable will be updated with the contents of the item selected by the user. If ***Multiple*** selections are allowed, this variable specifies the file to be created to contain the selections.

Property	Description
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **flistbox**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **flistbox**. However, you may want to change this ID in the actual script to suit your preferences.
FListbox Style	An **flistbox**'s style determines how it interacts with your script. To allow a user only a single selection from the list, select the ***Single*** option button. The ***Multiple*** option button allows one or more items to be selected at once. If the user selects more than one item from the **flistbox**, they will be written to the text file specified by the ***Select Variable***, one item per line.
Sort	If this check box is enabled, the item list will be sorted lexicographically in ascending order when displayed.
Horizontal Scroll	If this check box is enabled, the **flistbox** control will be displayed with the appropriate scroll bars, allowing the user to scroll the item display.

When you're satisfied with the settings you've selected for your **flistbox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **File List Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

Dirlistbox Control Options

Property	Description
Input File	This field accepts either a string constant or variable for the *filespec* used to scan the files in the target directory. This *filespec* may include DOS wildcards and/or a DOS path name. A vertical scrollbar will be displayed at run-time if the resultant directory listing is too long for the list box.
File Type	If this check box is enabled, its associated field accepts either a string constant or a variable. It can be used to limit the search pattern for the directory list. For more information, see **dirlistbox**.
Select Variable	This field only accepts a variable name. If a single selection is specified, the named variable will be updated with the contents of the item selected by the user. If multiple selections are allowed, this variable specifies the file to be created to contain the selections.

Property	Description
Dir Path ID	If this check box is enabled, its associated edit field accepts either a numeric constant or variable. This value represents the control ID for a **dirpath** control within the same dialog, which is updated with the current directory path selected by the user within the **dirlistbox** control.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **dirlistbox**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **dirlistbox**. However, you may want to change this ID in the actual script to suit your preferences.
Directory Listbox Style	A **dirlistbox**'s style determines how it interacts with your script. To allow a user only a single selection from the list, select the ***Single*** option button. The ***Multiple*** option button allows one or more items to be selected at once. If the user selects more than one item from the **dirlistbox**, they will be written to the text file specified by the ***Select Variable***, one item per line.
Sort	If this check box is enabled, the item list will be sorted lexicographically in ascending order when displayed.
Horizontal Scroll	If this check box is enabled, the **dirlistbox** control will be displayed with the appropriate scroll bars, allowing the user to scroll the item display.

When you're satisfied with the settings you've selected for your **dirlistbox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Directory List Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

Dirpath Control Options

Property	Description
Variable	If this check box is enabled, its associated edit field accepts a string variable name. This variable will be updated with the current directory path selected within a **dirpath** control and displayed.

Property	**Description**
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **dirpath**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **dirpath**. However, you may want to change this ID in the actual script to suit your preferences.

When you're satisfied with the settings you've selected for your **dirpath** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Directory Path Options** dialog without saving any changes you may have made, click on ***Cancel***.

Combo Box Control Options

Property	**Description**
Itemlist	This field accepts either a string constant or variable for the comma-separated items to be displayed in the **combobox**. A vertical scrollbar will be displayed at run-time if the item list contents are too numerous to be displayed simultaneously.
Select Variable	This field only accepts a variable name. The named variable will be updated with the contents of the item selected by the user.
Edit Length	If this check box is enabled, its associated edit field accepts either a numeric constant or variable, specifying the maximum number of characters that can be entered in a ***Simple*** or ***Drop Down*** **combobox**'s edit field.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **combobox**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **combobox**. However, you may want to change this ID in the actual script to suit your preferences.

Property	**Description**
Combo Box Style	A **combobox**‘s style determines how it interacts with your script. There are three styles of **combobox**es supported by ASPECT. A ***Simple*** **combobox** will always display the item list, with the current selection displayed in its edit field. The ***Drop Down*** **combobox** is similar to the ***Simple*** type, but the list is not displayed unless the button beside the edit field is selected. The ***Drop Down List*** **combobox** is similar to the ***Drop Down*** style, but the edit field is replaced by static text, meaning that no new items can be added to the list from the keyboard.
Sort	If this check box is enabled, the **combobox** item list will be sorted lexicographically in ascending order when displayed.

When you’re satisfied with the settings you’ve selected for your **combobox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Combo Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

FCombobox Control Options

Property	**Description**
Input File	This field accepts either a string constant or variable for the name of the item list file to be displayed in the **fcombobox**. A vertical scrollbar will be displayed at run-time if the item list file contents are too numerous to be displayed simultaneously.
File Offset	If this check box is enabled, its associated edit field accepts either a numeric constant or variable specifying the offset into the item list file where the displayed list begins.
File Length	If this check box is enabled, its associated edit field accepts either a numeric constant or variable specifying the maximum number of bytes to display from the offset.
Edit Length	If this check box is enabled, its associated edit field accepts either a numeric constant or variable, specifying the maximum number of characters that can be entered in a ***Simple*** or ***Drop Down*** **fcombobox**’s edit field.
Select Variable	This field only accepts a variable name. The named variable will be updated with the contents of the item selected by the user.

Property	**Description**
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **fcombobox**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **fcombobox**. However, you may want to change this ID in the actual script to suit your preferences.
Fcombobox Style	An **fcombobox**'s style determines how it interacts with your script. There are three styles of **fcombobox**es supported by ASPECT. A ***Simple*** **fcombobox** will always display the item list, with the current selection displayed in its edit field. The ***Drop Down*** **fcombobox** is similar to the ***Simple*** type, but the list is not displayed unless the button beside the edit field is selected. The ***Drop Down List*** **fcombobox** is similar to the ***Drop Down*** style, but the edit field is replaced by static text, meaning that no new items can be added to the list from the keyboard.
Sort	If this check box is enabled, the **fcombobox** item list will be sorted lexicographically in ascending order when displayed.

When you're satisfied with the settings you've selected for your **fcombobox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **File Combo Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

Edit Box Control Options

Property	**Description**
Variable	This field accepts a string variable name which will be updated with any changes made by the user.
Length	If this check box is enabled, its associated edit field accepts either a numeric constant or variable, specifying the maximum number of characters that can be entered in the control. If ***Length*** is not specified, up to 255 characters can be entered in the edit field. If necessary, the field will scroll to accommodate larger strings.

Property	Description
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **editbox**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **editbox**. However, you may want to change this ID in the actual script to suit your preferences.
Mask Input	If this check box is enabled, text in the **editbox** control will be displayed as asterisks. This is useful if the field is intended to contain sensitive information, such as a password or account number.
Multiline	If this check box is enabled, text displayed within the **editbox** will line-wrap, rather than scroll as characters are entered at the keyboard.

When you're satisfied with the settings you've selected for your **editbox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Edit Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

Feditbox Control Options

Property	Description
File Name	This field accepts either a string constant or variable for the name of the text file to be edited or created in the **feditbox**. The target file can contain a maximum of 65536 characters. A vertical scrollbar will be displayed at run-time if the target file's contents are too large to be displayed simultaneously.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **feditbox**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **feditbox**. However, you may want to change this ID in the actual script to suit your preferences.
Horizontal Scroll	If this check box is enabled, the **feditbox** control will be displayed with the appropriate scroll bars, allowing the user to scroll the item display.

When you're satisfied with the settings you've selected for your **feditbox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **File Edit Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

GroupBox Control Options

Property	**Description**
Label	If this check box is enabled, its associated edit field accepts either a string constant or variable to be displayed as the label for the **groupbox**. To provide a keyboard accelerator for the **groupbox**, simply place an ampersand (&) in front of the desired character within the label text. To display an ampersand in the text, use two ampersands. For example, the label "***&R&&D***" would display as "***R&D***," with the ***R*** appearing as an underlined accelerator. This sets the focus on the first control in the group.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **groupbox**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **groupbox**. However, you may want to change this ID in the actual script to suit your preferences.

When you're satisfied with the settings you've selected for your **groupbox** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Group Box Options** dialog without saving any changes you may have made, click on ***Cancel***.

Metafile Control Options

Property	**Description**
File Name	This field accepts either a string constant or variable name, containing the name of the metafile file to be displayed in the dialog box. If you select a string constant, a ***Browse*** button is provided, allowing you to select a metafile using a standard file open dialog.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **metafile**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **metafile**. However, you may want to change this ID in the actual script to suit your preferences.

When you’re satisfied with the settings you’ve selected for your **metafile** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Metafile Options** dialog without saving any changes you may have made, click on ***Cancel***.

Bitmap Control Options

Property	**Description**
File Name	This field accepts either a string constant or variable name, containing the name of the bitmap file to be displayed in the dialog box. If you select a string constant, a ***Browse*** button is provided, allowing you to select a bitmap using a standard file open dialog.
ID	This field requires a constant or a symbol defined as a constant, in the range of 1 to 999. The ID is used within a **dialogbox** statement group to reference the **bitmap**. Each control within a dialog box must have a control ID, and each ID must be unique. The *Dialog Editor* automatically supplies a control ID for the **bitmap**. However, you may want to change this ID in the actual script to suit your preferences.

When you’re satisfied with the settings you’ve selected for your **bitmap** control, click on ***OK*** to save them and return to the main editing window. If you want to close the **Bitmap Options** dialog without saving any changes you may have made, click on ***Cancel***.

A Sample Dialog

The dialog described below was constructed using the *Dialog Editor* and saved as a **.wud** file.The dialog contains a variety of controls including a variable text caption, a variable text field enclosed by a **groupbox** control, and a group of labeled **editbox**es. **pushbutton**s for closing the dialog and opening help are also used.

The **dialogbox** statement group generated by the *Dialog Editor* looks like this:

```
;Sample Dialog Box
dialogbox 0 0 0 245 146 26 DialogTitle parent 0
  bitmap 1 194 6 41 37  DialogGraphic
  groupbox 2 5 6 187 26
  text 34 8 16 180 11 DialogBodyText center
  groupbox 4 2 36 186 88
  text 54 8 49 38 8 "&Libraries:" right
  editbox 65 48 47 93 11 strvar3
  pushbutton 78 144 46 40 14 "&Select"
  text 84 8 68 38 8 "&Filename:" right
  editbox 95 48 65 49 12 strvar4
  text 10 109 68 38 8 "&Days old:" right
  editbox 11 150 65 32 11 strvar6
  text 12 8 85 38 8 "&Keywords: " right
  editbox 13 48 82 134 11 strvar2
  text 14 8 103 64 8 "&Uploaded by (ID):" right
  editbox 15 76 100 106 11 strvar5
  pushbutton 16 194 46 40 14 "&OK"
  pushbutton 17 194 64 40 14 "&Help"
  pushbutton 18 194 82 40 14 "&Cancel" CANCEL
enddialog
```

The User Window Editor

The *User Window Editor* helps you build a graphical interface that is displayed in the Procomm Plus *Terminal* or *Telnet* window. The editor allows you to create and edit your own *User window*s using a visual approach that is easier to use than manually creating a **uwincreate** statement group and provides real-time display of your changes.

With familiar Windows elements like **pushbutton**s, **icon**s, **bitmap**s, and **metafile**s, as well as special ASPECT objects like **dllobject**s and **hotspot**s, *User window*s can give your script applications a strong graphical presentation. For example, with a customized *User window*, you could create a background map of the United States displaying your sales regions and key cities. **hotspot**s could then be placed on map locations which would activate scripts to dial up host systems at regional sales headquarters.

As you design your *User window* and its objects, the *User Window Editor* maintains the ASPECT commands that will recreate it within a script. The editor produces a standard **uwincreate** statement group that you can easily modify or port between scripts. For more information, see “*Importing and Exporting User Windows*” on page 537.

The *User Window Editor* is never more than a few clicks away. From within Procomm Plus or the *ASPECT Editor*, you can select it from the ***Tools*** menu. Alternatively, you can run the *User Window Editor* from the ***Start*** menu by selecting ***Start | Programs | Procomm Plus | Procomm***

Plus Utilities* | *User Window Editor. The editor’s main window opens with an untitled *User window*.

Elements of the User Window Editor

The editor’s main window is composed of three main elements: a menu bar and *Action Bar*, a *User window* “work area,” and an *Object Description* status line. It can also display the *Terminal window*’s *Meta Keys* and *Quick Select line*.

The *User Window* work area provides the display backdrop while you’re editing a user window. Each object’s description is provided in the Status Line.

While the *Quick Select line* and *Metakeys* appear similar to the *Terminal window*’s, they are only shown to illustrate how a given *User window* will appear with these features enabled in actual use. They have no other function within the *User Window Editor*.

The Status Line

The *Object Description* status line appears at the bottom of the editor window.

The *User Window Editor* displays information about the currently selected object on the status line. If no object is selected, the status line fields will be blank.

The first field of the status line always displays the type of the currently selected object. For example, if a **pushbutton** is selected, “***Pushbutton***” appears in this field.

The values in the second and third status line fields reflect the location and size of the currently selected object. The ***x*** and ***y*** values indicate the horizontal and vertical coordinates of the upper left corner of the item in screen pixels, while the ***dx*** and ***dy*** values refer to the horizontal and vertical dimensions of the object in screen pixels. These values are converted to *User Window Units*, or UWUs when the ASPECT code is created. You can select ***View* | *User Window* | *Coordinates*** and toggle the status line’s display between the default ***Pixels*** values and ***UWUs***.

The final status line field displays the ID value of the currently selected object.

For further information on how UWUs are used in ASPECT, and other aspects of User windows and their objects, see the ***uwincreate*** *command.*

Importing and Exporting User Windows

Since you're using the *User Window Editor*, we'll assume that you are creating a custom *User Window*, or modifying an existing one. If you're creating a completely new *User Window*, you'll only need to worry about exporting it, but if you're modifying an existing *User Window*, you'll need to import it first.

Importing a User Window

There are two common methods of importing an existing **uwincreate** statement group from a script:

- By pasting it directly from the source file.

  You can paste the ASPECT commands into the *User Window Editor*. Within the *ASPECT Editor*, simply highlight the **uwincreate** statement group, copy it to the Windows Clipboard, switch to the *User Window Editor*, and select ***Edit | Paste Window***.
- By opening its *User Window Source* or **.uws** file.

  Many script writers save complex *User Window* commands in a separate file using the **.uws** extension, and reference them with the **#include** command. If this is the case, simply select the editor's ***File | Open...*** menu item and select your file.

Whichever way you import the **uwincreate** group, the editor loads and recreates your window, and you can make your changes.

Exporting a User Window

Once you've finished creating or modifying your *User Window*, there are two common methods of exporting the **uwincreate** statement group:

- By saving it as a *User Window Source or* **.uws** file.

  You can save your *User window* to disk as a separate file, and embed it in the main script's source as an **#include** file. This allows you to re-use the *User Window* in multiple scripts.
- By pasting it directly into your **.was** source file.

  From within the *User Window Editor*, select ***Edit | Copy Window*** to copy the **uwincreate** statement group to the Windows Clipboard, switch to the *ASPECT Editor*, and paste it directly into your **.was** file.

Regardless of the method you use to insert the statement group into your script, you'll need to write supporting statements that process the *User window* variables and act upon the user's responses.

Editing a User Window

Before you begin placing objects in your *User window*, it's a good idea to specify the window's properties. This is especially important if you'll be using a background graphic in the window. For example, if you intend to place buttons over particular areas on a background graphic, you'll need the graphic in place before you add the buttons.

The *User window*'s properties are specified in dialogs accessed from the ***User Window*** menu:

User Window Placement and Size

By default, *User window*s fill the entire *Terminal window*. To specify a different size and position, select ***View | User Window | Placement...*** to open the **User Window Placement** dialog.

The ***Window Position*** setting determines the *User window*'s placement within the *Terminal window*. Select ***Full Screen*** to have the *User window* fill the entire window, or select one of the ***Relative x*** options to align the *User window* with one of the four *Terminal window* borders.

By default, ***Full Screen*** is specified for the ***Window Position***, and the ***Window Size*** field is disabled. If you're not creating a ***Full Screen*** *User window*, use the ***Window Size*** field to determine the size of the *User window.*

The dimensions you specify in the ***Size*** field vary in size depending on the type of ***Units*** you specify. By default, *User window*s are measured as a percentage of the *Terminal window.* However, you can also choose to measure windows as a percentage of the entire display screen, or as a specific number of pixels by selecting one of the ***Units*** options.

User Window Background

By default, *User window*s are assigned a white background. To specify a different background, select ***View | User Window | Background...*** to open the **User Window Background** dialog.

You'll probably want to set up your background before you place any objects —changing background options after you've added your objects can cause them to move unexpectedly if the background is re-sized.

The ***Color*** group allows you to specify RGB values for the background color. To specify a background color for the *User window,* simply enter the appropriate values in the ***Red***, ***Green***, and ***Blue*** edit fields. Values from 0 to 255 are accepted. For example, entering 128, 128, 128 provides a neutral gray background.

The ***Graphics*** group allows you to specify a type of background graphic for your *User window*. The *User Window* behaves differently depending on the type of background graphic you choose. You can choose from three different types of backgrounds:

- ***Blank***

  If you choose ***Blank***, you can specify how objects move when the *User window* is re-sized. Click on ***Options...*** to open the **Blank Background Options** dialog.

  Select ***Act like Bitmap Background*** to have objects remain in a fixed position in the *User window.* Select ***Act like Metafile Background*** if objects should automatically re-size and/or move relative to the *User window* size. Click on ***OK*** to close the dialog.

- ***Bitmap***

  If you specify a **bitmap** background, you can browse for a bitmap file by clicking on ***New...*** After selecting a bitmap file, you can specify its placement and other properties. Click on ***Options*** to open the **Bitmap Background Options** dialog.

  ❖ The options in the ***Horizontal Positioning*** group allow you to center the bitmap horizontally, or align it either to the left or right of the *User window.* You can also choose to tile the bitmap to cover the window background.

  ❖ The options in the ***Vertical Positioning*** group allow you to center the bitmap vertically, or align it either to the top or bottom of the *User window.* Objects cannot be re-sized automatically on top of a bitmap background.

  ❖ The options in the ***Graphic Load Options*** group allow you to specify how ASPECT loads the graphic into the *User window.* If you select ***Load into Memory***, the graphic is displayed and repainted faster, but it occupies more of your system memory. If you select ***Load from File***, the graphic requires no extra system memory to load, but displays and repaints will be somewhat slower.

- ***Metafile***

  If you specify a **metafile** background, you can browse for the metafile you want to use by clicking on ***New...*** After you have selected a metafile, you can click on ***Options...*** to open the **Metafile Background Options** dialog.

  ❖ The options in the ***Horizontal Positioning*** group allow you to center the metafile horizontally, or align it to the left or right of the *User window.*

  ❖ The options in the ***Vertical Positioning*** group allow you to center the metafile vertically, or align it to the top or bottom of the *User window.*

  ❖ The options in the ***Graphic Mapping*** group allow you to specify how the graphic will appear in the *User window.*

    If you select ***Fill Window***, the metafile will fill the *User window.* If necessary, the metafile will be stretched or contracted. The ***Horizontal*** and ***Vertical Positioning*** group boxes are grayed when ***Fill Window*** is selected.

    If you choose ***Maintain Aspect Ratio***, the metafile's original proportions will be retained, no matter how the *User window* is re-sized. With this method, you can assign a horizontal alignment for the background as well as align it vertically at the top, bottom or center of the window. If ***Maintain Aspect Ratio*** is selected, the display graphic may not fill the entire *User window* if it's re-sized.

In order for the User window and its objects to be moveable and re-sizeable, two conditions must be met: the objects must be set to ***Bkg Relative****, and the background itself must be a* ***Metafile*** *or, if* ***Blank****, set to* ***Act Like a Metafile Background*** *in* ***Blank Background Options****. However, if a* ***Bitmap*** *background is used, and the bitmap is not located in the upper left corner, the bitmap will move when the User window re-sizes, and any background-relative objects will move with it.*

User Window Objects

After the background of a *User Window* is defined, the next step is to place a user window object, as described on page 554. There are seven types of objects you can place in the *User Window*, as shown in **Table 2** below:

Table 2: ASPECT User Window Objects

ASPECT Command	User Window Object
pushbutton	See "*Pushbutton Object Options*" on page 542
iconbutton	See "*Iconbutton Object Options*" on page 543
icon	See "*Icon Object Options*" on page 545
metafile	See "*Metafile Object Options*" on page 546
bitmap	See "*Bitmap Object Options*" on page 547
dllobject	See "*DLLObject Options*" on page 548
hotspot	See "*Hotspot Object Options*" on page 549

Re-sizing an Object

New *User window* objects are initially drawn by the editor in a default size that is determined by the object's type. To re-size an object, move the mouse pointer over one of the "drag handles" on its corners or sides. The mouse pointer changes to indicate the directions you can drag each particular handle.

If the mouse pointer does not change when placed over a handle, it means that the object is not re-sizable. In particular, **icons** and **iconbuttons** are not re-sizable.

If the object you want to re-size doesn't have drag handles, click on it once with the mouse to "select" it. Once selected, the object can be moved or re-sized. When you've located the handle you want to use, drag it in the desired direction. As you drag the handle, the editor leaves the object in its current state, but draws an outline to show you the object's new size. When you're satisfied with the object's size, release the mouse button, and the editor will update its display.

You can also use the cursor keys for more precise sizing. Select the object you want to re-size, and hold the <Shift> key down while you press the cursor keys. The up and down keys move the bottom of the object, while the left and right keys move the object's right side one pixel or UWU at a time.

The editor's status line will always report the size of an object in its third field; the ***dx*** *and* ***dy*** *values represent the object's width and height, respectively. These values will update as the object is being expanded or contracted.*

Moving an Object

If the object you want to move doesn't already have drag handles, click on it once to select it. To move the selected object, place the mouse pointer inside the area bounded by its drag handles, and drag it to its new location. The mouse pointer changes to indicate that you are moving the object.

The editor draws an outline around the mouse pointer to indicate the object's new position. The control's original position is unaffected until you release the mouse button.

You can also use the cursor keys for more precise positioning. Select the object you want to move, then use the cursor keys to move the object up, down, left, or right, one pixel or UWU value at a time.

The second field of the editor's status line will always report the selected control's position with respect to the User Window Editor's client area. The ***x*** *and* ***y*** *values represent the number of pixels or UWUs from the* ***left*** *and* ***top*** *of the editor's client area to the* ***left top*** *corner of the object, respectively. These values are updated continuously as you move the object.*

Duplicating Objects

You can duplicate an existing **pushbutton**, **dllobject** or **hotspot** by holding the <Ctrl> key down, and then dragging the selected object. The new object will have the same size and characteristics as the original. You can re-size it and double-click on it to modify its settings.

Deleting an Object

If you find that you no longer need an object, you will want to delete it. The first step is to select the unwanted object. Once it is selected, press <Delete>, or click on the ***Delete*** *Action Bar* button. The *User Window Editor* will remove the object without prompting you to confirm the action.

After you've placed an object in the *User window*, you can modify its settings by double-clicking on it to open its associated **Options** dialog. The selections you make in an object's **Options** dialog are reflected in the object's ASPECT command in the **uwincreate** statement group generated by the editor. Each of the "*Object Properties*" is discussed.

Object Properties

Within each object's **Options** dialog are the various settings specific to that object. Many objects include similar settings, and the ***ID*** setting is necessary for every control. To view or modify a control's properties double-click on it to open the object's **Options** dialog.

All of the objects include full descriptions of their settings:

- "*Pushbutton Object Options*"
- "*Iconbutton Object Options*"
- "*Icon Object Options*"
- "*Metafile Object Options*"
- "*Bitmap Object Options*"
- "*DLLObject Options*"
- "*Hotspot Object Options*"

Pushbutton Object Options

Property	Description
Initial Values	The fields in the ***Initial Values*** group box must be constants, specifying the start-up values for the push button ***Label*** and its control ***ID***. If you do not specify variable names in the corresponding fields within the ***Variable Names*** group box, the ***Initial Values*** will remain constant for the life of the object.

Property	Description
Label	This edit field accepts either a string constant or variable to be displayed as the **push button** label. To provide a keyboard accelerator for the **push button**, simply place an ampersand (&) in front of the desired character within the label text. To display an ampersand in the text, use two ampersands. For example, the label "***&R&&D***" would display as "***R&D***," with the ***R*** appearing as an underlined accelerator.
ID	This field requires a constant, in the range of 1 to 2147483647. The ID is used within a **uwincreate** statement group to reference the **pushbutton.** Its value is returned in the $OBJECT system variable when the push button is selected. Each object within a *User window* must have a control ID, and each ID must be unique.
Variable Names	The fields in the ***Variable Names*** group box are optional. They allow you to specify ASPECT variable names for the push button's ***Label***, ***ID***, location, and size parameters. If you supply variable names in these fields, the variables will be initialized with the appropriate values from the ***Initial Values*** group box and the push button's size and location values in its **uwincreate** statement. Otherwise, the push button's size, position, label, and ID will remain constant throughout its life.
Position	If you select the ***Fixed*** option in the ***Position*** group box, the push button will remain the same size and in the same location on your screen, even if the window is re-sized. If you select the ***Bkg Relative*** option, the push button will always maintain the same position and size relative to the *User window* background.

When you're satisfied with the settings you've selected for your **pushbutton** object, click on ***OK*** to save them and return to the main editing window. If you want to close the **Pushbutton Object Options** dialog without saving any changes you may have made, click on ***Cancel***.

Iconbutton Object Options

Property	Description
Initial Values	The fields in the ***Initial Values*** group box must be constants, specifying the start-up values for the **iconbutton**. If you do not specify variable names in the corresponding fields within the ***Variable Names*** group box, the ***Initial Values*** will remain constant for the life of the object.

Property	Description
Label	This edit field specifies the initial text displayed beneath the **iconbutton**. To include a keyboard accelerator in the **iconbutton**’s label text, simply place an ampersand (&) in front of the desired character. To display an ampersand in the text, use two ampersands. For example, the label “***&R&&D***” would display as “***R&D***,” with the ***R*** appearing as the underlined accelerator.
ID	This field requires a constant, in the range of 1 to 2147483647. The ID is used within a **uwincreate** statement group to reference the **iconbutton**, and is returned in the $OBJECT system variable when the **iconbutton** is selected. Each object within a *User window* must have a control ID, and each ID must be unique.
Filename	This edit field specifies the file containing the icon graphic. Click on the ***Browse*** button to locate the desired file. Note that icon graphics can be accessed in executable programs (**.exe**), DLL modules (**.dll**), and icon libraries (**.ico**, **.icl**, and **.nil** files).
Index	Windows programs or icon files often contain multiple icons. You can use the ***Next*** and ***Previous*** buttons to search through the selected file for the icon you want to use. The ***Index*** field reflects the location of the selected graphic within the source file. If the file is a simple icon file (**.ico**), this value is zero.
Variable Names	The fields in the ***Variable Names*** group box are optional. They allow you to specify variable names for the **iconbutton**’s label, filename, ID, index, and location parameters. If you supply variable names in these fields, the variables will be initialized with the appropriate values from the ***Initial Values*** group box and the **iconbutton**’s location values in its **uwincreate** statement. If you do not specify variable names, the **iconbutton**’s properties and location will remain constant throughout its life.
Position	If you select the ***Fixed*** option in the ***Position*** group box, the **iconbutton** will remain in the same location on your window, even if the window is re-sized. If you select the ***Bkg Relative*** option, the **iconbutton** will always maintain the same position relative to the *User window* background.

When you’re satisfied with the settings you’ve selected for your **iconbutton** object, click on ***OK*** to save them and return to the main editing window. If you want to close the **IconButton Object Options** dialog without saving any changes you may have made, click on ***Cancel***.

Icon Object Options

Property	Description
Initial Values	These fields must be constants, specifying the start-up values for the **icon**. If you do not specify variable names in the corresponding fields within the ***Variable Names*** group box, the ***Initial Values*** will remain constant for the life of the **icon**.
Filename	This field specifies the file from which to access the icon graphic. Click on the ***Browse*** button to locate the file. Note that icon graphics can be accessed in executable programs (**.exe**), DLL modules (**.dll**), and icon libraries (**.ico**, **.icl**, and **.nil** files).
ID	This field requires a constant, in the range of 1 to 2147483647. The ID is used within a **uwincreate** statement group to reference the **icon**. Each object within a *User window* must have a control ID, and each *ID* must be unique.
Index	Windows programs or icon files often contain multiple icons. You can use the ***Next*** and ***Previous*** buttons to search through the selected file for the icon you want to use. The ***Index*** field reflects the location of the selected graphic within the source file. If the file is a simple icon file (**.ico**), this value is zero.
Variable Names	The fields in the ***Variable Names*** group box are optional. They allow you to specify variable names for the **icon**’s filename, ID, index, and location parameters. If you supply variable names in these fields, the variables will be initialized with the appropriate values from the ***Initial Values*** group box and the **icon**’s location values in its **uwincreate** statement. If you do not specify variable names, the **icon**’s position and ID will remain constant throughout its life.
Position	If you select the ***Fixed*** option in the ***Position*** group box, the **icon** will remain in the same location on your window, even if the window is re-sized. If you select the ***Bkg Relative*** option, the **icon** will always maintain the same position relative to the *User window* background.

When you’re satisfied with the settings you’ve selected for your **icon** object, click on ***OK*** to save them and return to the main editing window. If you want to close the **Icon Object Options** dialog without saving any changes you may have made, click on ***Cancel***.

Metafile Object Options

Property	Description
Initial Values	These fields must be constants, specifying the start-up values for the **metafile**. If you do not specify variable names in the corresponding fields within the ***Variable Names*** group box, the ***Initial Values*** will remain constant for the life of the **metafile**.
Filename	This field specifies the metafile to display. Click on the ***Browse*** button to locate the desired file.
ID	This field requires a constant in the range of 1 to 2147483647. The ID is used within a **uwincreate** statement group to reference the **metafile**. Each object within a *User window* must have a control ID, and each ID must be unique.
Variable Names	The fields in the ***Variable Names*** group box are optional. They allow you to specify variable names for the **metafile**'s filename, ID, size, and location parameters. If you supply variable names in these fields, the variables will be initialized with the appropriate values from the ***Initial Values*** group box and the **metafile**'s size, position, and ID values in its **uwincreate** statement. If you do not specify variable names, the **metafile**'s filename, size, location, and ID will remain constant throughout its life.
Position	If you select the ***Fixed*** option in the ***Position***, the **metafile** will remain the same size and in the same location on your window, even if the window is re-sized. If you select the ***Bkg Relative*** option, the **metafile** will always maintain the same position and size relative to the *User window* background.

When you're satisfied with the settings you've selected for your **metafile** object, click on ***OK*** to save them and return to the main editing window. If you want to close the **Metafile Object Options** dialog without saving any changes you may have made, click on ***Cancel***.

Bitmap Object Options

Property	Description
Initial Values	These fields must be constants, specifying the start-up values for the **bitmap**. If you do not specify variable names in the corresponding fields within the ***Variable Names*** group box, the ***Initial Values*** will remain constant for the life of the **bitmap**.
Label	This field specifies the initial text displayed beneath the **bitmap** object. To include a keyboard accelerator in the **bitmap**'s label text, simply place an ampersand (&) in front of the desired character. To display an ampersand in the text, use two ampersands. For example, the label "***&R&&D***" would display as "***R&D***," with the ***R*** appearing as an underlined accelerator.
Filename	This field specifies the bitmap file to display. Click on the ***Browse*** button to locate the file.
ID	This field requires a constant in the range of 1 to 2147483647. The ID is used within a **uwincreate** statement group to reference the **bitmap**, and is returned to the $OBJECT system variable when the **bitmap** is clicked in the *User window*. Each object within a *User window* must have a control ID, and each ID must be unique.
Variable Names	The fields in the ***Variable Names*** group box are optional. They allow you to specify variable names for the **bitmap**'s filename, ID, size, and location parameters. If you supply variable names in these fields, the variables will be initialized with the appropriate values from the ***Initial Values*** group box and the **bitmap**'s size and location values in its **uwincreate** statement. If you do not specify variable names, the **bitmap**'s filename, size, position, and ID will remain constant throughout its life.
Position	If you select the ***Fixed*** option in the ***Position*** group box, the **bitmap** will remain in the same location on your window, even if the window is re-sized. If you select the ***Bkg Relative*** option, the **bitmap** will always maintain the same position relative to the *User window* background.

When you're satisfied with the settings you've selected for your **bitmap** object, click on ***OK*** to save them and return to the main editing window. If you want to close the **Bitmap Object Options** dialog without saving any changes you may have made, click on ***Cancel***.

DLLObject Options

Property	
Initial Values	These fields must be constants, specifying the start-up values for the **dllobject**. If you do not specify variable names in the corresponding fields within the ***Variable Names*** group box, the ***Initial Values*** will remain constant for the life of the **dllobject**.
Global Filename	This field requires a string constant specifying the drive, path and filename of the **.dll** to be associated with all **dllobject**s used by the script. You can click on ***Browse*** to search for the **.dll** you want.
ID	This field requires a constant in the range of 1 to 2147483647. The ID is used within a **uwincreate** statement group to reference the **dllobject**. Each object within a *User window* must have a control ID, and each ID must be unique.
Variable Names	The fields in the ***Variable Names*** group box are optional. They allow you to specify variable names for the **dllobject**'s filename, ID, size, and location parameters. If you supply variable names in these fields, the variables will be initialized with the appropriate values from the ***Initial Values*** group box and the **dllobject**'s size and location values in its **uwincreate** statement. If you do not specify variable names, the **dllobject**'s properties will remain constant throughout its life.
Position	If you select the ***Fixed*** option in the ***Position*** group box, the **dllobject** will remain the same size and in the same location on your window, even if the window is re-sized. If you select the ***Bkg Relative*** option, the **dllobject** will always maintain the same position and size relative to the *User window* background.

When you're satisfied with the settings you've selected for your **dllobject**, click on ***OK*** to save them and return to the main editing window. If you want to close the **DLLObject Options** dialog without saving any changes you may have made, click on ***Cancel***.

Hotspot Object Options

Property	Description
Initial Value ID	This field requires a constant in the range of 1 to 2147483647. The ID is used within a **uwincreate** statement group to reference the **hotspot**, and is returned to the $OBJECT system variable when the **hotspot** is clicked-on in the *User window*. Each object within a *User window* must have a control ID, and each ID must be unique.
Variable Names	The fields in the ***Variable Names*** group box are optional. They allow you to specify variable names for the **hotspot**'s ID, size, and location parameters. If you supply variable names in these fields, the variables will be initialized with the appropriate values from the ***Initial Values ID*** field, and the **hotspot**'s size and location values in its **uwincreate** statement. Otherwise, the **hotspot**'s size, position, and ID will remain constant throughout its life.

When you're satisfied with the settings you've selected for your **hotspot** object, click on ***OK*** to save them and return to the main editing window. If you want to close the **Hotspot Object Options** dialog without saving any changes you may have made, click on ***Cancel***.

A Sample User Window

The sample *User window* described below was constructed with the *User Window Editor*. The window uses a bitmap background, with **pushbutton**s for signaling script actions.

The **uwincreate** group generated by the *User Window Editor* looks like this:

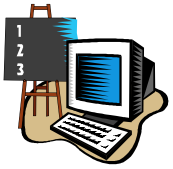
uwincreate full window 100 143 82 92 bitmap

bitmapbkg center center memory "C:\WINDOWS\MEDIA\C&CNOD1.bmp"

pushbutton 100 78 3486 1074 235 "&Enter Labyrinth" background

pushbutton 101 4209 3486 1074 235 "&Return to Base" background

Note that these statements don't form a complete script, but they can be easily included in your scripts! For example, commands can easily be placed in the ASPECT source file perform other script actions in response to these *User window* buttons.



Placing a Control in the Dialog Editor

With the Dialog Editor's "point and click" interface, placing and sizing any dialog control is a simple task. Just follow these steps:

1. **Select the control from the editor's *Controls* menu.**

2. **Move the mouse pointer to the control's approximate location in the dialog.**

   As you move the pointer over the dialog, you'll notice that it changes into a crossbar, indicating that you're placing a control. Move the crossbar to the approximate location for the control's top left corner.

3. **Click once in the work dialog, anywhere beneath its title bar.**

   The control's left top corner is placed in the location you clicked.

   When you place a new control, the *Dialog Editor* assigns it the default size and characteristics for that control. **text** controls, for example, are initially given a constant value of "***Text***," while **radiobutton**s and **checkbox**es are initialized to "***Label***." **bitmaps** and **metafiles** are initialized to "***BMP***" and "***WMF***," respectively. **pushbuttons** are given a constant value of "***Push***."

4. **Adjust the control's size or placement as needed.**

   After you place a control, it remains selected, allowing you to easily re-size it or modify its characteristics. Drag handles are displayed on the selected control, allowing you to change its size by dragging a handle in the appropriate direction.

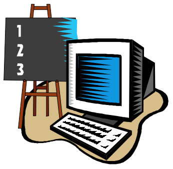
   You can "fine tune" the control's placement with the cursor keys, or use the mouse to drag the control to a new location. "*Moving a Control*" on page 517 offers further explanation.

   Some controls, such as **icon**s and **iconbutton**s, are not re-sizable. See "*Re-sizing a Control*" on page 517 for further information.

5. **Modify the control's characteristics or settings.**

   Double-click on the control to open its options dialog and adjust its settings. The **Options** dialog for each type of ASPECT dialog control is discussed in "*Control Properties*" on page 518.

   Depending on the type of control you're placing, you may find that you need to adjust its size or placement after you've changed its settings.



Placing a User Window Object in the Editor

With the *User Window Editor*'s "point and click" interface, placing and sizing any object is a simple task:

1. **Select the object from the editor's *Objects* menu.**
2. **Move the mouse pointer to the control's approximate location in the *User window.***

   As you move the pointer over the *User window*, you'll notice that it changes into a crossbar, indicating that you're placing an object. Move the crossbar to the approximate location for the object's top left corner.
3. **Click once in the *User window* to place the object.**

   The object's top left corner is placed in the location you clicked.

   When you place a new object, the *User Window Editor* assigns it the default size and characteristics for that object. **pushbutton** objects, for example, are initially given a constant label of "***Push***."
4. **Adjust the object's size or placement as needed.**

   After you place an object, it remains selected, allowing you to easily re-size it or modify its characteristics. Drag handles are displayed on the selected object, allowing you to change its size by dragging a handle in the appropriate direction.

   Some objects, such as **icon**s and **iconbutton**s, are not re-sizable. For more information, see "*Re-sizing an Object*" on page 540.

   You can "fine tune" the object's placement with the cursor keys, or use the mouse to drag the object to a new location. See "*Moving an Object*" on page 541 for additional information.
5. **Modify the object's characteristics or settings.**

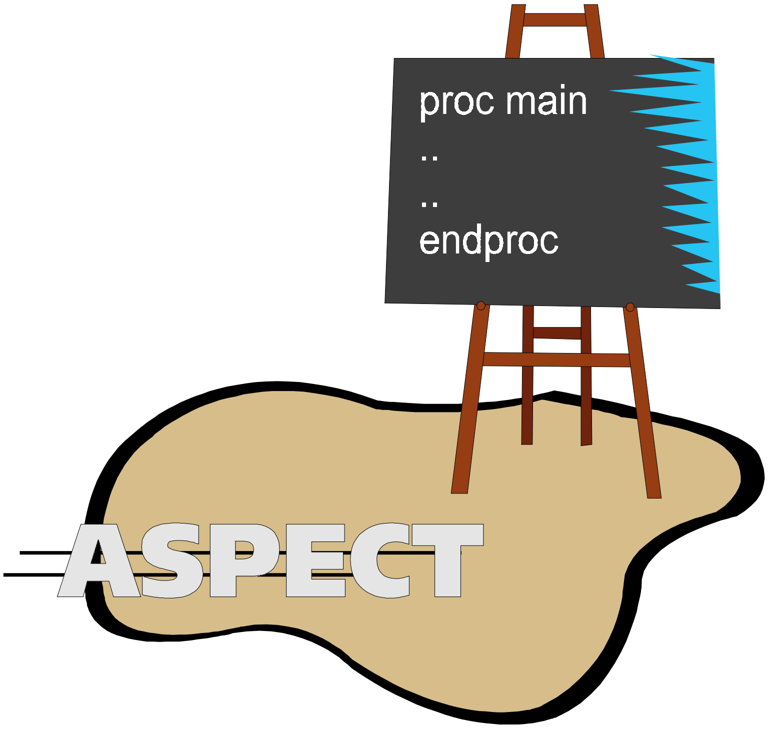
   Double-click on the object to open its options dialog and adjust its settings.

   The **Options** dialog for each type of ASPECT *User window* object is discussed in "*User Window Objects*" on page 540.

   Depending on the type of object you're placing, you may find that you need to adjust its size or placement after you've changed its settings.

Chapter 8

Debugging Scripts



Introduction

It’s rare that a programmer in any language writes a program that compiles and works perfectly the first time! Odds are, the program will have logic, syntax, and/or flow errors. These glitches or errors are referred to as bugs and the process of tracking them down and fixing them is called *debugging*.

This section introduces you to the ASPECT debugging process, provides information about the debugging tools included with ASPECT, and offers coding tips that are helpful for the debugging process. It describes errors that can occur during the compilation or execution of your ASPECT script, and techniques you can use to isolate them.

There are three kinds of errors that can commonly occur as you are developing a script. They are:

- “*Compile-Time or Syntax Errors*”
- “*Execution or Run-Time Errors*”
- “*Logic Errors*”

Compile-Time or Syntax Errors

Before your ASPECT script can be executed by *Procomm Plus*, it must be translated into symbols that Procomm Plus can understand. This translation is performed by the *ASPECT Compiler*.

The *ASPECT Compiler* reads your source **.was** file(s) line-by-line, encoding your commands, constants, and variables into an executable **.wax** file. If the compiler encounters a line in a source file that it cannot interpret, it posts an error message in its ***Message List*** window. Hence the term “*Compile-Time error*.”

For example, consider this simple script:

```
; Script to demonstrate compiler errors
proc Main
string mystring = "Example."
  usermsg "Why won't this work? %s" mmystring
endproc
```

This script contains a simple mistake: the "***mmystring***" argument to the **usermsg** command is misspelled. The *ASPECT Compiler* is not allowed to "guess." Its only option is to report "***mmystring***" as invalid, with as much information as it can provide.

Let's take a closer look at the message produced by the compiler and what information it provides. The compiler provides valuable information:

1. The source file which contains the error.
2. The number of the line in which the error occurs.
3. The kind of error encountered.
4. The offending token involved in the error (this piece of information may not be reported, depending on the error).

If you have more than one error, the compiler provides you with this information for each one. With this information, you can begin debugging your script. Solving compile-time errors can be frustrating when you're trying to learn a new programming language, but be patient. As you become more acquainted with ASPECT, you'll find that it becomes easier and easier. To accelerate the learning process, we've included a step-by-step guide to "*Resolving Compile-Time Errors*" on page 565. We've also included a complete list of "*Compile-Time Error Messages*" on page 572. You may also encounter *Execution or Run-Time Errors*.

Execution or Run-Time Errors

The *ASPECT Compiler* translates or compiles your human-readable script source **.was** file(s) into a computer readable **.wax** script file that Procomm Plus can run. The compiler cannot anticipate some problems that may occur while the script is running, however. Once you've compiled your script, you do not have to worry about compile-time errors, but that doesn't mean that your script is problem free.

After you've compiled your script, if Procomm Plus encounters a problem, it reports a *run-time error*. Run-time errors occur in two variants, which are labeled "critical" or "non-critical." While *critical* run-time errors are rare, they signal potentially dangerous conditions within the script execution routines. Procomm Plus will always terminate a script if a critical error occurs.

Non-critical errors are less serious. They are often the result of logic errors within the script; for more information, see "*Logic Errors*" on page 562. If Procomm Plus encounters a non-critical error, it prompts for confirmation before continuing script execution. It isn't required, but it's usually a good idea not to continue a script after a non-critical error, since the problem may cause unintended results or even produce a critical error.

If you receive a run-time error, you should note the error message and attempt to resolve it. We have provided a list of "*ASPECT Run-Time Error Messages*" on page 580 with a brief explanation of each message. This information will help you debug the script.

Resolving Run-Time Errors

Resolving a run-time error is generally more difficult than resolving a compile-time error. Because ASPECT scripts are compiled, a run-time error will not be directly connected to statement syntax. You need to learn more about the error before you can to resolve it.

The script below includes a fairly common logic error:

```
; Demonstration of Run-Time error.
proc Main
string teststr = ""        ; Define test string, init to null
string showstr             ; Define string for display
integer testlen            ; Define int. to report test length
integer bad_index          ; Define int. index to test string
  clear                  ; Eliminate screen clutter.
  for bad_index = 0 upto 256;This is wrong! The String length
                         ; must not exceed 256 Characters!
    strputc teststr bad_index '&';
    strlen teststr testlen; Get the length of the string.
    strfmt showstr "`"TestStr`" is %d chars." testlen
                         ; Format and show the string.
    locate 1 1           ; Move the cursor
    termwrites showstr   ; Write the string to the Terminal.
  endfor
endproc
```

Strings in ASPECT are limited to 256 characters, but their character indexes are zero-based. This means that the proper **strputc** command to target the last string position is:

```
strputc teststr 255 ‘&’    ; Target the last char position
```

Instead, the **for** loop increments the ***bad_index*** variable from 0 to 256. The script executes normally until ***bad_index*** equals 256, at which point

```
strputc teststr 256 ‘&’    ; Bad Index value!
```

is targeting a *non-existent* 257th character position in ***teststr***.

Let's use what we already know about the run-time error in this script to help us learn how to resolve other run-time errors. It becomes easier to solve run-time problems when you approach them using five processes:

- *Review the run-time error message.*
- *Compile the script for debug.*
- *Repeat the actions that caused the error.*
- *Use the debugger's information.*
- *Use the breakpoint command.*

Review the run-time error message.

In our example, Procomm Plus displayed the "Error 1: Value out of range" error message. ***Error 1: Value out of range*** indicates that an argument value given to one of the commands in the example was not valid. Because our example script is quite brief, it's easy to examine it line-by-line, and verify the arguments for each of the commands.

In larger, more complex script projects, line-by-line evaluation is impractical. When searching for causes in such scripts, consider:

- What portion of the script was executing when the error occurred?
- What commands in the script relate to the run-time error?

  For example, a string-related command is an unlikely culprit for an ***Error 02: Divide by zero***, while a math-related command is unlikely to cause an ***Error 04: String length limit exceeded***.
- If the script has run previously without error, what has changed? Have you added new commands or procedures to the script?

If the cause of the problem is not clear, you should recompile the script using the *ASPECT Compiler's* debug option.

Compile the script for debug.

Procomm Plus' debugging utility assists you in tracking down problems in complex scripts. To use the debugger, however, you'll need to re-compile the script with the ***Compile for debug*** check box enabled. It is enabled from the **ASPECT Compiler Options** dialog.

If you are within Procomm Plus, you can access the **ASPECT Compiler Options** by pressing <AlT><F3> to open the **Compile/Edit ASPECT File** dialog, and then clicking the ***Compile Options*** button. If you are using the *ASPECT Editor* to edit your script files, you can open the **ASPECT Compiler Options** dialog by selecting ***Tools | Compiler Options*** from the menu.

The ***Compile for debug*** option, if enabled, instructs the compiler to include debugging information within the compiled script. Procomm Plus displays this information in the debugging window if a run-time error occurs, or if the script is aborted. After you've enabled the ***Compile for debug*** option, click on ***OK*** to close the dialog. Compile the script again for a debug version.

*The **Create map file option** instructs the ASPECT Compiler to generate a text file with the same name as your **.was** file, but with a file extension of **.map**. The text file contains details about your compiled script, the script's procedures, and functions, and reports the number of times each procedure or function is called throughout the script.*

*The **Include line numbers** option is only available if you're generating a **.map** file. It instructs the ASPECT Compiler to include a line number table in the **.map** file. If a run-time error occurs, you can use the line number table to locate the approximate line in the source file that is causing the problem.*

Repeat the actions that caused the error.

The example script requires no user interaction to force its run-time error. However, if your script relies on user input or other variable conditions, you'll need to reproduce the conditions that led to the run-time error.

After compiling for debug, Procomm Plus responds to a run-time error by displaying the **ASPECT Run Time Debugger** dialog.

*If a script has been compiled for debugging, you can activate the **ASPECT Run Time Debugger** dialog at any time during execution by pressing <Ctrl><Break>. If the script is currently executing a command that can be exited with <Ctrl><Break>, the command will be canceled when script execution resumes.*

Use the debugger's information.

The debugger window displays information you may need to track down a run-time error message:

- The error message generated by the problem.
- The name of the procedure or function executing when the error occurred.

- The offset into the target **.wax** file where the error occurred.
- The name of the source **.was** file.
- The offending line of code from the source file.
- The text of the line in the source file containing the offending code.

In our example script, the debugger shows line number 10:

strputc teststr bad_index '&'

as the offending source line. Since our error message relates to an invalid range, we know that either the *index* argument ***bad_index***, or the character argument '***&***' to **strputc** is at fault. The ampersand '***&***' enclosed in single quotes is a valid character, so the value of **bad_index** must be the problem.

If the value of **bad_index** is invalid, what is it? You could determine the value by inserting a message command of some kind in the script before the **strputc** line. Then, when the script executed, you would know that the last value reported for **bad_index** before the error was invalid. The debugger, however, gives you an easier method.

Within the **ASPECT Run-Time Debugger** dialog is the ***Variable*** drop-down list box. By default, ***Predefined*** variables are displayed, but you can select ***Global*** or ***Local*** variables as well. Simply click on the list and select the variable name, or type of variable, you want displayed. Once you've selected a variable, its contents or value is automatically displayed.

Additionally, you should note the ***Next*** and ***Prev*** buttons. They are used for displaying the values of arrays. If you select an array variable, the ***Array Offset*** field and the ***Next*** and ***Prev*** buttons are enabled. This allows you to look through the array's elements for problem values.

If your script has caused a run-time error, there are several other buttons available in the debugger window that can help you obtain more information:

- The ***Continue*** button attempts the continuation of the suspended script. The debugger dialog will close and the script will resume normally.
- The ***Step*** button continues execution of the suspended script on a line-by-line basis. The debugger dialog remains open as the script executes, allowing you to monitor the selected ***Variable***'s value, the procedure, the source file, and the source code line.
- The ***Edit Source*** button opens the source **.was** file for editing. If you're using the *ASPECT Editor*, it will place the cursor on the source line indicated in the debugging dialog.
- The ***Exit Script*** button terminates the script without resuming execution. The debugger dialog will close.

If you're unable to discover the source of the problem using the information provided by the debugger dialog, you may need to use the **breakpoint** command.

*Use the **breakpoint** command.*

The **breakpoint** command allows you to open the debugger before the problem occurs. It suspends script execution and opens the **ASPECT Run Time Debugger** dialog. Now you can look “under the hood” of your script *before* the run-time error occurs.

When the sample script encountered its run-time error and opened the **ASPECT Run Time Debugger**, and you selected **bad_index**, its value was already 256. By using a **breakpoint** command, you can use the ***Step*** button to monitor **bad_index**’s value and detect when the error occurs.

You can insert the **breakpoint** command anywhere in your script. However, it is generally placed at the start or end of a procedure or function, or around a suspect block of code. You can also specify a message to be posted in the debugger’s ***Message*** field to help you determine which **breakpoint** has fired.

A **breakpoint** command is ignored by the compiler when a script is compiled without the ***Compile for Debug*** option enabled, making it unnecessary to insert conditional-compilation code around each **breakpoint**! If you’re still having problems, it could be due to logic errors.

Logic Errors

The hardest program errors to track down are those within a script that executes without reported errors, but does not produce the expected results. Typically, the problem is caused by a mistake in logic, or by a variable being assigned an improper value.

One method of debugging such an error is to “watch” critical program variables as the script runs. The **usermsg, errormsg**, **sdlgmsgbox**, and **statmsg** commands provide ways to monitor variables during script execution. For example:

```
proc Main
integer x = 256, y, z
string msg = "This is a string"
  x -= 1
  y = x >> 4
  z = y % 2
  usermsg "x = %d y = %d z = %d msg = %s" x y x msg
  ; The usermsg dialog won’t close until you press a button.
endproc
```

The **statmsg** command allows you to monitor variables while the script runs, without stopping the execution for user input. For example:

```
proc Main
integer x = 256, y, z
string msg = "This is a string"
  x -= 1
  y = x >> 4
  z = y % 2
  statmsg "x = %d y = %d z = %d msg = %s" x y x msg
  ; Script execution is not paused by the statmsg command.
endproc
```

Even something as simple as the **beep** command can be used effectively during the debugging process. For instance, in debugging a **when** procedure that handles an asynchronous event, you may simply wish to know approximately how often that procedure is called. An audible debug signal is ideal for this purpose.

You can also use the *ASPECT Run-Time Debugger* to monitor a variable's value as a script executes. Information about the debugger and using the **breakpoint** command to monitor script flow are described in "*Resolving Run-Time Errors*" on page 558.

Conditional Compilation

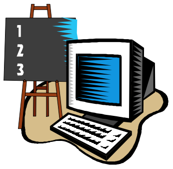
You can include debugging code in your script with preprocessor and macro **#define** statements. For example, in the script fragment below:

```
#define DEBUG            ; #Define this macro if debugging
proc Main
string today
  today = $DATE
  #ifdef DEBUG; If debugging, display message
    usermsg "`"The returned date was %s`"" today
  #endif
  .
endproc
```

the *DEBUG* macro would be defined, or active if this script were compiled now. Thus, the **usermsg** placed between the **#ifdef** DEBUG and the **#endif** commands would be processed and included in the compiled script. When the script executed, the **usermsg** dialog would be displayed. However, if a comment symbol (;) were placed in front of the **#define** statement:

```
; #define DEBUG            ; #Define this macro if debugging
```

the *DEBUG* macro would be undefined, or inactive, and any commands placed between the **#ifdef** DEBUG and the **#endif** commands would be ignored by the compiler. For even more automated conditional compilation, you may wish to use one of ASPECT's Predefined Macro Definitions: ASPDEBUG.



Resolving Compile-Time Errors

Compile-time errors are usually generated by statements that contain one or more syntax errors, just like the error in the "Compile-Time or Syntax Errors" sample script. Compile-time errors are usually easy to resolve since the ASPECT Compiler detects and reports them as it processes your source file(s).

To resolve compile-time errors, follow these steps:

1. **Open the source file that contains the error.**

   If the error occurs in the file shown in the ***Compile Progress*** group box, you can click on the ***Edit Source File*** button. The *ASPECT Compiler* will automatically load the file into your editor. If the error is within a **#include** file, you will have to open it using conventional means.

   Do ***not*** close the **ASPECT Script Compiler** window.

2. **Go to the line number indicated in the error message.**

   If you're using the *ASPECT Editor*, you can press <Ctrl><G> and enter the appropriate line number in the **Goto Line** dialog. The *ASPECT Editor* moves the cursor to the specified line when you click ***OK***.

   It is important that you always start with the *first* error shown in the ***Message List*** window. A single syntax error can generate multiple Compile-Time error messages.

3. **Examine the error message given by the *ASPECT Script Compiler.***

   The script sample in "*Compile-Time or Syntax Errors*" on page 556 caused the compiler to report an "***Invalid expression token***," because the string variable **mystring** was misspelled as ***mmystring***. When the compiler tried to interpret the ***usermsg*** command, it couldn't find a valid variable declaration.

4. **Locate the word or symbol flagged by the compiler.**

   In the example, "***mmystring***" was flagged because the compiler couldn't find a string variable by that name.

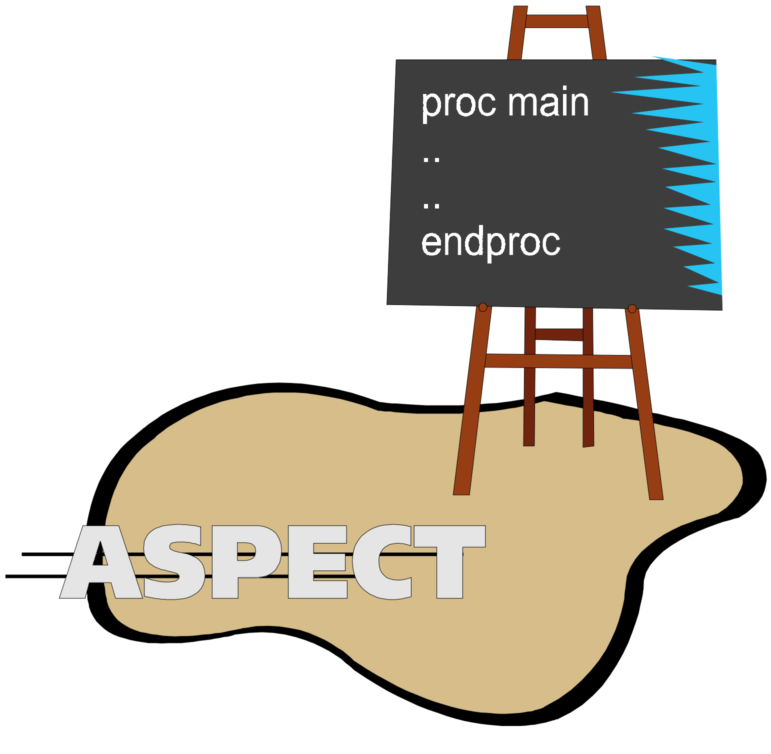
   There are, however, a variety of things that can cause compile-time errors. If, after comparing the error message to the source line, you're still uncertain about the meaning of the error message, or where the problem is, try comparing the command syntax you've used against its formal syntax.

   Within the *ASPECT Editor*, you can quickly open a command's help reference by placing the cursor on the command right-clicking. Use these references to make sure that you haven’t omitted a required keyword, or tried to use a data type that is not allowed for the command.

A complete list of compile-time errors appears in “*Compile-Time Error Messages*” on page 572.

Appendix A

The ASPECT Compiler



Introduction

he *ASPECT Compiler* offers a high level of input, configuration, and feedback. Here, we offer a detailed view of the *ASPECT Compiler*. We include explanations of "*The Compilation Process*," the various "*Compiler Options*" available, and the "*Command-Line Syntax*"for the compiler executable. Additionally, there are complete lists of "*Compile-Time Error Messages*" on page 572, "*Compiler Warning Messages*" on page 579, and "*ASPECT Run-Time Error Messages*" on page 580.

The ASPECT Compiler

The *ASPECT Compiler* is designed for speed, flexibility, and ease of use. Its options and settings allow you to specify the amount of information reported to you as your scripts are processed, and to control the information optionally stored for debugging activity.

The *ASPECT Compiler* resides in the Procomm Plus **\programs** directory. It can be run from any application, not just Procomm Plus. For instance, you could select ***Start | Run...*** , type in "***...\ASPCOMPW***" followed by a script filename, and press <Enter> to launch the compiler. Additionally, if an application provides "programmable" keys or buttons, you could configure one of them to activate the compiler. Only Procomm Plus, however, allows you to execute a compiled script.

The Compilation Process

The *ASPECT Compiler* is a three-pass compiler. This means that it scans the script source **.was** file three times before producing the compiled target script file.

The first compilation pass is a quick scan through the file to determine the names of all defined procedures and functions, and their respective calling conventions in terms of parameter ordering and data type. In the case of user-defined functions, the return type is also determined. This pass is called the ***prototyping pass***, and is required to insure that the next compilation pass knows how to correctly parse and verify calls to these procedures and functions.

The second compilation pass performs a ***syntactical scan*** of the source file. All variable references and command, procedure, and function arguments are verified for correctness.

If the first and second passes complete without errors, the third compilation pass, referred to as ***code generation***, produces the compiled target script file. A preexisting target file will not be overwritten if the first or second pass produced errors and halted the compilation.

Compiler Options

The *ASPECT Compiler* options allow you to control the kinds of warning messages reported to you as your script is processed by the compiler. They also allow you to enable the amount of debugging information generated by the compiler, and to specify additional options for the compiler command-line.

To configure the compiler options, open the **Compile/Edit ASPECT File** dialog, either by pressing <Alt><F3> or by clicking on the ***Compile/Edit*** button on the *Action Bar*. Then, click on the ***Compile Options*** button to open the **ASPECT Compiler Options** dialog, where there are four groups of options.

Warning Level

The ***Warning Level*** setting determines whether the compiler reports warnings, in addition to any syntax errors encountered in the script. A compiler warning will not prevent a script from compiling and running, but it may be an indication of a potential problem within the script.

- **Level 0.** No warnings of any kind are reported at this level.
- **Level 1.** At this level, unreferenced variables and labels are reported. The *ASPECT Compiler* defaults to ***Level 1***.
- **Level 2.** At Level 2, unreferenced procedures and functions are reported, in addition to unreferenced variables and labels.

Other Options

The ***Other Options*** settings group allow you to specify the kinds of debugging information generated by the compiler as your script is processed. It also includes a setting for the number of errors the compiler will process before terminating a compile attempt. The debugging options are discussed in more detail in “*Resolving Run-Time Errors*” on page 558.

- ***Compile for Debug***

  If this box is checked, the *ASPECT Compiler* will embed debugging information in the compiled script. If the script contains **breakpoint** commands, selecting this option will cause them to “fire” during script execution.

- ***Create map file***

  If this box is checked, the *ASPECT Compiler* will generate a map file along with the compiled script. The map file, with an extension of **.map**, will be created in the same directory as the target source file. A **.map** file contains detailed information about the generated **.wax** file, including user-defined variables, procedures and functions, and global variable usage.

- ***Include line numbers***

If this box is checked, the compiler will include a table specifying source line numbers and associated file offsets in the compiled script. This option is only available if ***Create map file*** is enabled.

- ***Maximum errors before terminating***

  This field specifies the number of compile-time errors the *ASPECT Compiler* will allow before terminating the compile attempt. By default, this value is 20.

ASPECT Include File Path

This edit field allows you to specify directories that the *ASPECT Compiler* will search for **#include** files that are not found in the main script file's directory. You can enter multiple paths by separating them with a semicolon. For example:

c:\aspect;c:\aspect\shared;d:\aspmisc

Additional Options and Macro Definitions

These edit fields allow you to specify options and macro definitions to be passed to the compiler as command-line arguments. The options that can be specified in these fields are discussed in "*Command-Line Syntax*" on page 570.

- Additional Options and Macro Definitions (saved).

  Options specified in this field will be saved whenever the dialog is closed via the ***OK*** button. They are stored in the **pw4.ini** file in the ***[ASPCOMPW]*** section, under the ***Options*=** entry, and used for future script compilations.

- Additional Options and Macro Definitions (not saved).

  Options in this field will be referenced only in the current session, allowing you to easily add or remove compile options temporarily, without affecting the **.ini** compiler settings.

Command-Line Syntax

The *ASPECT Compiler* executable, **aspcompw.exe**, resides in the **programs** directory under the Procomm Plus home directory. You can start it using the following syntax:

ASPCOMPW [options] filespec

Where **[options]** may include:

Switch	Description
/?	Display syntax screen

Switch	Description
/Dx[=text]	Define macro ***x*** [and initialize to ***text***]
/En	Set maximum allowable error count to ***n*** (default 20)
/ES	Save messages to an error file, with an **.err** extension
/F	Generate far calls to procedures and functions
/Ipath1[;path2] [;...]	Specify the directories searched to find **#include** files.
/M[L]	Generate symbol map [with source line reference table]
/N	Ignore the command line options in **pw4.ini**.
/O	Disable compiler optimizations
/Q	Quiet operation mode, with no screen output
/S	Compile with syntax check only
/Wn	Warning level ***n*** where ***n*** = 0, 1 (default), or 2
/X	Exit immediately after successful compile
/Z	Compile script with run-time debugging information

The *ASPECT Compiler* reports the following exit code values upon termination:

Value	Description
0	Normal termination, indicating a successful compile
-1	Error termination, indicating an unsuccessful compile
-2	Abort termination, indicating that the user closed the compiler window during compile
-3	Abort termination, indicating that the user ended the Windows session during compile

The *ASPECT Compiler* accepts a maximum of twenty command line options. Options, or arguments, are parsed according to the following rules:

- Arguments are delimited by white space, meaning tab or space characters.
- A string surrounded by double quotes is interpreted as a single argument, regardless of white space it may contain. The quotes surrounding the string will not be considered part of the argument after it is parsed.
- A double quote preceded by a backslash (\) character is interpreted as a literal double quote (") character, and will be included in the argument with the backslash omitted.

Quotes would typically be used only when defining macros on the command line. Quotes are necessary if the macro text consists of more than one token. If the text itself should be enclosed in double quotes, such as when defining a macro as a literal string, then escape the quote characters with a backslash. For example:

```
/Dmac="1 2 3"           ; is equivalent to #define mac 1 2 3
/Dmac="\"a b c\""       ; is equivalent to #define mac "a b c"
/Dmac="\"a `\"b`\" c\""      ; is equivalent to #define mac "a `"b`" c"
```

In the last example, back tick characters (` -ASCII 96) precede the innermost double quotes. A back tick character followed by a double quote is an ASPECT escape sequence, and is required when embedding quotes within literal strings. When the compiler parses this string, it will be equivalent to: a "b" c.

Compile-Time Error Messages

These error messages are reported at compile time by the *ASPECT Compiler*. For help debugging compile-time error messages, see "*Compile-Time or Syntax Errors*" on page 556.

Compilation continues when one of these error messages is reported, but the error must be resolved before a target ***.wax*** *script can be created. When displayed in the ASPECT Compiler window, the error message will include a line number indicating the location of the error in the source file.*	
C000	Illegal character
C001	Missing terminating quote
C002	Missing space between tokens
C003	Unexpected escape sequence
C004	Maximum token length exceeded
See "*Compile-Time or Syntax Errors*" on page 556 for more information.	

*Compilation continues when one of these error messages is reported, but the error must be resolved before a target **.wax** script can be created. When displayed in the ASPECT Compiler window, the error message will include a line number indicating the location of the error in the source file.*	
C005	Escape sequence character out of range
C006	Invalid character constant
C007	New line character in constant
C020	Invalid token
C021	Unexpected token(s) at end of line
C022	Missing token
C023	Invalid expression token
C024	Invalid command or expression token
C025	Unexpected command
C026	Invalid use of reserved word
C027	Invalid identifier or name
C028	Duplicate symbol
C029	Missing comma in token list
C030	Undefined label
C031	Invalid label location
C032	Variable limit exceeded
C033	Variable not defined
See "*Compile-Time or Syntax Errors*" on page 556 for more information.	

Compilation continues when one of these error messages is reported, but the error must be resolved before a target ***.wax*** *script can be created. When displayed in the ASPECT Compiler window, the error message will include a line number indicating the location of the error in the source file.*

C034	Maximum stack displacement exceeded
C050	Invalid preprocessor command
C051	Preprocessor command must occur as first token
C052	Maximum macro definition length exceeded
C053	Maximum macro argument length exceeded
C054	Maximum macro expansion length exceeded
C055	Missing #ENDCOMMENT command
C056	Missing #ENDIF command
C060	Missing ENDSWITCH command
C061	Missing ENDCASE command
C062	Missing ENDIF command
C063	Missing ENDWHILE command
C064	Missing ENDFOR command
C065	Missing PROC or FUNC command
C066	Missing ENDPROC or ENDFUNC command
C080	Missing WITH before argument list

See "*Compile-Time or Syntax Errors*" on page 556 for more information.

Compilation continues when one of these error messages is reported, but the error must be resolved before a target ***.wax*** *script can be created. When displayed in the ASPECT Compiler window, the error message will include a line number indicating the location of the error in the source file.*

C081	Invalid number of arguments
C082	Mismatched argument type
C083	Maximum argument count exceeded
C084	Missing or invalid return type or value
C085	Procedure does not return a value
C086	Undefined procedure or function
C087	Unresolved procedure or function call
C088	Missing procedure MAIN
C100	Expression too complex
C101	Missing operator
C102	Missing operand
C103	Missing right paren
C104	Missing left paren
C105	Missing right bracket
C106	Missing left bracket
C107	Missing variable operand
C108	Missing colon
C109	Unexpected colon

See “*Compile-Time or Syntax Errors*” on page 556 for more information.

Compilation continues when one of these error messages is reported, but the error must be resolved before a target ***.wax*** *script can be created. When displayed in the ASPECT Compiler window, the error message will include a line number indicating the location of the error in the source file.*

C110	Invalid operand
C111	Invalid variable operand
C112	Invalid use of operator
C113	Invalid floating point operation
C114	Illegal initializer expression
C115	Divide by zero
C116	Value out of range
C120	Invalid subscript expression
C121	Invalid constant expression
C122	Subscript on non-array
C123	Maximum array dimensions exceeded
C124	Maximum array elements exceeded
C150	Invalid number
C151	Invalid numeric variable
C152	Invalid numeric constant
C153	Invalid integer
C154	Invalid integer variable

See "*Compile-Time or Syntax Errors*" on page 556 for more information.

*Compilation continues when one of these error messages is reported, but the error must be resolved before a target **.wax** script can be created. When displayed in the ASPECT Compiler window, the error message will include a line number indicating the location of the error in the source file.*

C155	Invalid integer constant
C156	Invalid long
C157	Invalid long variable
C158	Invalid long constant
C159	Invalid float
C160	Invalid float variable
C161	Invalid float constant
C162	Invalid string
C163	Invalid string variable
C164	Invalid string constant
C165	Invalid integer or string
C166	Invalid number or string
C167	Invalid numeric or string variable
C168	Invalid numeric or string constant
C169	Invalid global variable
C200	Duplicate control id
C201	Too many controls in dialog box
C202	Default button already defined

See “*Compile-Time or Syntax Errors*” on page 556 for more information.

*Compilation continues when one of these error messages is reported, but the error must be resolved before a target **.wax** script can be created. When displayed in the ASPECT Compiler window, the error message will include a line number indicating the location of the error in the source file.*

C203	Missing ENDDIALOG command
C204	Missing ENDGROUP command

See “*Compile-Time or Syntax Errors*” on page 556 for more information.

Compilation will terminate immediately if one of these error messages is reported.

C500	Internal compiler error
C501	Insufficient memory
C502	Unexpected end of file
C503	Stack under flow
C504	Stack overflow (nesting too deep)
C505	Procedure too large to compile
C520	Invalid command line option
C521	Missing script filespec
C522	Invalid script file name
C523	Error opening script file

See “*Compile-Time or Syntax Errors*” on page 556 for more information.

Compilation will terminate immediately if one of these error messages is reported.	
C524	Error opening output file
C525	Error writing to output file
C600	Fatal exception
See “*Compile-Time or Syntax Errors*” on page 556 for more information.	

Compiler Warning Messages

Warnings are non-fatal messages generated at compile time by the *ASPECT Compiler*. They are intended to provide information about potential oversights in the script. A target script is still successfully created if one or more of these messages occur. When displayed in the *ASPECT Compiler* window, they will have a line number indicating the location of the possible problem in the source file.

C700	Too many/few arguments in macro expansion
C701	Unreferenced label
C702	Unreferenced parameter
C703	Unreferenced local variable
C704	Unreferenced global variable
C705	Far calls required (use /F command line option)

C800	Unreferenced procedure/function
C801	Function ends without returning a value
C802	Redefinition of user-defined global symbol

ASPECT Run-Time Error Messages

Listed below are the messages displayed by Procomm Plus if an ASPECT error occurs while a script is executing. They are divided into non-critical and critical errors. More information concerning run-time errors and how to resolve them, see "*Execution or Run-Time Errors*" on page 557.

Non-critical errors

Non-critical errors are often the result of logic errors within a script. If Procomm Plus encounters a non-critical error, it prompts for confirmation before continuing script execution.

001 Value out of range

A value was encountered which was not within the valid range of values accepted for a particular command parameter. For example, attempting to **strputc** a value that is outside of the range of valid characters, 0 to 255, into a string will generate this message. This is perhaps the most common error message, and does indicate a problem that should be resolved.

002 Divide by zero

A division (/) or modulus (%) operator attempted to divide a value by zero.

003 Insufficient memory

An ASPECT command could not be completed successfully due to insufficient memory.

004 String length limit exceeded

An operation which updates, concatenates, inserts or otherwise writes to a string variable has attempted to write data past the end of a string's data area. The maximum length of a string in ASPECT is 256 characters.

005 Invalid identifier

An ASPECT command attempted to reference an id, such as a file I/O, memory block, **when**, or **setjmp**/**longjmp** id, that was not valid at the time the command executed.

006 Child scripts nested too deeply

The limit of nested scripts launched with the **execute** command has been reached. This message probably indicates an execution problem with recursion, as the allowed number of nested scripts is 1000!

020 Error creating dialog box

Typically, this message indicates a memory or resource allocation problem.

Critical errors

Critical errors signal potentially dangerous conditions within the script execution routines. Procomm Plus will always terminate a script if a critical error occurs.

101 Insufficient memory

There is not enough memory to run the script file.

102 Stack overflow in procedure call

A procedure or function call was made which required more stack space than was available at the time, either for passed parameters or local variables.

Usually this problem occurs if there are procedure or function calls nested very deeply, as might occur in a recursive call, or procedures or functions with large numbers of local string variables.

103 Unexpected end of file

An attempt to read the next script command from the compiled script failed. See error 106 for more details.

104 File I/O error

A critical error occurred while reading data from the target script file. The data being read was not valid. See error 106 for suggestions.

105 Unable to restore script task

Upon return from a child script, an error occurred preventing the parent script from being completely restored.

106 Invalid command encountered

A command was encountered which is not recognized among ASPECT's command set. This is a serious condition which indicates a corrupted file or an error within the compiler. First, attempt to re-compile the script file and execute it again.

107 Invalid data type encountered

A data type was not recognized as valid. This is a serious condition which indicates a corrupted file or an error within the compiler. First, attempt to re-compile the script file and execute it again.

108 Buffer overflow in string format

A string format operation such as **strfmt** exceeded the maximum length allowed for the output of formatted data. This error is critical in that it may have corrupted internal data within Procomm Plus.

109 Value out of range

This error occurs when a valid value was required for continued successful script execution, but because one was not provided, the run-time engine was in a state where it could not resume execution without the possibility of causing more serious problems.

120 Internal Windows ASPECT run-time error

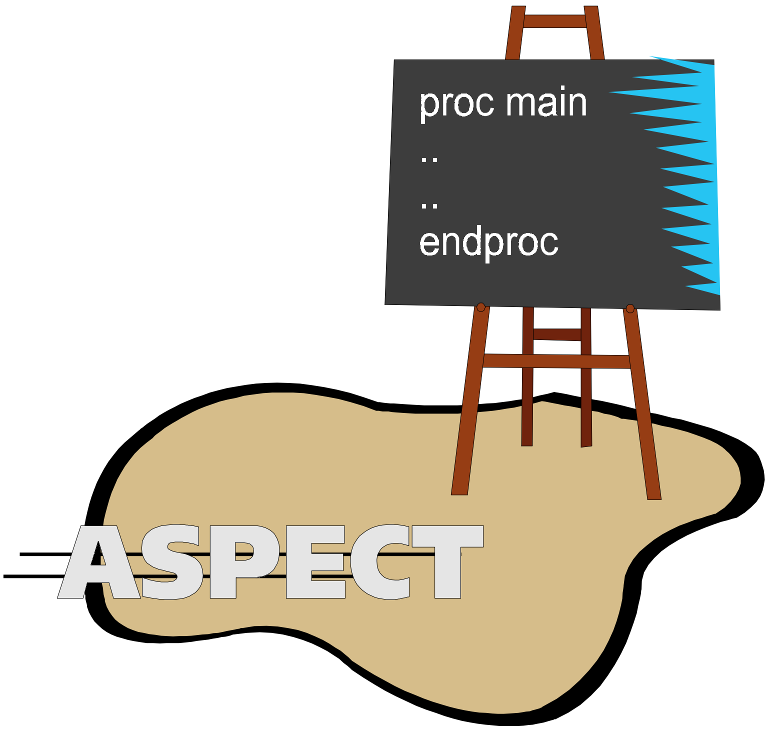
This error indicates a serious problem within the ASPECT run-time processor. First, attempt to re-compile the script file and execute it again.

121 Fatal exception during script execution

This error indicates a serious error within ASPECT or the instance of Procomm Plus that is running the script. Restart Procomm Plus and attempt to run the script again.

Appendix B

ASPECT Reserved Words



Reserved Symbols

The following symbols are reserved in ASPECT:

!

!=

%

%=

&

&&

&=

(

)

*

*=

+

++

+=

,

-

--

-=

/

/=

:

<

<<

<<=

<=

=

==

>

>=

>>

>>=

?

[

]

^

^=

|

|=

||

~

Reserved Words

The following words are reserved in ASPECT:

#COMMENT

#DEFINE

#ELIFDEF

#ELIFNDEF

#ENDCOMMENT

#IFDEF

#IFNDEF

#INCLUDE

#UNDEF

$ACTIONBAR

$ACTIONBARS

$ACTIVEWIN

$ASPECTPATH

$ASPMENU

$CALLERID

$CAPTURE

$CARRIER

$CHAINEDFILE

$CHATWIN

$CNCTMSG

$COL

$COMPANY

$CONNECTOPEN

$CTS

$DATAOPTIONS

$DATE

$DDEADVISE

$DDIRFNAME

$DIALCHANGED

$DIALCONNECT

$DIALDIR

$DIALENTRY

$DIALING

$DIALQUEUE

$DIALSELECT

$DTR

$ERRORNUM

$EXITCODE

$FATTR

$FAXFILE

$FAXMESSAGE

$FAXOPTIONS

$FAXRECVCNT

$FAXSENDCNT

$FAXSTATUS

$FDATE

$FEXT

$FILENAME

$FILESPEC

$FLOWSTATE

$FLTIME

$FNAME

$FOCUSWIN

$FSIZE

$FTIME

$FTPCONNECT

$FTPOPTIONS

$FTPSTATUS

$IPADDRESS

$KEYHIT

$LMOUSEEVENT

$LMOUSESTATE

$LMOUSEWIN

$LMOUSEX

$LMOUSEY

$LPARAM

$LTIME

$MAILOPTIONS

$MAINWIN

$MAPIENABLED

$MCIDEVICEID

$MCINOTIFY

$MESSAGE

$METAKEYS

$MISC

$MMOUSEEVENT

$MMOUSESTATE

$MMOUSEWIN

$MMOUSEX

$MMOUSEY

$MODEMCONNECT

$MONITORWIN

$NEWSOPTIONS

$NEWSSTATUS

$NULLSTR

$NUMCOLORS

$NUMTASKS

$OBJECT

$OS

$OSVER

$PARENTFILE

$PASSWORD

$PKRECV

$PKSEND

$PLAYBACK

$POINTERTASK

$POINTERWIN

$POINTERX

$POINTERY

$PROTOCOL

$PWACTIVE

$PWLOCALPATH

$PWMAINWIN

$PWMENUBAR

$PWMENUDEF

$PWMODE

$PWTASK

$PWTASKPATH

$PWTITLEBAR

$PWVER

$PWWINSTATE

$QUICKSELECT

$RMOUSEEVENT

$RMOUSESTATE

$RMOUSEWIN

$RMOUSEX

$RMOUSEY

$ROW

$RTS

$RXCOUNT

$RXDATA

$SCRIPTENV

$SCRIPTFILE

$SCRIPTMODE

$SCROLLBACK

$SERIALNUM

$SETUP

$SFILENAME

$STATIONID

$STATMSG

$STATUSLINE

$TASK

$TELNETOPTIONS

$TERMCOLORS

$TERMCOLS

$TERMFONT

$TERMINAL

$TERMROWS

$TIME

$TIME24

$TITLEBAR

$TXCOUNT

$TXDATA

$USERDISK

$USERDISKSTR

$USERID

$USERNAME

$USERPATH

$USERWIN

$UWINACTIVE

$VOLUME

$WINCOLORS

$WINPATH

$WPARAM

$WWWSTATUS

$XFERFILE

$XFERSTATUS

$XOFFRECV

$XOFFSENT

$XPIXELS

$YPIXELS

1KXMODEM

1KXMODEMG

2KWINDOW

4KWINDOW

ABORTDNLD

ABORTRETRY

ACCEPTCALL

ACCESS

ACTION

ACTIONBAR

ADAPTIVE19200

ADAPTIVEANS

ADDCALLINFO

ADDFILENAME

ADDRESS

ADDS60

ADDS90

ADM31

ADM3A

ADM5

ALARM

ALARMTIME

ALL

ALT

ALTCTRL

ALTCTRLSHIFT

ALTSHIFT

ALWAYS

ANIMATE

ANONYMOUSLOGON

ANSIBBS

ANSITOKEY

ANSITOOEM

ANSRINGS

APPEND

AREACODE

ASCII

ASCIIXLAT

ASPDEBUG

ASPECT

ASPECTPATH

ASPFILE

ASPLINE

ASPMENU

ASPNOTIFY

ASPPAUSE

ASPVERSION

ATOF

ATOI

ATOL

ATT4410

ATT605

ATTRIBUTE

AUTO

AUTOANSWER

AUTODNLD

AUTOLOGON

AUTOSIZE

AUTOSTART

BACKGROUND

BACKSPACE

BAUDRATE

BEEP

BEGIN

BINARY

BINARYMODE

BIT8MODE

BIT8QUOTE

BITMAP

BITMAPBKG

BLANKEXPAND

BLINKRATE

BLOCK

BLOCKCHECK

BLOCKCURSOR

BLOCKMODE

BLOCKSTART

BOLD

BOTTOM

BREAK

BREAKLEN

BREAKPOINT

BY

CALL

CALLERID

CANCEL

CAPTURE

CAPTURESTR

CASE

CDINXFER

CEIL

CELLULAR

CENTER

CHAIN

CHARACTER

CHARPACE

CHARSET

CHATMODE

CHDIR

CHECKBOX

CHECKGROUPS

CISB

CLASS

CLASS1

CLASS2

CLEAR

CLEARXOFF

CLIPBOARD

CLIPCHAR

CLIPFILERMV

CLIPTOFILE

CLIPTOSTR

CLOSE

CLOSED

CNCTMSG

CODEPAGE

COLORS

COLUMNS

COMBOBOX

COMGETC

COMMANDMODE

COMPANY

COMPILE

COMPLETE

COMPUTC

COMREAD

COMWRITE

CONNECT

CONNECTALL

CONNECTED

CONNECTION

CONNECTMANUAL

CONTENTS

CONTROL

CONVENTIONAL

CONVERTER

COPYFILE

COUNTRY

COVERSHEET

CR

CRASHRECOVER

CRC

CRC16

CRC32

CREATE

CRLFXLAT

CRLF_ETX

CR_LF

CTRL

CTRLBREAK

CTRLQUOTE

CTRLSHIFT

CURRENT

CURSORKEYAPP

CURSORPOS

DATA

DATABASE

DATABITS

DATAKEY

DATE

DBLCLICK

DDEADVISE

DDEEXECUTE

DDEINIT

DDEPOKE

DDEREQUEST

DDETERMINATE

DDEUNADVISE

DEC

DECIMAL

DECLINEWRAP

DECRYPT

DEFAULT

DELETE

DELFILE

DELLINE

DELPAGES

DEST

DEVICE

DGD100

DGD200

DGD210

DIAL

DIALADD

DIALCANCEL

DIALCLASS

DIALCOUNT

DIALCREATE

DIALDELETE

DIALDIR

DIALED

DIALENTRY

DIALFIND

DIALINGBOX

DIALINSERT

DIALLOAD

DIALNAME

DIALNUMBER

DIALNUMBERONLY

DIALOG

DIALOGBOX

DIALSAVE

DIALSTATS

DIM

DIR

DIRECT

DIRLISTBOX

DIRPATH

DISABLE

DISABLED

DISCONNECT

DISK

DISKFREE

DISPLAY

DLGCTRL

DLGCTRLWIN

DLGDESTROY

DLGEVENT

DLGEXISTS

DLGLIST

DLGSAVE

DLGSHOW

DLGUPDATE

DLGWIN

DLGWINCTRL

DLLCALL

DLLFREE

DLLLOAD

DLLOBJECT

DLLOBJFILE

DLLOBJUPDT

DNLD

DNLDPATH

DNLDPROMPT

DOS

DOWNTO

DROPDOWN

DROPDOWNLIST

DROPDTR

DUPLEX

DYNAMIC

EACH

ECHO

ECM

EDITBOX

EDITOR

ELAPSED

ELSE

ELSEIF

ENABLE

ENCRYPT

ENDCASE

ENDDIALOG

ENDFOR

ENDFUNC

ENDGROUP

ENDIF

ENDPROC

ENDSEQUENCE

ENDSWITCH

ENDWHILE

ENQUIRY

ENQUIRYSTR

ENTERCRLF

ENTERKEY

ENTRY

ENVIRONMENT

EOLCHAR

EOLCONVERT

EOLSTR

EOT

ERRORDETECT

ERRORMSG

ESCAPEM

ESPRIT3

ETX

EVEN

EXACT

EXCLAMATION

EXECUTE

EXIT

EXITACTION

EXITFOR

EXITSWITCH

EXITWHILE

EXITWINDOWS

EXTENDED

F0

F1

F2

F3

F4

F5

F6

F7

F8

F9

FAILURE

FAST

FAX

FAXCANCEL

FAXLIST

FAXMODEM

FAXNUMBER

FAXPOLL

FAXPRINT

FAXREMOVE

FAXSEND

FAXSTATUS

FAXVIEW

FAXXMIT

FCLEAR

FCLOSE

FCOMBOBOX

FDELBLOCK

FEDITBOX

FEOF

FERROR

FETCH

FFLUSH

FGETC

FGETS

FILE

FILEGET

FILELIST

FILESET

FILETOCLIP

FILETYPE

FILEVIEW

FILEXFER

FILEXFERBOX

FILTER

FILTERED

FINDFIRST

FINDNEXT

FINISH

FINSBLOCK

FIRST

FIRSTTASK

FIXED

FLENGTH

FLISTBOX

FLOAT

FLOOR

FLOWCONTROL

FLUSH

FONT

FONTNAME

FONTSIZE

FOOTER

FOPEN

FOR

FOREVER

FORWARDADDRESS

FPUTC

FPUTS

FRAME

FREAD

FSEEK

FSTRFMT

FTELL

FTEXT

FTOA

FTP

FTRUNCATE

FULL

FULLPATH

FUNC

FWRITE

GETCUR

GETDIR

GETENV

GETFILE

GETFILENAME

GETPATHNAME

GETVOLUME

GLOBAL

GOTO

GROUP

GROUPBOX

HALF

HALT

HANGUP

HARDFLOW

HARDWARE

HEADER

HEATH19

HELP

HELPFILE

HIDDEN

HOLD

HOST

HOSTDIR

HOSTPRINT

HOSTTYPE

HOTKEYS

HOTSPOT

HSCROLL

HTML

I0

I1

I2

I3

I4

I5

I6

I7

I8

I9

IBM3101

IBM3161

IBM3270

IBMPC

ICON

ICONBUTTON

ICONFLASH

ICONPATH

IF

INBOX

INCNCTLIST

INCREMENTAL

IND$FILE

INFORMATION

INIT

INSERT

INTEGER

INTERNET

INTL

INTO

INTSLTIME

IPADDRESS

IPPORT

ISFILE

ISKEY

ISO

ITALIC

ITEM

ITEMCOUNT

ITEMCREATE

ITEMFIND

ITEMNAME

ITEMREMOVE

ITOA

KEEP

KERMIT

KERMSERVE

KEYBOARDFILE

KEYFLUSH

KEYGET

KEYPADAPP

KEYS

KEYSTATE

KEYTOANSI

KEYTOOEM

L0

L1

L2

L3

L4

L5

L6

L7

L8

L9

LANDSCAPE

LARGEABARS

LEFT

LENGTH

LF

LINEPACE

LINETYPE

LINEWRAP

LISTBOX

LMOUSE

LOCAL

LOCALDIR

LOCATE

LOCATION

LOGGING

LOGOFF

LOGONNAME

LOGONTIMEOUT

LOGOUT

LONG

LONGDISTANCE

LONGFILENAME

LONGJMP

LOOPFOR

LOOPWHILE

LRECL

LTIME

LTIMEELAPSED

LTIMEINTS

LTIMEMISC

LTIMESTRING

LTIMESTRS

LTOA

MAIL

MAKEPATH

MANAGER

MAPISEND

MARGINS

MARK

MASKED

MATCHCASE

MAXIMIZED

MAXLENGTH

MCIEXEC

MCISEND

MEMADDRESS

MEMALLOC

MEMAVAIL

MEMCHR

MEMCMP

MEMFREE

MEMGETC

MEMICMP

MEMLOAD

MEMMOVE

MEMO

MEMORY

MEMPUTC

MEMREAD

MEMREALLOC

MEMSET

MEMSIZE

MEMTOTAL

MEMWRITE

MENU

MENUBAR

MENUCHECK

MENUITEM

MENUITEMCOUNT

MENUPOPUP

MENUPOPUPID

MENUSELECT

MENUSHOW

MENUSHOWPOPUP

MENUSTATE

METAFILE

METAFILEBKG

METAKEY

METAKEYFILE

METAKEYS

METHOD

MINIMIZED

MISC

MKDIR

MODEM

MONTHSTR

MOUSECOORD

MSPAUSE

MULTILINE

MULTIPLE

MULTIPLEWINDOWS

MUSIC

MVS_TSO

NEGOTIATE

NEGOTIATION

NEVER

NEWER

NEWS

NEXT

NEXTTASK

NO

NONCONTIG

NONDEST

NONE

NOPAINT

NORMAL

NOT

NOTES

NOTESFILE

NOTESPATH

NULLSTR

NULLSUPPRESS

NUMTOSTR

OBJCOORD

OBJHIDE

OBJMOVE

OBJPAINT

OBJPOINTID

OBJREMOVE

OBJSHOW

ODD

OEMTOANSI

OEMTOKEY

OFF

OFFSET

OK

OKCANCEL

ON

ONCE

ONVERIFY

OPEN

OPTIONS

ORGANIZATION

ORIENTATION

ORIGTIME

OTHER

OUTBOX

OVERWRITE

PACECHAR

PACELINES

PACKETSIZE

PADCHAR

PADNUM

PAGENUMBERS

PAGING

PARAM

PARENT

PARITY

PASSIVEMODE

PASSWORD

PASTETEXT

PATH

PATTERN

PAUSE

PAUSECHAR

PERMANENT

PHONENUMBER

PHYSICAL

PIXELS

PKMODE

PKRECV

PKSEND

PLAYBACK

PLAYBACKPACE

POPUP

PORT

PORTRAIT

PREFIX

PRINT

PRINTALIGN

PRINTATTR

PRINTCAPTURE

PRINTCHAR

PRINTER

PRINTFIT

PRINTFONT

PRINTMARGIN

PRINTSTR

PRINTTABS

PRINTTABSTR

PROC

PROFILERD

PROFILEWR

PROGRAM

PROMPT

PROMPTHEADERS

PROTECT

PROTECTATTR

PROTOCOL

PUSHBUTTON

PUTENV

PW

PWEXIT

PWMENU

PWMODE

PWTITLEBAR

QUERY

QUESTION

QUICKOPTION

QUICKSELECT

QUIET

RADIOBUTTON

RADIOGROUP

RAND

RAW

RAWASCII

RAWPRINT

READ

READAPPEND

READWRITE

REBOOT

RECEIVE

RECEIVED

RECEIVER

RECFM

RECORDMODE

RECVBAUD

RECVCMD

RECVPRINT

RECVVIEW

RECYCLE

RELAXED

REMOTE

REMOTECMD

REMOVE

RENAME

REPAINT

REPLYADDRESS

RESET

RESTORE

RESUME

RETAINFILES

RETRIES

RETRYCANCEL

RETRYDELAY

RETURN

REVERSE

REVERSEBIT

REWIND

RGET

RGETCHAR

RIGHT

RIP

RMDIR

RMOUSE

RMVPOLLED

ROWS

RSTRCMP

RUN

RXCR

RXDATA

RXFLUSH

S0

S1

S2

S3

S4

S5

S6

S7

S8

S9

SAVE

SBBUFFER

SBPAGES

SBSAVE

SCALE

SCHEDULED

SCREEN

SCREENTOWIN

SCRIPTFILE

SCRIPTPATH

SCRIPTSTART

SCROLL

SCROLLMETHOD

SDLGFOPEN

SDLGINPUT

SDLGMSGBOX

SDLGSAVEAS

SEARCH

SECURITY

SELECT

SEND

SENDCMD

SENDER

SENDFILE

SENDKEY

SENDKEYSTR

SENDPOLLED

SENDVKEY

SENT

SEPARATOR

SERVERADDRESS

SET

SETJMP

SETPOINTER

SETUP

SHARED

SHELL

SHIFT

SHORT

SHORTPATH

SHOWFAXSTATUS

SHUTDOWN

SIERRA

SIGNATUREFILE

SIMPLE

SINGLE

SIZE

SIZEOF

SKIP

SNAPSHOT

SOFTFLOW

SOFTWARE

SORT

SPACE

SPAWN

SPLITPATH

STATCLEAR

STATIONID

STATMSG

STATUSATTR

STATUSLINE

STOP

STOPBITS

STRCAT

STRCHR

STRCMP

STRCPY

STRCSPN

STRDELETE

STREAMING

STREXTRACT

STRFIND

STRFMT

STRGETC

STRICMP

STRING

STRINSERT

STRIP

STRIPBIT8

STRLEN

STRLWR

STRNCMP

STRNICMP

STRPUTC

STRQUOTE

STRRCHR

STRREAD

STRREPLACE

STRREV

STRRIGHT

STRSEARCH

STRSET

STRSLTIME

STRSPN

STRTOCLIP

STRTOK

STRTONUM

STRUPDT

STRUPR

STRWRITE

STYLE

SUBSTR

SUCCESS

SUSPEND

SWITCH

TABEXPAND

TABKEY

TABSTOPS

TARGET

TASKACTIVATE

TASKEXISTS

TASKEXIT

TASKNAME

TASKPATH

TASKWIN

TELNET

TERMCOLORS

TERMFONT

TERMGETC

TERMGETS

TERMINAL

TERMINALID

TERMKEY

TERMMSG

TERMPUTC

TERMPUTS

TERMREADC

TERMREADS

TERMRESET

TERMVKEY

TERMWRITEC

TERMWRITES

TEXT

TIGHT

TILED

TIME

TIMEOUT

TIMESTAMP

TIMING

TOOLTIPS

TOP

TOPIC

TRANSLATE

TRANSMIT

TTY

TURNCHAR

TVI910

TVI912

TVI920

TVI922

TVI925

TVI950

TVI955

TXFLUSH

TXMETHOD

TXPACE

TYPE

UNDERLINE

UNSELECT

UNTIL

UPDATE

UPLD

UPLDPACE

UPLDPATH

UPTO

USELRECL

USEPACECHAR

USERECFM

USEREXIT

USERID

USERMSG

USERNAME

USERWIN

US_CR

UWINCREATE

UWINPAINT

UWINREMOVE

UWUS

UWUTOWIN

VARIABLE

VIDTEX

VIEWCURSOR

VIEWGIF

VIRTUAL

VISUAL

VM_CMS

VOICE

VOICENUMBER

VT100

VT102

VT220

VT320

VT52

WAITFOR

WAITQUIET

WAITUNTIL

WAVEFILE

WEEKDAYSTR

WHEN

WHENSUSPEND

WHENTARGET

WHILE

WINACTIVATE

WINCLOSE

WINCOLORS

WINCOORD

WINDOW

WINENABLED

WINEXISTS

WINFOCUS

WINHIDE

WINMAXIMIZE

WINMINIMIZE

WINMOVE

WINOWNER

WINRESTORE

WINSHOW

WINSIZE

WINSTATE

WINTASK

WINTEXT

WINTOSCREEN

WINTOUWU

WINVISIBLE

WITH

WIZARD

WORD

WRITE

WWW

WYSE100

WYSE50

WYSE60

WYSE75

XFERCANCEL

XFERMODE

XFERYIELD

XLATIN

XLATOUT

XLATSTR

XMITBAUD

XMODEM

XOFF

XWINDOW

YES

YESNO

YESNOCANCEL

YIELD

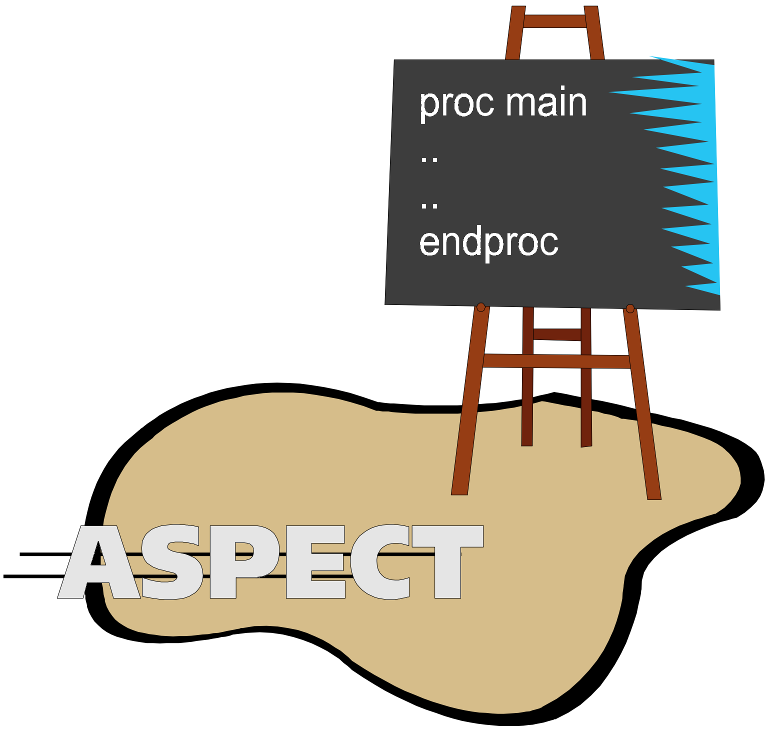
YMODEM

YMODEMG

ZMODEM

Appendix C

Virtual Key Codes



Introduction

The table below details the mnemonic constant names, hexadecimal values, and key-names for Microsoft Windows virtual key codes, in numeric order. Keys described as “Keyboard specific” may not appear on every keyboard.

Note that VK_A through VK_Z are the same as their ASCII equivalents (’A’ through ’Z’ with hexadecimal values from 0x41 thru 0x5A). VK_0 through VK_9 are the same as their ASCII equivalents (’0’ thru ’9’ with hexadecimal values from 0x30 through 0x39).

To **#define** these mnemonic key-names and values in your scripts, simply **#include** the file **vkeys.inc**, which is located in your default ASPECT directory.

Only keycodes which have an actual keyboard equivalent can be used in a key-sending command. The mouse-related keycodes, for example, cannot be sent. Likewise, you cannot use the **keyget** command to see a mouse button event. However, you can use the mouse keycodes in a command such as **keystate** to determine the state of a mouse button.

VIRTUAL KEY CODES TABLE

Mnemonic	Value	Description
VK_LBUTTON	0x01	Left mouse button
VK_RBUTTON	0x02	Right mouse button
VK_CANCEL	0x03	Ctrl-Break
VK_MBUTTON	0x04	Middle mouse button
	0x05-0x07	Undefined
VK_BACK	0x08	Backspace key
VK_TAB	0x09	Tab key
	0x0A-0x0B	Undefined
VK_CLEAR	0x0C	Clear key or Keypad 5 with Num lock off
VK_RETURN	0x0D	Enter or Return key
	0x0E	Undefined
	0x0F	Undefined
VK_SHIFT	0x10	Shift key
VK_CONTROL	0x11	Control or Ctrl key
VK_MENU	0x12	Menu or Alt key

VIRTUAL KEY CODES TABLE

Mnemonic	Value	Description
VK_PAUSE	0x13	Pause key
VK_CAPITAL	0x14	Capital or Caps Lock key
	0x15-0x19	Reserved for Kanji Systems
	0x1A	Undefined
VK_ESCAPE	0x1B	Esc or Escape key
	0x1C-0x1F	Reserved for Kanji Systems
VK_SPACE	0x20	Spacebar
VK_PRIOR	0x21	Page Up key
VK_NEXT	0x22	Page Down key
VK_END	0x23	End key
VK_HOME	0x24	Home key
VK_LEFT	0x25	Left arrow key
VK_UP	0x26	Up Arrow key
VK_RIGHT	0x27	Right Arrow key
VK_DOWN	0x28	Down Arrow key
VK_SELECT	0x29	Select key
VK_PRINT	0x2A	OEM specific
VK_EXECUTE	0x2B	Execute key
VK_SNAPSHOT	0x2C	Print Screen key
VK_INSERT	0x2D	Insert key
VK_DELETE	0x2E	Delete key
VK_HELP	0x2F	Help key
VK_0-VK_9	0x30-0x39	0 key through 9 key
	0x3A-0x40	Undefined
VK_A-VK_Z	0x41-0x5A	A key through Z key
	0x5B-0x5F	Undefined
VK_NUMPAD0	0x60	Numeric Keypad 0
VK_NUMPAD1	0x61	Numeric Keypad 1

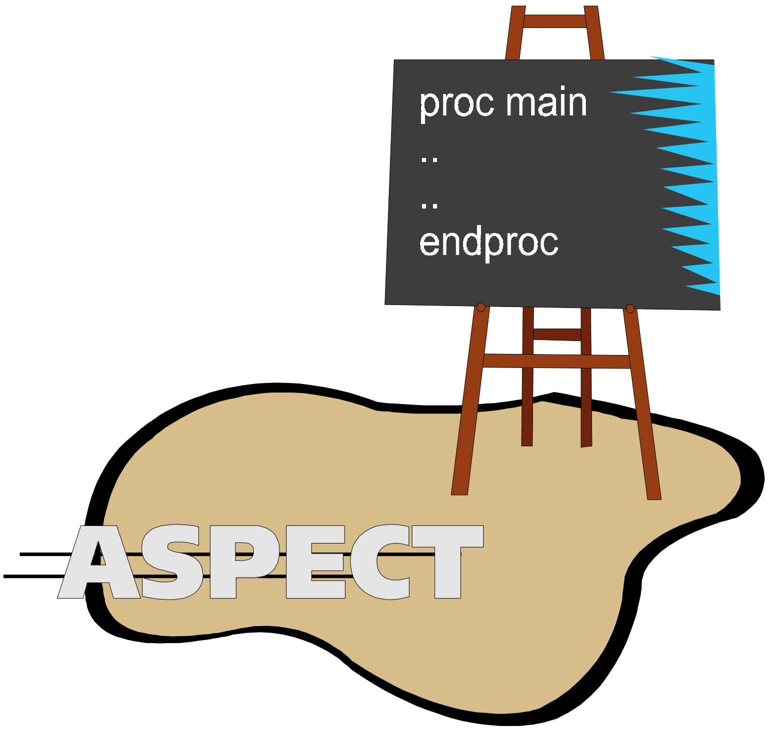
VIRTUAL KEY CODES TABLE

Mnemonic	Value	Description
VK_NUMPAD2	0x62	Numeric Keypad 2
VK_NUMPAD3	0x63	Numeric Keypad 3
VK_NUMPAD4	0x64	Numeric Keypad 4
VK_NUMPAD5	0x65	Numeric Keypad 5
VK_NUMPAD6	0x66	Numeric Keypad 6
VK_NUMPAD7	0x67	Numeric Keypad 7
VK_NUMPAD8	0x68	Numeric Keypad 8
VK_NUMPAD9	0x69	Numeric Keypad 9
VK_MULTIPLY	0x6A	Multiply key
VK_ADD	0x6B	Add key
VK_SEPARATOR	0x6C	Separator key
VK_SUBTRACT	0x6D	Subtract key
VK_DECIMAL	0x6E	Decimal key
VK_DIVIDE	0x6F	Divide key
VK_F1	0x70	Function key F1
VK_F2	0x71	Function key F2
VK_F3	0x72	Function key F3
VK_F4	0x73	Function key F4
VK_F5	0x74	Function key F5
VK_F6	0x75	Function key F6
VK_F7	0x76	Function key F7
VK_F8	0x77	Function key F8
VK_F9	0x78	Function key F9
VK_F10	0x79	Function key F10
VK_F11	0x7A	Function key F11
VK_F12	0x7B	Function key F12
VK_F13	0x7C	Function key F13
VK_F14	0x7D	Function key F14

VIRTUAL KEY CODES TABLE		
Mnemonic	Value	Description
VK_F15	0x7E	Function key F15
VK_F16	0x7F	Function key F16
VK_F17	0x80	Function key F17
VK_F18	0x81	Function key F18
VK_F19	0x82	Function key F19
VK_F20	0x83	Function key F20
VK_F21	0x84	Function key F21
VK_F22	0x85	Function key F22
VK_F23	0x86	Function key F23
VK_F24	0x87	Function key F24
	0x88-0x8F	Unassigned
VK_NUMLOCK	0x90	Num Lock key
VK_OEM_SCRO LL	0x91	Scroll Lock key
	0x92-0xB9	Unassigned
	0xBA-0xC0	Keyboard specific
	0xC1-0xDA	Unassigned
	0xDB-0xE4	Keyboard specific
	0xE5	Unassigned
	0xE6	Keyboard specific
	0xE7-0xE8	Unassigned
	0xE9-0xF5	Keyboard specific
	0xF6-0xFE	Unassigned

Appendix D

Using Remote ASPECT Commands



Introduction

Procomm Plus allows you to send ASPECT commands to a remote computer that is running Procomm Plus. This does not offer video redirection and complete control over a system like Procomm Remote. This does, however, allow you to initiate processes on your PC from a distant location, and perform a variety of tasks using ASPECT commands.

Remote commands require that both systems be using Procomm Plus, and that they be connected in *Data Terminal* mode. This remote capability will work with all terminal emulations, but to use them you must enable the ***Remote script commands*** check box in the ***System, System Options*** panel of *Setup*. If you enable the check box, you can also specify a password.

If provided, the optional password may be up to 8 characters long and the sending system's password must match it exactly (including case). If the sending system does not include the matching password, the remote command will be aborted immediately. After three such failures have occurred, Procomm Plus will automatically disable the ***Remote script commands*** check box.

Once these prerequisites are met, you can begin sending remote commands. If one of the computers is running a different version of Procomm Plus, you'll need to use specific syntax, as described in "*Backwards Compatibility*" on page 613.

Sending Remote Commands

Once you have connected to another system running Procomm Plus, you can send remote commands to the other system. The system receiving the remote commands must have the ***Remote script commands...*** check box enabled in order for the commands to be processed. If this is enabled, ASPECT commands can be sent to the system by enclosing them in the format described below in the "*Remote Command Format*" section. Upon receipt, the host computer begins processing the remote commands and then executes them. If you use remote commands often, you may wish to automate your remote commands.

Remote Command Format

To ensure that the information being sent from the other computer is not just ordinary text, but actually a remote command, remote commands are enclosed within a pair of **EOT** codes. The EOT is an ASCII 04, which is sent by pressing <Ctrl><D> or by **transmit**ting "^D" ("*Caret Translation*" on page 49 provides further details).

Remote commands are sent using this syntax:

<Ctrl><D>[password][command [<Ctrl><J> command] ...] <Ctrl><D>

where <Ctrl><D> = ASCII 04 (the EOT codes), and <Ctrl><J> (^J) = ASCII 10 (line feeds), and "**command**" is a valid ASPECT command. As you can see, a single remote command can contain multiple ASPECT commands separated by the <Ctrl><J> character. The remote command terminates with the second <Ctrl><D> character.

If you need to cancel a remote command, you can do so by sending the remote computer an ASCII 24 (<Ctrl><X>), at any time before the closing <Ctrl><D>.

Processing Remote Commands

Once the remote system has received a **^D**, it will wait up to 10 seconds between any two received characters, to allow ample time in case you're entering the commands manually. The backspace character is recognized by the receiving system to correct errors, but only to the beginning of the current line (the last line feed (**^J** or <Ctrl><J>). Note that the remote system accepts the carriage return (ASCII 13 **^M**) and interprets it as a carriage return (<Ctrl><M>) followed by a linefeed (<Ctrl><J>). Therefore, you can type an ASPECT command, press <Enter>, type another ASPECT command, press <Enter>, and so forth, until closing the remote command with a **^D**.

Once the closing **^D** has been received, the remote command is considered complete. The remote command processor automatically places the received commands between the required **proc main** and **endproc** ASPECT syntax lines and compiles it like any other ASPECT script. Once this script has been compiled, it is executed just as though you had initiated it locally. After the remote command is received, compiled, and executed, any source and target script files that were generated are automatically removed from the remote system.

Because the received ASPECT commands must be compiled, any syntax or other compile-time errors will cause the remote command to fail!

Problems can occur if a remote command is sent to the receiving system before a previously sent remote command has finished processing. To avoid this complication, consider pairing remote commands with a **set aspect rxdata on** command. This allows the receiving system to hold incoming data for processing until after the remote command has completed. This solution is effective as long as the first set of remote script commands is not retrieving incoming data itself.

Automating Remote Commands

Entering remote commands manually can be tedious, especially if you send the same commands often. One way to automate the sending of remote commands is through the use of *Meta Keys*. If you send the same commands repeatedly, configuring a *Meta Key* to send the full command makes it as easy as a single click. You may wish to read about the *Meta Key Editor* in the Procomm Plus help system.

Additionally, if you are sending a number of ASPECT commands to the remote system, you can help ensure that they will work by compiling them beforehand. Within your script editor type the ASPECT commands you wish to send as a regular script. Go ahead and include a **proc main** and **endproc**. Now, compile it to make sure it will work. Assuming it compiled fine, select all of the commands except the **proc main** and **endproc**, and copy them to the clipboard. You can then paste the clipboard's contents to the remote system and be certain you will not have a typing or syntax error. If you've written a complex script, "*Sending Complete Scripts*" on page 612 provides full details for adding conditional compilation to your scripts. Additionally, Procomm Plus includes the "*Remote Script Commander*". For more information, see page 613.

Sending Complete Scripts

Another way to send multiple ASPECT commands is to start the remote processor by pressing <Ctrl><D>, sending an ASPECT script using an ASCII file transfer, and following the transfer with the closing <Ctrl><D>.

Even though the script already has procedures in it, the remote processor still assumes it needs to add an initial **proc main** and **endproc**. To accommodate this situation, the remote command processor defines a macro named **REMOTEASP** to be automatically passed to the *ASPECT Compiler*. This allows the script's existing **proc main** and **endproc** to easily be handled with conditional compilation.

To modify an existing script for use as a remote command script, add the following lines to the top of the script:

```
#ifdef REMOTEASP       ; If compiling script as remote command
   #define MAIN MYMAIN  ; Redefine Main procedure as MYMAIN
   MYMAIN()            ; Call the renamed Main procedure
   ENDPROC             ; Terminate remote command file's PROC MAIN
```

Then, add the following lines to the end of the script, outside of any existing function or procedure:

```
#ifdef REMOTEASP       ; If compiling script as remote command
```

```
   proc REMOTEDUMMY     ; Begin definition of ‘dummy’ procedure
#endif
```

Remote Script Commander

For an easier way to send a complete script as a set of remote commands, we’ve provided the *Remote Script Commander*. This script executable, **remcmd.wax**, is installed in the **...\aspect** directory by default.

To use the *Remote Script Commander*, follow these simple steps:

1. Once you’ve established a connection with the remote computer, activate *Remote Script Commander* by starting **remcmd.wax**.
2. If the remote computer requires a password for remote commands, enter the password in the ***Password*** edit field.
3. Using the list box in the upper left corner of the **Remote Script Commander** dialog, select the ASPECT script you want the remote computer to run.
4. Click on ***Run*** to send the selected script to the remote computer. The remote computer will compile and run the script automatically as soon as it's received.

While *Remote Script Commander* greatly eases the execution of scripts on remote computers, there are two important limitations:

- The ASPECT source cannot use any **#include** statements. The *Remote Script Commander* can only transmit a single file at a time.
- If the ASPECT source contains any **chain** or **execute** commands, the script files that are referenced by those commands must already reside on the remote computer.

Backwards Compatibility

ASPECT has added numerous commands since its earlier iterations. Not all of the commands that work with Procomm Plus will work on earlier versions. For this reason, if you are sending remote commands to a machine that is running a different version of Procomm Plus, make sure that the enclosed ASPECT commands are valid for that version.

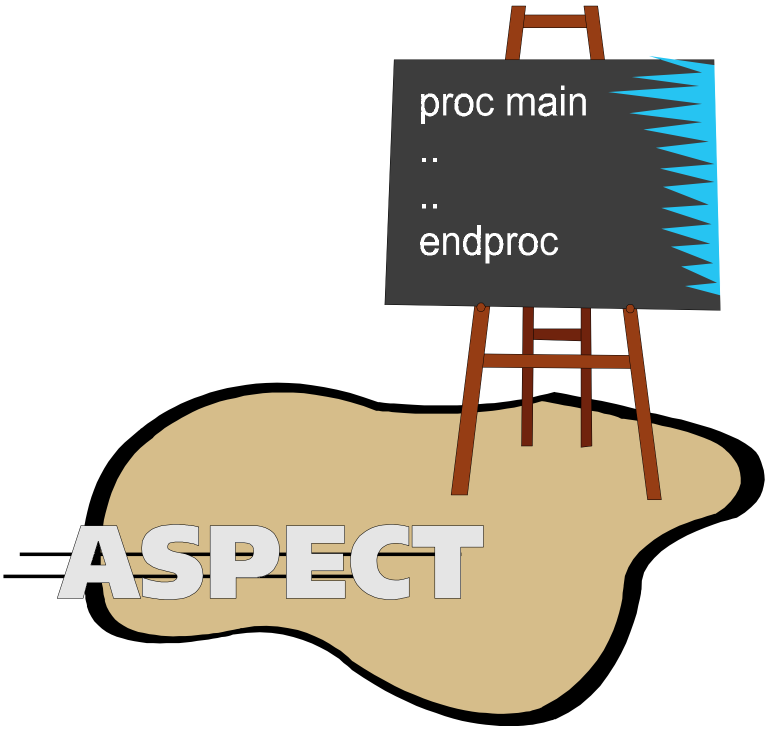
For backwards compatibility to version 1.0x of Procomm Plus for Windows, where remote commands were terminated with the receipt of the first carriage return, you can add a **pw4.ini** entry. This forces the remote command processor to revert to this less capable implementation. To use the older format of the remote commands, add the lines:

```
[Options]
RemoteCmdCR = 1
```

to the end of the **pw4.ini** file. If the ***[Options]*** section already exists in **pw4.ini**, just add the second line immediately after it. Changing the value of this entry to **0** will re-enable the current version of remote command support.

Appendix E

Using ASPTIME.DLL



Using ASPTIME.DLL

Many advanced scripts have the need for requesting a time from the user. Rather than creating your own information gathering routine, why not use the same one that Procomm Plus uses? We've taken the time and date mechanisms from Procomm Plus, and made them available to ASPECT scripts through the use of a **.dll** file - **asptime.dll**. You can access it from within your scripts using the **dllload** command followed with **dllcall** commands. The ASPTIME Dynamic Link Library may be used in your own ASPECT scripts to add custom dialog boxes for retrieving date and time information from the user.

The dllcall command uses the following syntax:

dllcall integer string [arglist]

integer	The ID of the DLL previously loaded with **dllload**.
string	A string identifying the process to be executed within the DLL.
arglist	A maximum of 12 variables can be passed as arguments.

The *integer* is returned with the issued **dllload** command. The *string* is easy enough to provide since the only process is called ASPTIME. The *arglist*, however, is somewhat lengthy since ASPTIME supports eight arguments: 3 *integer*s, 2 *long*s, and 3 *string*s.

Supported Arguments

Listed below are ASPTIME's supported arguments and their descriptions.

Argument	Description
parent	This is an *integer* value indicating the ID of the "owning" window. The ASPTIME dialog box is modal, thus its owner or parent will be disabled while the dialog box is displayed.
mode	This is a required *integer* value indicating which type of dialog to display. It can have one of the following values: 0 - Display calendar for date selection 1 - Display clock for time selection 2 - Display calendar and clock for date and time selection 3 - Display two clocks for two individual time selections 4 - Display two calendars for two individual date selections

Argument	Description
retval	This is an *integer* value returning how the user exited the ASPTIME dialog. It's value is 1 if the user clicked ***OK***, or 0 if the user clicked ***Cancel***, pressed <Esc>, or clicked the X. Use this value for information validation.
timeval1	This *long* value is required, and is set to the date and/or time value selected by the user. When two calendars or two clocks are displayed, this value corresponds to the selection in the left side dialog box control.
timeval2	This *long* value, required only when *mode* is set to 3 or 4, is set by the second (right side dialog control) date or time selected. Do not specify this value for modes 0, 1, and 2.
dlgtitle	This *string* argument specifies the title displayed in the ASPTIME dialog. While this argument is optional, but must be present if you want to supply a title for the first dialog box control.
ctrltitle1	This *string* argument specifies the title is displayed for the first control in the ASPTIME dialog. Again, this argument is optional, but must be present if you want a title for the first dialog box control.
ctrltitle2	This *string* argument, invalid when *mode* is set to 1 or 2, specifies the title that is displayed for the second control in the dialog. Like *ctrltitle1*, this argument is optional, but must be present if you want a title for the second control.

The DLLCALL will fail if any required arguments are missing or invalid. The ASPECT SUCCESS and FAILURE system variables will be set to indicate the validation of arguments.

Sample ASPECT Code

In order to help you implement the use of **asptime.dll**, we've provided the following script sample which will display a series of interactive clocks and calendars in various combinations and then return your selections.

```
proc main
   integer id, retval
   long timeval1, timeval2
   string timestr1, timestr2, timestr3
   timeval1 = timeval2 = 0   ; use current system date and time
   if dllload "ASPTIME.DLL" id
```

```
        dllcall id "ASPLTIME" $pwmainwin 0 retval timeval1 \
        "Calendar Only" " Please Select a Date: "
        ltimestrs timeval1 timestr1 timestr2
        usermsg "The date selected was: %s" timestr1
        dllcall id "ASPLTIME" $pwmainwin 1 retval timeval1 \
        "Clock Only" " Please Select a Time: "
        ltimestrs timeval1 timestr1 timestr2
        usermsg "The time selected was: %s" timestr2
        dllcall id "ASPLTIME" $pwmainwin 2 retval timeval1 \
        "Calendar and Clock" " Please Select a Date: " \
        " Please Select a Time: "
        ltimestrs timeval1 timestr1 timestr2
        usermsg "The date and time selected were:`n%s  %s" \
        timestr1 timestr2
        dllcall id "ASPLTIME" $pwmainwin 3 retval timeval1 \
        timeval2 "Clock and Clock" " Please Select a Time: " "\
        Please Select a Time: "
        ltimestrs timeval1 timestr3 timestr1
        ltimestrs timeval2 timestr3 timestr3
        usermsg "The times selected were:`n%s and %s" timestr1\
        timestr2
        dllcall id "ASPLTIME" $pwmainwin 4 retval timeval1 \
        timeval2 "Calendar and Calendar" \
        " Please Select a Date: " " Please Select a Date: "
        ltimestrs timeval1 timestr1 timestr3
        ltimestrs timeval2 timestr2 timestr3
        usermsg "The dates selected were:`n%s and %s" timestr1 \
        timestr2
      endif
    endproc
```

Index