

Intermediate ASPECT Scripting

Table of Contents

Intermediate ASPECT Scripting	1
Table of Contents	3
Introduction.....	7
About This Manual	8
Chapter 1 - General Scripting Conventions	9
ASPECT Editor	9
Parameter Passing	17
Arrays.....	18
Compiling and Debugging	19
Chapter 2 - Standard Dialogue Boxes	23
User Input	23
Message Box	24
Example of Standard Dialog Box Script	25
File Open	25
File Save.....	26
Chapter 3 - String Manipulations	28
Compare	28
Search.....	30
Replace or Insert	31
Parsing.....	32
Miscellaneous	34
Formatting.....	34
Conversions to and from Numbers	37
Chapter 4 - Dialog Boxes	38
Dialog Editor	38
Layout Assistance.....	42

Dialog Box Styles	43
Programming with Dialog Boxes.....	45
Dialog Box Events	48
Common Dialog Box Elements	50
Chapter 5 - Connection Directory	56
Set/Fetch dialentry Commands	56
Dialing Commands	58
Chapter 6 - Fax.....	60
Chapter 7 - DOS File Commands.....	65
File Existence	65
File Information Commands.....	68
File Path Commands	69
File-Moving Commands.....	70
Directory Commands	70
Miscellaneous DOS commands.....	71
Chapter 8 - File I/O Procedures.....	73
Chapter 9 - File Transfers.....	77
Chapter 10 - DDE.....	79
DDE Session	80
DDE Initiation	80
Execution of Commands.....	81
Procomm Plus as a Host	81
Procomm Plus as a Client	84
Exercise Source Codes	87
Exercise 2a.....	87
Exercise 2b.....	88
Exercise 3a.....	88
Exercise 3b.....	89
Exercise 3c.....	90
Exercise 3d.....	91
Exercise 3e.....	91
Exercise 3f.....	92

Exercise 3g	93
Exercise 4a	94
Exercise 4b	95
Exercise 4c	96
Exercise 4d	98
Exercise 5a	101
Exercise 5b	103
Exercise 6a	106
Exercise 6b	109
Exercise 7a	110
Exercise 8a	111
Exercise 9a	113
Index	115

Introduction

If you've called an on-line service, host system or BBS before, you've probably found yourself performing many repetitive tasks. For instance, each day, you may connect to a host system, log on with your ID and password, read any new electronic mail you've received, upload a data file and log off. Although this example session is quite simple, it can be tedious – why not let Procomm Plus do it for you, automatically?

An ASPECT script file is a simple ASCII file you create containing commands to be executed by Procomm Plus. In our example above, you could write a script that would call or connect to your host and perform the same functions, just as if you were entering them at the keyboard. To run the script, you'd select the script name from the Action Bar and click on the Run Script icon.

Other applications for scripts include:

- If you are working in a doctor's office that routinely transfers medical information and claims to insurance carriers, an ASPECT script can easily handle this task for you.
- You could create a script for a non-programmer that downloads a database file from the home office every morning and renames the file with today's date.
- It would be easy to write a procedure that uploads the day's sales and orders to a remote computer.
- You could connect to a computer-controlled instrument and start a capture file to log the data as it scrolls across the screen.

About This Manual

Audience

This manual is intended for a user with some experience in programming. The programming may be in ASPECT or some other language. It is important that the user have knowledge of program structure, common elements and debugging.

Commands and Syntax

We will cover a fraction of the total ASPECT commands in this manual. It is intended to give you training in these commands and also to teach you how to use the available resources. Most of the commands that are included in this manual are given with abbreviated syntax and a small amount of Help information. Our goal here is to make you aware of the types of commands that are available for various circumstances and why you would want to choose one or the other. It is expected that you will refer to the on-line Help included in Procomm Plus for specific syntax. The ASPECT Script User's Guide is another resource of information; this book is available for purchase from Symantec.

Chapter 1 - General Scripting Conventions

ASPECT Editor

The script files are text files with no special formatting rules. They could be created by any text editor but the ASPECT editor has features that make scripting easier.

Help

All of the documentation for ASPECT is available on-line. It is arranged by several groupings:

- alphabetical
- functional
- related topics

In addition, context sensitive help is accessible by right clicking on any command. The Help screen will jump right to the page for that command.

Most Help screens for a specific command have a Related Topics button at the top and a group of See Also commands listed at the bottom. This is a very good place to start looking for more information.

Automatic Indent

The spacing of a command is not critical; it doesn't need to start in any particular column. However, in structured programming, it is easier to read a program if the various levels are indented. The Editor will automatically start each line at the column the line above started; then you can use Tab or Backspace to move in or out a level. Thus, Example

1 is much easier to read than Example 2. Both are acceptable to the compiler.

```
Proc main
  string myname

  fopen 0 "name.txt" READ TEXT
  fgets 0 myname
  if myname == "Kay"
    usermsg "This is my file"
  endif
  fclose 0

endproc
```

Example 1 – Indented script source code

```
Proc main
string myname
fopen 0 "name.txt" READ TEXT
fgets 0 myname
if myname == "Kay"
usermsg "This is my file"
endif
fclose 0
endproc
```

Example 2 – Non-indented script source code

Variable Names

Variable names must start with a letter and be unique within the first 30 characters. They can contain letters, numbers or underscores but no other symbols or spaces. Variable names are not case sensitive; you may use upper case to improve readability. Thus, UserNameInAtlanta is much easier to read than usernameinatlanta but they are equivalent to the compiler.

The Reserved names are listed in the Help files under Contents | ASPECT Reserved Words. You should not use any of these names for personal variable names.

Conditional Commands

A program in any structured programming language relies heavily on special keywords that control the flow of events. While many logon scripts

are linear, scripts that are more complex 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 ASPECT, "true" means "non-zero" while "false" represents a value of zero. The expression "5 == 4" is the same as zero, or false in ASPECT, while "5 == 5" is the same as a non-zero, or true value. Be careful of the logical notation. "Year==Month" may test false but "Year=Month" is always true because you just set Year equal to Month.

Logical Operators

Symbol	Operation
<	Less Than
<=	Less Than or Equal to
>	Greater Than
>=	Greater Than or Equal to
==	Equal to
!=	Not Equal to

Simple Conditionals

The simplest type of conditional expression uses the if ... endif construct:

```

proc main
  integer Test      ; Declare variable named "Test" of integer type
  . . .            ; Some code that manipulates the 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.

Success/Failure

Many of the ASPECT commands set the Success/Failure flags depending on the outcome of their action. The commands that use this feature are marked with **SF** in a red and white circle at the upper left corner of the Help window. This information can be used in several ways in your script. The simplest way is to test to see if Success is TRUE.

```

Fopen 1 "myfile.txt" READ TEXT
If Success
  readfile ()      ;Call a procedure to read the file
else
  usermsg "Could not open myfile.txt"
endif

```

Loops

Many of the sample scripts in this tutorial include some variation of this script fragment:

```

proc main
  LOOPTEST = 1
  ;
  ;      ;Other code here.
  ;
  while LOOPTEST      ; 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 keyword while evaluates to non-zero, or true, everything between the **while** and **endwhile** executes repeatedly. 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 LOOPTEST / 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 LOOPTEST
  transmit "hi" ; This would look like hihihihihihih...
endwhile
```

Quite 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 the variable "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" ; Send a carriage return
  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 is 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 ; Declare integer variable "Test1".
  integer Test2 = 1 ; Notice that we can initialize a variable here.

  while test1 != 5 ; Outer loop.
    while test2 < 10 ; Inner loop. Notice use of less than
```

```

    test2 ++      ; Increment test2
    ;           ; More code.
    ;
endwhile        ; End of inner loop.
test1 ++        ; Increment test1

endwhile        ; End of outer loop.
endproc

```

When

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
    ; Declare 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 declare a **when** command to watch for the prompt from the remote system. We also suspend the script while we're on-line or connected using the **while ... endwhile** loop. This puts the script into an endless loop that we get out of only when we hang up. 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. Any time Procomm receives the string "Press Enter" our script file sends a carriage return.

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. When you click on an object in the dialog with an id of 0, your script calls the procedure named ProcessEvent. ProcessEvent reads and evaluates the value returned by the **dlgevent** command.

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 the 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 30 call SendSpace
```

In this example, our script will call the SendSpace procedure every 30 seconds. The SendSpace procedure is used to transmit the space character.

Finally, **when** clauses may 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 icon is selected, or when the user selects Stop Script from the 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 although the user tried to stop it!

Switch

This is a very powerful command that simplifies multiple decisions. The same results could be obtained with **if...elseif...elseif...elseif.....** commands which quickly become convoluted and hard to follow. In the following example, **Alpha** is evaluated. If **alpha** equals 0, we go to that **case** and display that user message; if **Alpha** is 1 then we display the next user message. This can continue for a large number of **cases**. The programming between **case** and **endcase** can be anything from a simple user message to calling a procedure to anything else you might want.


```
proc main
  integer Alpha = 2      ; Integer to check in switch

  switch Alpha           ; Find value of variable and display corresponding message
  case 0
    usermsg "Alpha = 0"
  endcase
  case 1
    usermsg "Alpha = 1"
  endcase
  case 2
    usermsg "Alpha = 2"
  endcase
endswitch
endproc
```

Parameter Passing

Parameters can be passed to other procedures in a number of ways, some of which are easier than others. The simplest method to implement is to use Global Variables so that no extra coding is required. Global Variables are declared before the Main procedure and then are available in all procedures.

```
string myname

proc main
  NameInput ()
  usermsg myname
endproc

proc NameInput
  termputs "Enter your name"
  termreads myname
endproc
```

In this example, because **myname** is Global, the value that is read in the **proc NameInput** is written in **proc main** with no visible passing of parameters. This is easy to write but it may be hard to maintain because any number of procedures may use and change the Global Variable. If the program isn't well documented, it is too easy to lose track of what is happening.

A more controlled method is to pass parameters by reference. Using this technique, the value of the parameters is passed to the next procedure. After the second procedure is finished, the current value of the parameters is passed back to the calling procedure.

```
proc main
  string yourname

  yourname = "Marlena Evans"
  InputName (%yourname)
  usermsg "yourname"
endproc

proc InputName
  param string yname
  termwrites "Enter your name"
  termreads yname
endproc
```

In this example, the letters that were entered for the **yname** would be passed back to **proc main** and written out in the user message for **yourname**. The parameters that are passed by reference can be given a different name in the called procedure although this can lead to confusion.

The format for calling a procedure can be in several different forms. The following have identical behaviors:

```
NameInput (%yourname)
call NameInput with %yourname
```

You can use whichever one you are most comfortable with. The first sample is preferred for most applications with the major exception being a WHEN statement which requires the second.

Arrays

Arrays are defined and used much like arrays in other programming languages. There may be up to 12 dimensions in an array. Arrays may be local or global.

```
integer StudentName[5]
integer mynumbers[3][3][3]
string ClassNames[12]
```

The array StudentName contains 5 elements which are numbered StudentName[0], StudentName[1], StudentName[2], StudentName[3] and StudentName[4]. The number in brackets in the declaration is the number of elements in the array, not the top value. Remember that string variables are always 256 characters long so arrays of string variables can get very large very quickly.

Compiling and Debugging

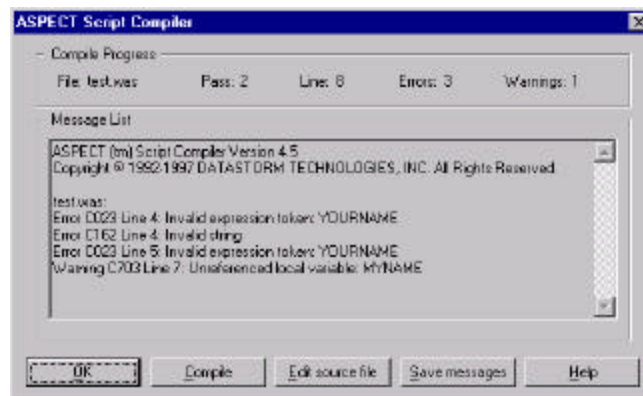
Here is an example of a program with an error:

```
proc main
  string MyName

  termreads YourName
  usermsg "Your name is %s" YourName

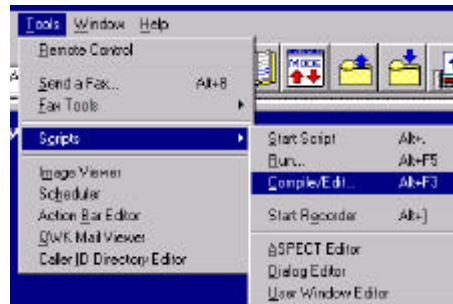
endproc
```

When we compile this, we get the following screen:

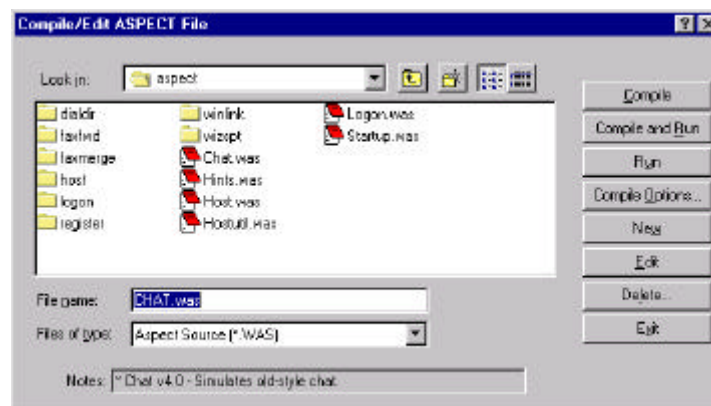


This lists the syntax and spelling errors and is relatively simple to follow to get the compile errors corrected. A general Rule of Thumb is to start at the top and correct all the errors you understand. The next time you compile, you might find that you have also fixed some of the errors that didn't understand. You might have to compile and fix several times to get all the compile errors fixed.

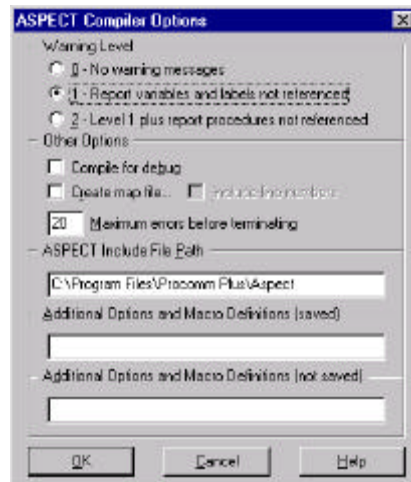
The compile errors are usually simple to locate and fix. The more difficult errors are the logic errors. We have a Debug tool that simplifies working on logic errors. To turn this on, start at Tools on the top-line menu. Choose Scripts and then Compile/Edit.



This brings up the Compile/Edit screen.



Now select the Compile Options...



Put a check in the Compile for Debug checkbox. Now you will get a Debug box when a compile error occurs or when you add a **breakpoint** command.

We can use the following script to see how debug will help us track down logic errors.

```
proc main
  integer Peanuts[5]
  integer counter = 0

  while counter != 6
    Peanuts[counter] = counter
    usermsg "counter is %i" counter
    counter ++
  endwhile
endproc
```

Executing this program without Debug gives the following error message after displaying the user message of "counter is 4".

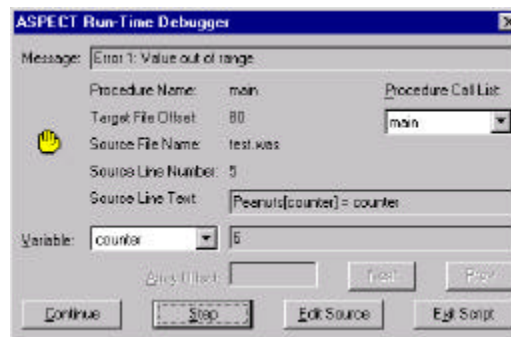


This is not terribly useful!

Executing this same program with the Debug option checked will bring up this window



Since the error we are getting is Value out of Range, we can pull up the suspected variable Counter then click on the Step button to step through the script.



Now we see that Counter goes as high as 6 and that is clearly out of bound on the array Peanuts[5]. Of course, this is much more useful and necessary on a larger script that is more complicated.

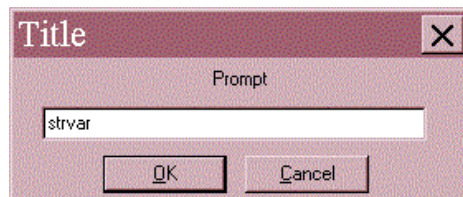
Chapter 2 - Standard Dialogue Boxes

Very few scripts that people write in ASPECT run invisibly. Most scripts require some user input and then write some kind of response to the screen. Later we will learn how to write sophisticated dialogue windows but in order to get started we need some simple techniques. ASPECT provides us with four commands for this purpose:

- **sdlginput** -- input
- **sdlgmsgbox** – message output
- **sdlglopen** -- file open
- **sdlgsaveas** – file save

User Input

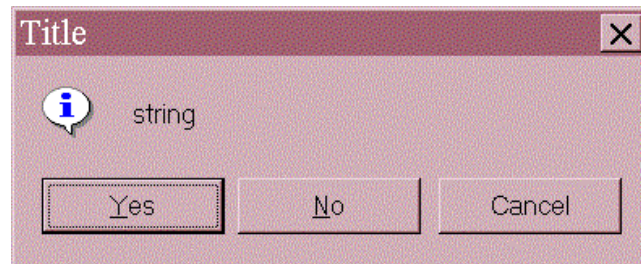
sdlginput title prompt StrVar [MASKED] [DEFAULT]



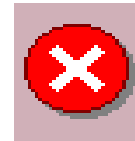
The Title and the Prompt appear on the input window. Whatever the user types will be returned in the StrVar. If the input is something personal, using the MASKED switch will make whatever the user types in appear as asterisks. If the DEFAULT switch is used, the current value of StrVar will appear in the window and the user can accept it by pressing Enter.

Message Box

sdlmsgbox title string icon button intvar [integer] [BEEP]



The Title and String appear in the window. The String may be a user prompt, a warning or a question. The choice of icons is information, exclamation, question, stop or none.



The button configuration is one or more of the following: OK, Cancel, Yes, No, Abort, Retry or Ignore. You can tell which button the user clicked on because the integer relating to the button is returned in the **intvar**. The integer declares which of the three possible buttons is the default button. You may choose to have the computer beep when the message box opens.

Example of Standard Dialog Box Script

```

proc main

    string InputString      ;what the user types in
    integer UserMsgButton   ;which button is clicked

    ;input box for the user to enter his name
    sdlginput "Names" "Please enter your name" InputString

    ;if the user clicked on the Cancel button, exit
    if Failure
        exit
    endif

    ;message box to report what the user typed
    ;INFORMATION -- a lower-case "i" will be displayed
    ;YESNOCANCEL -- 3 buttons (yes, no, cancel) will be displayed
    ;UserMsgButton -- the integer relating to which button was clicked
    sdlgmsgbox "Input Name" InputString INFORMATION YESNOCANCEL UserMsgButton

    ;take different actions depending on which button was clicked
    switch UserMsgButton
        case 2
            exit
        endcase
        case 6
            usermsg "Thank you for following instructions"
            exit
        endcase
        case 7
            usermsg "You need to practice your typing!"
            exit
        endcase
    endswitch

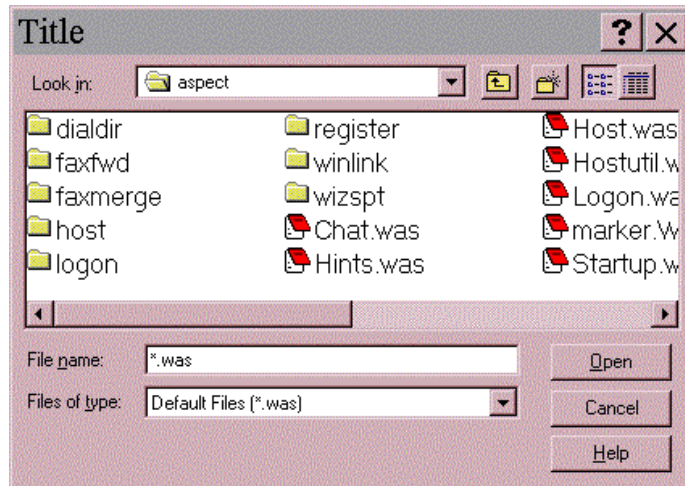
endproc

```

File Open

sdlglopen title filespec {SINGLE strvar} | {MULTIPLE filespec}

This opens a standard Windows File Open dialogue box with the Title on the title bar and displaying files as specified in FileSpec.

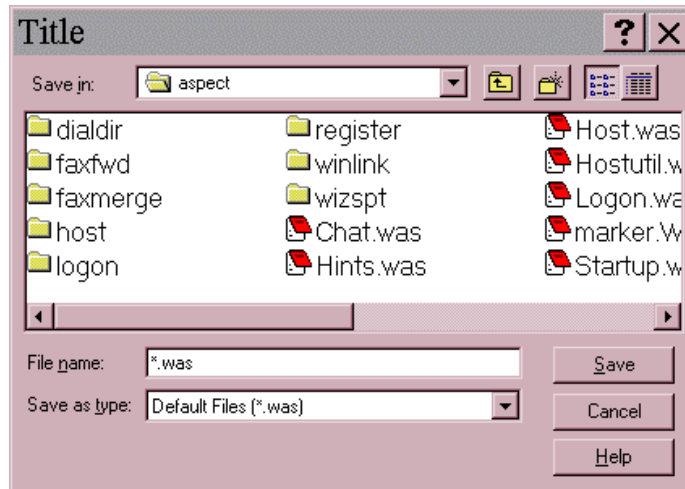


The command can specify whether the result is a single file whose name is returned in the StrVar or a group of files that are listed in the file named by the FileSpec.

File Save

sdlsaveas title filespec strvar

This command opens a standard Windows File Save As dialogue box with the Title on the title bar and displaying files as specified in FileSpec. The file name that the user chose is returned in the StrVar.



Exercise 2a



Create a script to display a standard message box with 3 buttons: yes, no and cancel. If the user presses "YES", then the second message box says "You pressed YES". If the user presses "NO", the second message box says "You pressed NO". If the user presses "CANCEL", the script exits immediately.

Exercise 2b



Create a script to display a standard input box asking for a general directory specification. Using that directory specification, open a standard file listing box and let the user choose one file. Report the name of the file that was chosen.

Chapter 3 - String Manipulations

Procomm Plus ASPECT is a powerful programming language with many uses. One of the most common uses is to create some sort of user interface that simplifies regular, repeated tasks. This requires asking for input from the user, parsing it then taking some action. Another common use is to process information from a file. Both of these tasks involve string manipulations and Procomm Plus has many commands to make this easy.

The easiest way to locate the string commands is to go to ASPECT Help | Contents | ASPECT Command Reference | Commands Listed by Function | String Commands. There are 33 commands listed there. We will go through a few of the most often used.

Compare

strcmp string1 string2 {intvar}

This command performs a case-sensitive comparison on two strings, for the length of the shorter string. 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.

Sample code to demonstrate the strcmp command:

```
proc main

    string AllowedName = "Joyce" ;Name of the one person to be allowed
    string InputName    ;user input name

    ;request the user input name
    sdlginput "Name Input" "Please enter your name" InputName
    ;compare it to the allowed person
    strcmp AllowedName InputName
    ;report success/failure to the user
    if Success
        usermsg "You have my permission"
    else
        usermsg "You are not the allowed person"
    endif

endproc
```

Related or similar commands:

- **stricmp** – case-insensitive comparison
- **strncmp** – case-sensitive comparison for a specified number of characters
- **strnicmp** – case-insensitive comparison for a specified number of characters
- **rstrcmp** -- compares the contents of two strings up to the specified length

Exercise 3a

Input 2 strings then test to see if they are identical. The output should report if they are identical or not.

Search

strfind string character intvar [matchcase]

This command searches for the first occurrence of the specified text in a given string, returning SUCCESS or FAILURE depending on the outcome. **Intvar** is 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.

Sample script using strfind, strcmp and sdlginput

Notice that **strcmp** does an exact match in the initial characters of the strings while **strfind** searches anywhere in the target string for a match.

```
proc main
    ;Name of the people to be allowed
    string AllowedNames = "Joyce, Dave, Becky, Marsha, Nancy"
    ;password
    string SecretPassWord = "Glenda"
    ;user input name and password
    string InputName
    string InputPassword

    ;request the user input name
    sdlginput "Name Input" "Please enter your name" InputName
    ;compare it to the allowed persons
    strfind AllowedNames InputName
    ;if found in the list then ask for password
    if Success
        ;user input masked because it is a password
        sdlginput "Password Input" "Please enter the password" InputPassword MASKED
        ;compare to the stored password
        strcmp SecretPassWord InputPassword
        ;report if the password matched
        if Success
            usermsg "You have permission"
        else
            usermsg "You typed in the wrong password"
        endif
    else
        ;If not in the list, then exit
        usermsg "You are not on the list of allowed persons"
    endif
endproc
```

Related or similar commands:

- **strchr** – Searches for the first occurrence of a character in a string
- **strrchr** – Searches for the last occurrence of a character in a string
- **strsearch** – searches for the number of occurrences of a string in another string

Exercise 3b



Input a list of names separated by blanks or commas. Report whether or not the list contains a "J".

Replace or Insert

strreplace strvar string1 string2 [integer] [MATCHCASE]

This command searches a string variable for a specific character or pattern of characters, replacing the target character or character pattern with another string. It will replace every occurrence unless the integer is specified.

Related or similar commands:

- **strinsert** – inserts a string into another string at a specified position.
- **strupdt** – overwrites a string with another string at a specified position
- **strdelete** – removes characters from a string
- **substr** – copies the indicated number of characters from a string, beginning at a specified position

Sample of code using *strreplace* and *\$DATE*

The object of this piece of code is to use the system date (**\$DATE**) as the name of the capture file. We can't use **\$DATE** directly because the format is MM/DD/YY and the slashes aren't allowed in a file name. So we

can use a **strreplace** command to replace each "/" with a "-" and the file name complies with naming conventions.

```
proc main
    string Filename
    FileName = $DATE
    strreplace FileName "/" "-"
    strcat FileName ".cap"
    set capture file FileName
    ...
    ...
```

Exercise 3c



Input a list of names. Check to see if there are any J's in the list and replace them with QD's. Report the resulting list.

Parsing

strextact strvar string1 string2 integer

This command extracts a string from a list of elements where string2 is the delimiter. This is useful for parsing comma-delimited files like the generic import files for the Connection Directory. The integer is used to denote which element. The numbering of the elements starts with 0.

Sample script using strextact

This sample initializes the string variable LunchMenu with the choices. This is to simplify the scripting process but not a very authentic

demonstration. We check for **nullstr** to see if we are at the end of the list. The increment (++) function is used; this is the same as

Counter = Counter + 1

that we would have seen in earlier scripting. Note that Counter is initialized to 0, not 1.

```
proc main
    ;menu list. In reality, this would be input
    string LunchMenu = "Soup,Nuts,Pie,Cake,Soda,Iced Tea"
    ;holder for each individual item
    string LunchItem
    ;counter to step through the items
    integer Counter = 0

    ;stay in the extraction loop until we have all of the items
    while 1
        ;extract each item in turn
        stextract LunchItem LunchMenu "," Counter
        ;end of list so exit the loop
        if nullstr LunchItem
            exitwhile
        endif
        ;formatted user message for each item
        usermsg "Item Number %i is %s" Counter LunchItem
        ;increment the counter
        Counter++
    endwhile
endproc
```

Related or Similar commands:

- **strtok** – extracts a string from a list of elements then removes that element from the string.

Exercise 3d



Input a comma-delimited list of names. Output the 4th name in the list.

Miscellaneous

strlen string intvar

This command returns the length of a null-terminated string into the intvar.

strcat string1 string2

This command concatenates string 2 onto string 1 and stores the results in string 1.

Formatting

strfmt strvar formatstr [arglist]

Creates a formatted string using a template, and modifies it with string or numeric variables. This is a good way to get numerical values into a string before writing it to the screen.

```
proc main
  string Address, AddrStreet
  integer AddrNumber

  AddrNumber = 1829
  AddrStreet = "N. Monroe"
  strfmt Address "My address is %i %s" AddrNumber AddrStreet
  usermsg Address
endproc
```

The user message would be "My address is 1829 N. Monroe".

Format Specifiers

- **d or i** The parameter is converted to signed decimal notation.
- **s** The string parameter is displayed up to the first null character or until the precision value is reached.
- **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; for example, **%7.3f** specifies output with 3 digits after the decimal point and a maximum of 7 total digits. 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).
- **c** The integer parameter is output as a single ASCII character.
- **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.
- **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.

Escape sequences

Another ingredient in formatting is non-printing characters, such as Carriage Return and Line Feed. These can make the output easier to read. They are accomplished with the use of the backtick, ```, in combination with a letter. The backtick is usually the key just to the left of the 1 key. All the possibilities for non-printing characters are in Help under Escape Sequences.

<code>`n</code>	New line/line feed
<code>`r</code>	Carriage return
<code>`f</code>	Page feed
<code>`"</code>	Quote

Exercise 3e



Input 2 strings. Output the strings and their lengths individually. Concatenate them and report the combined string and its length.

Conversions to and from Numbers

User input is always in character format and needs to be converted to numbers before calculating. Other times you might want to convert numbers to characters without using **strfmt**.

atoi string intvar

This command will convert the string variable to an integer and store it in intvar.

Related or Similar Commands

- **atol** – converts a string variable to a long variable
- **atof** – converts a string variable to a floating point value
- **itoa** – converts an integer variable to a character string
- **ltoa** – converts a long variable to a character string
- **ftoa** – converts a floating point variable to a character string
- **numtostr** – converts a number to a string; the numerical base (2,8,10,16,etc.) can be specified
- **strtonum** – converts a string to a number: the numerical base can be specified

Exercise 3f



Input 3 numbers. Add them together and report the sum.

Exercise 3g



Input 3 floating point numbers. Report the average to 2 decimals.

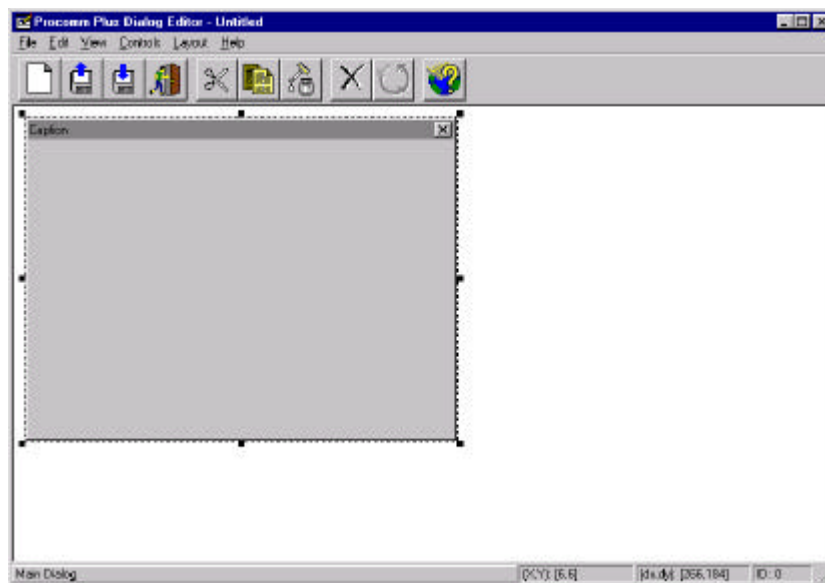
Chapter 4 - Dialog Boxes

Users are so spoiled! They expect only a graphical user interface even in a personal script. The GUI is handy for walking a user through a particular number of steps. It can also be used for displaying information in easily readable form.

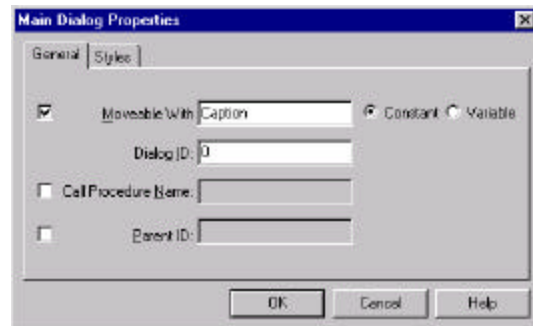
Dialog Editor

Procomm Plus provides a user-friendly dialog editor with a graphical interface. You can design your dialog boxes using simple Drag and Drop techniques. You can insert the newly designed dialog box in your script with Cut and Paste.

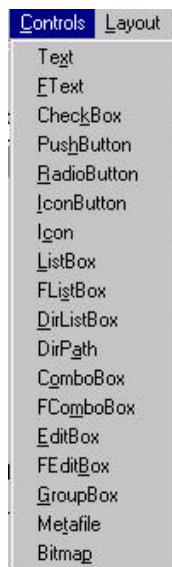
Let's start designing a dialog box and see how it progresses.



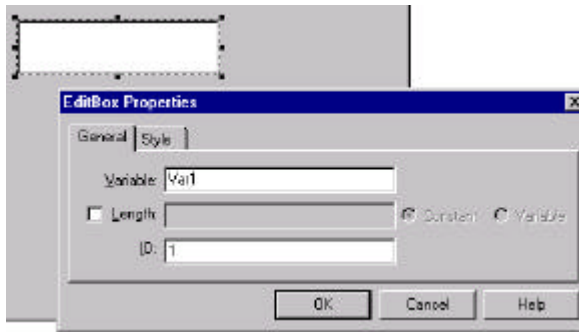
Opening the Dialog Box Editor will give you a generic window to use as a starting point. You can change the size on this to the approximate size that you want. You can also double-click on any element to bring up a Properties window.



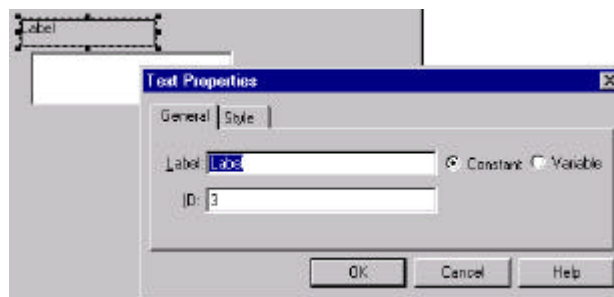
You can define the caption and make some style choices such as: center dialog box, disable close, update variables on event and others.



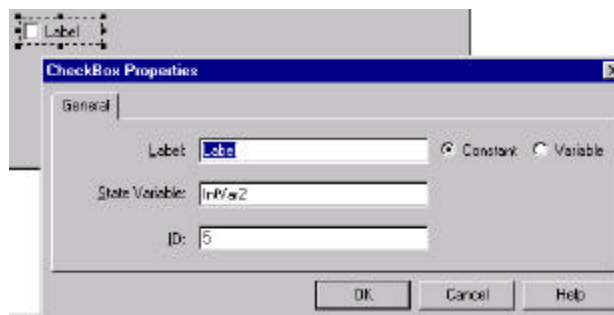
Now you are ready to add components to your window. This shows the entire list. We will choose a few of them.



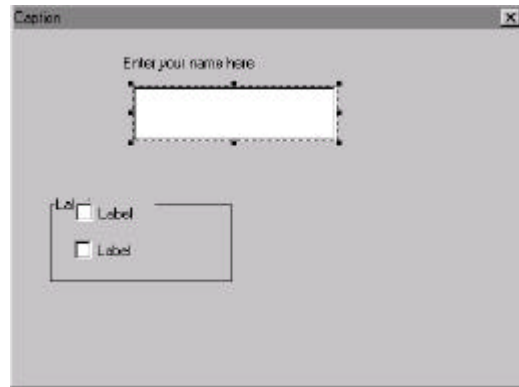
An edit box is where users will enter information. It can be either a single- or a multi-line box. It will support highlight, copy, paste and other normal Windows edit functions.



Text can be added anywhere in the dialog box.



Then you can add one or two Check Boxes. And draw a Group Box around them.



This is the total result. Now you can do Ctrl-C to copy the code and then go to the PW Editor to paste in the code. The code direct from the dialog Box Editor is below.

```
dialogbox 0 8 20 264 169 2 "Caption"
  editbox 1 62 26 104 25 Var1
  text 3 56 10 70 12 "Enter your name here" left
  checkbox 4 31 80 42 11 "Label" IntVar1
  checkbox 5 30 98 42 11 "Label" IntVar2
  groupbox 6 19 76 94 43 "Label"
enddialog
```

Note the similarities of the dialog box commands. The first number is the ID number for that element. The next 4 numbers are the XY coordinates for the upper left corner, the X-size and the Y-size. That is followed by label information and variable name, if applicable. In the editor, you can make minor adjustments to the numbers to align or space elements in a particular way. You can cut and paste in both directions between the PW editor and the dialog box editor. You can also change the generic variable names to the ones you are using in your script; this information will be carried in the cuts and pastes. Arrays are not supported in the dialog box editor so array information will have to be typed in again.

Layout Assistance



Use Control-right click to select a group of elements. Using a combination of these aids is an easy way to improve the look of your script. For example, if you have a row of push buttons along the bottom of your dialog box you want them to look well planned. You can quickly align them in a straight row, make them all the same height and evenly spaced.

Align Controls

The group can be aligned either right, left, top or bottom. Be careful using this because if you select two elements that are horizontally separated and click on left align, they will end up on top of one another!

Space Evenly

This can be used to easily arrange small multiple elements like push buttons, radio buttons or checkboxes. Elements can be even horizontally or vertically.

Make Same Size

This is an easy way to get the small elements similar. They can be the same width, height or both.

Tab Order

This means that when you press the Tab, you go to the next field. Users expect a predictable, logical order. The order that you create elements will be the default Tab Order. You can change the tab order later with this choice.

Test

Using this option, you can see the dialog box as it would appear rather than with the editing marks.

Dialog Box Styles

dialogbox id left top width height style [title]

The Dialog Box command is similar to other dialog commands because it starts with an **id** number followed by **left, top, width and height**. The **id** must be unique if you are going to use it to identify the window. The title is used only if the style requires it.

The style of the Dialog Box is determined by a sum of numbers that are discussed below. For a full description of these styles, see the Help entry for dialogbox. There is a script on the supplementary diskette (DialogBoxStyles.was) that demonstrates many of these styles.

- **0** A normal modal dialog box. A modal dialog box prevents the user from accessing its owner.
- **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.
- **8** Trap Esc, Alt-F4 and Close events. This is used to prevent the user from destroying the dialog box without script intervention.
- **16** Remove Close command from system menu.
- **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.
- **64** Suspend script execution until the dialog box is 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.

Modal versus Modeless

Modal means that you can't access the parent window while this dialog box is displayed. Modeless means you can access it. Here is an situation that illustrates this concept: if you have a modal dialog box displayed in Procomm, you can't click on the Send File button on the Action Bar. If the dialog box is modeless, you can click on the button and send a file while the dialog box is displayed. There are times when you want to restrict the user's actions so you would use a modal dialog box. If the user were to remain free to do other things, you would use a modeless dialog box.

Dialog Box Style Examples

Now, let's consider some examples. Functioning dialog boxes with most of the possible combinations may be created with the script **DialogBoxStyles.was** that is on the supplementary diskette.

64 + 2 + 1 + 0 = 67

This creates a dialog box that suspends the script (64) as long as the dialog box is displayed. The dialog box is centered (1) with respect to its parent and it is movable, with a caption (2). It is modal (0) so it captures the focus and won't allow you to access its parent.

16 + 8 + 4 + 2 = 30

The script is not suspended (<64) so you will need a WHILE loop to keep the script cycling while the dialog box is displayed. The user is prevented from exiting the dialog box in unplanned ways: the exit methods are trapped (8) so nothing happens when the user tries to exit and, further, the Close is removed from the system menu (16). The dialog box is modeless (4), which means the parent is accessible. The dialog box is movable with a title bar and a caption (2).

64 + 16 + 8 + 4 + 2 = 94

This has all the characteristics of the previous example (16 + 8 + 4 + 2) and, in addition, the script is suspended while the dialog box is displayed (64).

Programming with Dialog Boxes

There are several ways that I know of to program Dialog Boxes – all of which are valid. You may know of other methods that work. You should choose one method and stick with it for all your programming. The requirements are to display a dialog box, allow the user to make choices, perhaps update the dialog box based on the user-choices and then act on the user's choices. The dialog box may be destroyed during this process or at the end of the script. Other dialog boxes may be displayed and go through the process again.

Method 1

Characteristics

- Dialog Box Style is 64 or higher so the script is suspended while the Dialog Box is displayed
- WHEN statement reacts to events in the Dialog Box
- Dialog Box and its dlgevent-processor are in different procedures
- Uses global variables to convey Dialog Box entries to the dlgevent-processor

```
string DialogBoxVariables...
```

```
proc main
when DIALOG 0 call Handler0
```

```
    DisplayDialog0 ()
endproc
```

```
proc DisplayDialog0
```

```
    dialogbox 0 x y dx dy 71 "Caption"
```

Enddialog

while 1
 dlgevent 0 Event0

```
Enddialog  
endproc
```

```
proc Handler0  
integer Event0
```

```
dlgevent 0 Event0  
  
switch Event0  
  case ...  
    ...  
  endcase  
  case ...  
    ...  
  endcase  
endswitch  
  
endproc
```

Dialog Box Events

When a user has pressed a pushbutton or entered data in an edit box, your program needs to know about it. In addition, your program needs to know which of several events occurred so that you can take the proper action. There is a common set of commands that can accomplish these goals: **when dialog**, **dlgevent** and **switch-case**.

When Dialog

when dialog id call procedure

When an event happens in the dialog with the specified **id**, the procedure that is named will be called to handle the event. This **when** can be placed at the beginning of the Proc Main or in the subprocedure that constructs the dialogue window. Once it has been executed, it will be in effect until the **when** is canceled or the script ends. **dlgevent** must be used to clear the targeted dialog's event queue, or the **when** will fire repeatedly.

dlgevent

dlgevent id intvar

This usually appears near the top of the called subprocedure. The **id** is the id of the dialogue window. The returned **intvar** is the id of the element in the dialog box where the event occurred. When **dlgevent** accesses the event, the event queue is reset. There are other parameters for this command that allow you to Flush the event queue and Save or delete the events.

Switch – Case

This combination of commands allows you to react to specific events in an easily programmed manner. The command is discussed in more detail in Chapter 1.

Event Handling Example

```
proc EventDialog0
  integer event0

  dlgevent 0 event0

  switch event0
    case 0
      ;Do this if something happened to element 0
    endcase
    case 2
      ;Do this if something happened to element 2
    endcase
  endswitch
endproc
```

Common Dialog Box Elements

In addition to the Source Codes for the Exercises that are given in Appendix A, there are some well-commented examples of dialog box

scripting included on the supplementary diskette (DialogBoxControls.was).

dialogbox and Related Commands

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

...

enddialog

This pair of commands describes the dialog box and will make it appear. Every element listed between **dialogbox** and **enddialog** will appear in this box and be related to it. The dialogbox command has many options that are explained in the Help listing. The **style** for a normal movable window with a caption is **2**. The Call and Parent modifiers are optional.

dlgdestroy

This command destroys the box.

dlgupdate id ctrlid [ctrlid]

dlgsave id ctrlid [ctrlid]

This pair of commands processes the contents of the elements in a dialog box without closing the dialog box. The **dlgupdate** command updates the contents of edit boxes, list selections, radio buttons and other elements. The **dlgsave** command saves the contents of an element into its associated variable or file. These contents would routinely be saved when the dialog box is closed.

editbox

editbox id left top width height strvar

[strlength][MASKED][MULTILINE]

feditbox id left top width height filespec [HSCROLL]

The **editbox** command has the first 5 normal parameters followed by the string variable that will hold the contents of the edit box. The string length can be specified if you are asking for input like a telephone number, social security number or password that has a certain length. If the input is masked then all that the user sees is asterisks; this is mostly used for entry of passwords. The edit box may be multi-lined so that it wraps within the box. A single-lined box may have information beyond the edge of the box and the user would have to scroll to see it.

If the contents of the edit box are greater than 256 characters, then the **feditbox** command will use a file for storage.

text and ftext

text id left top width height label LEFT | RIGHT | CENTER

ftext id left top width height filespec [{OFFSET fileoffset [LENGTH filelength]} | DYNAMIC] [HSCROLL]

These commands display text in the dialog box. It may be a variable, specific text or the contents of a file. The text may be positioned within the text box.

checkbox

checkbox id left top width height label intvar

The label is displayed to the right of the checkbox. To specify a keyboard accelerator for the checkbox, simply include an ampersand (&) in front of the desired character. Intvar is either 1 or 0 for checked or unchecked. You can specify the initial setting by defining intvar then check the value of intvar later to see if the box is checked or not.

pushbutton

pushbutton id left top width height label [OK | CANCEL] [DEFAULT]

This command places a standard pushbutton control within dialog. The user may click on this control to indicate a choice. The difference between OK and CANCEL is that for OK changes made to file-related controls will be saved to disk. 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.

radiogroup and radiobutton

radiogroup id intvar

 radiobutton id left top width height label

 radiobutton id2 left top width height label

 ...

endgroup

Radio buttons can only appear within a radio group. The radio group ID will be used to determine if a radio button event has occurred then the radio button ID will specify which radio button.

listbox and flistbox

listbox id left top width height itemlist [tabstring]
SINGLE | MULTIPLE strvar [HSCROLL] [SORT]

flistbox id left top width height filespec
[OFFSET fileoffset [LENGTH filelength]]
[tabstring] {SINGLE strvar} |
{MULTIPLE filespec} [HSCROLL] [SORT]

This pair of commands displays a drop-down list box. The main difference between them is the size of the list. The list box input is a comma-delimited string that is limited to the standard 256 characters. The file list box input is a file of unlimited size. You can specify whether one or multiple choices are allowed. The information may be sorted alphabetically. The chosen information is returned to the script in the strvar or put into a file.

Related commands

combobox and **fcombobox** are similar commands except that they have an edit box at the top of the list so the user can add things that aren't in the list.

dirlistbox and dirpath

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

dirpath id left top width height [strvar]

Together these combine to make a Standard Windows Browse box. You can specify the path and type of files to be listed then obtain information on which files were chosen by the user.

Exercise 4a

Repeat Exercise 3f but now do the input of all three numbers in the same dialogue box.

Exercise 4b

Present a lunch menu to the user consisting of at least 7 items. Let the user choose 3 items then list the choices.

Exercise 4c

Continue with Exercise 4b by asking the user to confirm the choices and giving the option to make the choices again.

Exercise 4d

Create a dialogue box with 3 radio buttons labeled left, center and right. Depending on the user's choice, display a window at the left, center or right of the screen.

Chapter 5 - Connection Directory

The Connection Directory is a powerful part of Procomm Plus so is it only natural that the Connection directory commands be an essential part of ASPECT.

Set/Fetch dialentry Commands

The set/fetch dialentry commands are used to retrieve and change the settings for specific Connection Directory entries. The set dialentry access command allows you to select the Connection Directory entry that these commands affect.

Set dialentry access OFF | {dialclass string}

...

dialsave

Specifies the Connection Directory entry by name and class that subsequent set commands affect. When Connection Directory changes are complete, the **dialsave** command must be issued to save them.

Fetch 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, fetch the intvar is assigned to 1 (DATA) and the strvar 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.

Set/fetch dialentry areacode string

This command 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.

Set/fetch dialentry company string

This 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.

Set/fetch dialentry country string

This specifies the contents of the Country edit field for the selected Connection Directory entry. Fetch allows an optional string variable which is assigned the dialing code for the current Country selection. This command is valid only for Data, Fax and Voice entries.

Set/fetch dialentry dialnumberonly OFF | ON

This specifies whether Procomm Plus should dial only the Data number for the current entry. Fetch returns a zero for OFF and a 1 for ON. This command is valid only for Data, Fax and Voice entries.

Set/fetch dialentry longdistance OFF | ON

This specifies whether Procomm Plus 32 should always dial the current entry as a long distance number. Fetch returns a zero for OFF and a 1 for ON. This command is valid only for Data, Fax and Voice entries.

Set dialentry phonenumber string

This specifies the phone number to use for the selected Data-, Fax-, or Voice-class Connection Directory entry.

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.

Set/fetch dialentry scriptfile filename | NONE

This 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.

Dialing Commands

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

or

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

These are identical commands. The more current command is **connect** because Procomm Plus can make WWW, Telnet, FTP or Fax connections. The **dial** command has been retained for backward compatibility.

This command can be used to dial a single entry or a group of entries. If CONNECTALL is specified, Procomm Plus will keep trying until it connects to all of them. If it is not specified, any successful connection will satisfy the command.

Exercise 5a



Create a dialogue window to input name, area code, fax number and company name. Create a Connection Directory entry for this

name. You should allow the user to choose to enter another name or exit the program.

Exercise 5b

Display the first 3 names (with their companies) in the Data section of the default Connection Directory. The user should be able to choose one and have it dialed. After the connection is made, the script should exit. If Procomm is unable to make a connection, the user should be given the option to exit or dial another number.

Chapter 6- Fax

The fax commands can be used to automate sending or managing faxes. It is also possible to use a combination of the fax commands and dialogue boxes to create a customized user interface that combines elements of Fax Status and Fax Manager in one compact window.

Faxsend

```
faxsend {index | string | CURRENT | FIRST string1 string2} | {DIALDIR [GROUP] string}
        [COVERSHEET filespec | NONE] [NOTES string] [timeval]
        [SINGLE | MULTIPLE filespec [BINARY]] [MEMO filespec]
```

This single command covers many possibilities for sending a fax file from a script. The first group of parameters specifies the connection and the recipient. Either the connection, recipient and fax number are required or else the Connection directory entry may be specified. The coversheet and optional Note file are next in the parameter list. The fax may be scheduled for later by giving the long integer time variable. The actual fax consists of a single or multiple fax files that may be binary or it may be a Memo fax.

An example script demonstrating the possibilities of this command is found on the supplementary disk (FaxCommand.was).

The command is long, with many options. Let's review the syntax of the Help file. The hierarchy of the symbols is

- { | } Choose one of the options between the {}
- {} | {} Chose one of the options from the first {}-group or the second {}-group

- [] [] [] Choose options from the first []-group and/or the second []-group and/or the third []-group. The choices must appear in order.

Thus, in the first grouping, which is { } | { }, we are asked to choose how and to whom to send the fax.

```
faxsend CURRENT "Dan Hughes" "1,800-345-1515" SINGLE "c:\orders\tv.fax"
```

```
faxsend DIALDIR "Kim Parrish" SINGLE "c:\orders\silver.doc"
```

```
faxsend DIALDIR GROUP "New Hosts" SINGLE "c:\orders\congrats.txt"
```

The second grouping, which is [] [] [] , covers what kind of fax to send:

```
[COVERSHEET filespec | NONE] [NOTES string] [timeval]
[SINGLE | MULTIPLE filespec [BINARY]] [MEMO filespec]
```

So, we have a choice of Coversheet with or without a Note, the scheduled time, whether the fax contains a single or multiple files or if the fax is a Memo Fax. Variations of this portion of the command are:

```
faxsend DIALDIR "Jane Rudolph-Treacy" COVERSHEET "faxosaur.cvr" NOTES
"This is a string for the Notes Field" SINGLE "c:\orders\silver.fax"
```

```
faxsend CURRENT "Dan Wheeler" "8,1,800-345-1212" COVERSHEET NONE
```

```
faxsend DIALDIR "Kathy Levin" MEMO "c:\orders\jewels.txt"
```

\$FAXSTATUS

Sending a fax is an asynchronous process that means that the fax will send independent of the script; the script will continue with the next command. If the script needs to wait for the fax to complete, it is necessary to monitor the \$FAXSTATUS with a WHILE loop.

The values for \$FAXSTATUS are:

- | | |
|-------------------|-----------------------------------|
| 0 not busy | 3 successfully sent or received |
| 1 busy, sending | 4 unsuccessfully sent or received |
| 2 busy, receiving | |

```
iTemp = 1 ;Set iTemp value to 1 (for loop.)
faxsend FIRST "SYMANTEC" "(573) 555-4321" SINGLE FileName
while iTemp <= 1 ;While faxing is going on.
    iTemp = $FAXSTATUS ;Store value in $FAXSTATUS in iTemp
    yield ;Yield processor time.
endwhile
if iTemp == 3 ;Test the value of iTemp for success
    usermsg "Fax successfully sent."
else ;or failure of fax
    errormsg "Fax transfer failed!."
endif
```

The reason for writing the WHILE-loop this way is so that we can determine if the fax was successful or not. Once \$FAXSTATUS is set to 3 (successful) or 4 (failed) within the WHILE loop and it is accessed by the script, \$FAXSTATUS is set back to zero. This is why it is important to use the parameter iTemp; it preserves the next to the last value of \$FAXSTATUS. When the fax has completed, the value of \$FAXSTATUS is zero (not busy) but we want to know if the transmission was successful or not.

Other Fax Commands

faxcancel	Cancels the current fax operation
faxlist	Enumerates the faxes displayed in the received or scheduled fax lists
faxmodem	Determines if 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
faxstatus	Queries the status of a specific fax connection
faxview	Displays a fax file

These commands are used to determine information about the faxing system. They are used in conjunction with the Set and Fetch commands that control the paths and other fax settings that are contained in the Setup Options.

Exercise 6a



Write a script that lets the user list fax files in a fax outbox directory and choose one fax to send. Then the user is queried for the name and fax number of the recipient in order to send the fax. After the fax transmission ends, report to the user whether or not it was successful.

Exercise 6b



Create a dialog box that reports the current settings for Fax Retries, Fax Retry Delay Interval, whether or not faxes are automatically printed upon receipt, and whether or not faxes are deleted after sending.

Chapter 7 - DOS File Commands

One of the most popular uses of ASPECT is to automate file uploading and downloading. An important component of this is manipulating the files at the DOS level. There are commands to check for the file's presence, determine its size and date, delete it or rename it plus many more actions.

File Existence

findfirst filespec [string]

findnext

The **findfirst** command locates a file as specified. The argument for the file search may contain wildcards. The search will default to the current directory if no path is given. The optional string determines the attributes to be used in the search. 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.

The **findnext** command locates additional files as specified in the **findfirst** command.

Sample script with FINDFIRST and System Variables

;sample for FINDFIRST and \$-variables

```

*****
;
;* GLOBAL VARIABLES
;
*****
string FileTypes      ;file specification for searching
string FileExtension  ;Extension of the found file
string FileTitle      ;name of the file
string SizeOfFile     ;size of the file
string DateOfFile     ;date of the file
string TimeOfFile     ;time stamp of the file
string AttributesOfFile ;attributes of the file

*****
;
;*
;
;* MAIN
;
;* The Main procedure DisplayDialog to display the dialog box. It also
;* initializes the WHEN statement for the events in dialog Box 0.
;*
;
;* Calls: Proc1, Proc2
;* Modifies globals: Var1
;*
;
*****
proc main

    when dialog 0 call Handler0

    DisplayDialog ()

endproc

*****
;
;*
;
;* DisplayDialog
;
;* The procedure DisplayDialog displays the dialog box with the file
;* specification for the FINDFIRST and FINDNEXT commands. It also displays
;* the results of the file that was found by these two commands.
;*
;
;* Calls: none
;* Called by: Main
;* Modifies globals: Filetypes
;*
;
*****
proc DisplayDialog

    ;Create the dialog box. This is a Style 71 box --
    ;64 = suspend script while the dialog box is displayed
    ; 4 = Modeless
    ; 2 = Movable with caption
    ; 1 = centered
    dialogbox 0 107 46 264 113 71 "FINDFIRST Command"
        ;The text explains what is to be entered into the edit
        ;box. In this case, it is to be something like "*.wax"
        text 1 15 11 34 11 "FileSpec" left
        editbox 2 63 10 102 11 FileTypes
            ;The texts 3-8 will contain the contents of the system
            ;variables that describe the file found with the
            ;findfirst command.
        text 3 22 36 92 11 FileTitle left
        text 4 22 53 92 11 FileExtension left
        text 5 22 69 92 11 AttributesOfFile left

```

```

text 6 155 36 92 11 TimeOfFile left
text 7 155 53 92 11 DateOfFile left
text 8 155 69 92 11 SizeOfFile left
;Click on this button to do a FINDFIRST on the file
;specs listed in the edit box.
pushbutton 24 17 86 54 13 "FindFirst" DEFAULT
;click this button to do a FINDNEXT on the same filespecs
pushbutton 25 97 86 54 13 "FindNext"
;click this button to exit the script
pushbutton 26 177 86 54 13 "Exit" CANCEL
enddialog

endproc

;*****
;
;*
;*
;* Handler0
;* The procedure Handler0 precesses the events from Dialog Box 0. According
;* to which button was pushed, a file is located. The information about this
;* file is retrieved from the system variables and displayed in the dialog
;* box.
;*
;*
;* Calls: none
;* Called by: Main
;* Modifies globals: FileName, FileExtension, SizeOfFile, DateOfFile,
;* TimeOfFile, AttributesOfFile
;*
;*****
proc Handler0
integer Event0 ;event in Dialog Box 0

;Which event happened in Dialog Box 0?
dlgevent 0 Event0

;make decisions based on the event
switch Event0
;Findfirst button
case 24
;Find the first file with the file specs contained
;in FileTypes
findfirst FileTypes
;Format the system variables describing the first file
;into the corresponding variables for the texts in the
;dialog box.
strfmt FileName "File Name is %s" $FNAME
strfmt FileExtension "Extension is %s" $FEXT
strfmt SizeOfFile "Size is %li bytes" $FSIZE
strfmt DateOfFile "Date is %s" $FDATE
strfmt TimeOfFile "Time is %s" $FTIME
strfmt AttributesOfFile "Attributes are %s" $FATTR
;After the variables are formatted, then update the
;texts in the dialog box.
dlgupdate 0 3 8
endcase
;FindNext button
case 25
;Find the next file with the same file specs as the
;last FINDFIRST command
findnext
;Format the system variables describing the first file
;into the corresponding variables for the texts in the
;dialog box.
strfmt FileName "File Name is %s" $FNAME
strfmt FileExtension "Extension is %s" $FEXT
strfmt SizeOfFile "Size is %li bytes" $FSIZE
strfmt DateOfFile "Date is %s" $FDATE
strfmt TimeOfFile "Time is %s" $FTIME
strfmt AttributesOfFile "Attributes are %s" $FATTR

```

```

        ;After the variables are formatted, then update the
        ;texts in the dialog box.
    dlgupdate 0 3 8
endcase
    ;exit button
case 26
    exit
endcase
endswitch
endproc

```

isfile filespec [intvar]

This command determines if a file exists. It requires a specific file name, not wildcards. It works by trying to open a file for reading. This may fail even if the file exists because another process may be using the file. The command sets Success/Failure so it is often used as follows:

```
if isfile NameFile
```

```
    .....
```

```
else
```

```
    .....
```

```
endif
```

File Information Commands

fileget filespec {ATTRIBUTE | DATE | TIME strvar} |
 {LTIME | SIZE longvar}

fileset filespec {ATTRIBUTE | DATE | TIME string} |
 {LTIME | SIZE long}

These file commands will report or set the date, time, attributes or size for a specified file.

File Path Commands

getfilename strvar filespec

getpathname strvar filespec

These commands are useful for splitting the file specification into the file name and the path name.

addfilename pathname filename

This command will add the file name to the path name and store the result in the pathname. This could be done by using **strcat** to concatenate the path name and the file name but the difference is that **addfilename** adds a backslash if needed.

splitpath filespec drive path FileName FileExtension

makepath filespec drive path FileName FileExtension

These commands are similar to the previous commands except these work with much greater detail.

fullpath strvar filespec [pathname]

shortpath filespec strvar

These commands return the fully qualified path name including the drive identification. The **fullpath** command uses long file names and the **shortpath** command uses the 8.3 aliases. The **shortpath** result may be useful on networks that don't support long file names.

File-Moving Commands

copyfile FromFileSpec ToFileSpec

copyfile works similar to the DOS COPY command. However, copyfile does not support wildcards. Script execution is suspended until the file is copied. The FileSpec should be the fully qualified path and filename in order to avoid surprises.

delfile FileSpec

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

rename FromFileSpec ToFileSpec

If no path is supplied, rename searches the User Path for the source file. Files can be renamed to a different directory on the same drive, which is equivalent to moving them from one directory to another.

Directory Commands

chdir pathname [ASPECTPATH]

This command will change to the specified path or disk. Note, this is different behavior from the DOS **chdir** command. Optionally, the new path may become the ASPECT directory.

getdir diskID strvar

This command returns the current working directory on the specified disk. DiskID is 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.

mkdir pathname

This command will create a new directory following the standard DOS rules. If no path is specified, then the new directory will be a subdirectory of the current directory.

rmdir pathname

This command will remove an empty directory using the specified path and file name. If no path is specified, a sub-directory of the current directory will be assumed.

Miscellaneous DOS commands

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

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

shell [intvar]

These commands will run an external program, execute a DOS command or display the DOS command prompt, respectively. In each case, the task id of the session is returned in the integer variable. This task id can be used in any other commands that require it.

Exercise 7a



Write a script that will find your AUTOEXEC.BAT and report its size and date. Check to see if MYAUTOEXEC.BAT exists in the ASPECT directory. If it exists, rename it to MYAUTOEXEC.OLD. Then copy AUTOEXEC.BAT to the ASPECT directory and rename it to MYAUTOEXEC.BAT.

Chapter 8 - File I/O Procedures

Files are an excellent way to communicate between programs, to save information and to transport data to other computers. Procomm Plus has several commands that make file manipulations easy.

fopen and fclose

fopen id filespec READ | WRITE | READWRITE | CREATE |
APPEND | READAPPEND [TEXT] [SHARED]
fclose id

These are the mandatory first and last commands. You must open the file, tell the system how you plan to use the file and receive an ID. The READ/WRITE options are self-explanatory. If you specify the TEXT option, then 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. ASPECT also appends a carriage return/line feed combination to a string written with the **fputs** command. The file can be shared with **child** scripts but not other programs.

fgets, fputs, fgetc and fputc

fputs id string

These are the commands for file I/O. The commands **fgets** and **fputs** will transfer a string while **fgetc** and **fputc** transfer a character. Remember that if the file was opened in TEXT mode, a CR/LF will be automatically inserted for **fputs** or stripped for **fgets**.

Sample script using fopen and fgets

This is a script to illustrate opening and reading a file. The chosen file is the PW4.INI file that is in the "Windows" directory. This can be different on each computer so use the \$WINPATH to get the path name. Then it is concatenated onto the file name. The file is opened as READ only because there is no need to write into the file at this time. The TEXT parameter is used because this is a text file and the CR/LF should be stripped off each line. The demonstration is just to read and display the first 10 lines in the file. Contrast this with the **profilerd** example below.

```

proc main

    integer Counter    ;counter for the loop
    string LineFile    ;line read from the file
    string IniFileName ;name of the file to open

    ;Get the path for the Windows directory
    IniFileName = $WINPATH
    ;add on the file name
    strcat IniFileName "\PW4.INI"
    ;open the file for read only and text processing
    fopen 0 IniFileName READ TEXT

    ;arbitrary loop to read 10 lines from the file
    for Counter = 1 upto 10
        ;read the next line from the file
        fgets 0 LineFile
        ;if we aren't at the end of file
        if SUCCESS
            ;report to the user what the line was
            usermsg "Line %i is `n`r%s" Counter LineFile
        endif
    endfor
endproc

```

Fstrfmt

fstrfmt id formatstr [arglist]

This command is similar to **strfmt** except that the output is written to the file.

profilewr filespec topic item variable

profilerd filespec topic item variable

These commands are a very easy way to store or access data in a file. The format of the file must be in the ".INI" format which is

[topic]

item=variable

item1=variable1

The difference between these and other file-access commands is that the script doesn't need to open or close the file. The script also doesn't have to read the file one line at a time and parse each line to find the value of the variable for a particular item. These commands open the file, read the file, parse the line, access the variable and close the file all in the one command. If the topic or item doesn't exist, **profilewr** will create them.

Sample script using profilerd

For this command, we don't have to open or close the file, just access it. Here we do need to know the Topic and the Item as well as whether the value is an integer or a string. If it is an integer, we can read it as an integer. Usually, we would read a file as a string then convert it to a number.

```
proc main

integer IniValue    ;value read from the INI file
string IniFileName  ;name of the file to open

    ;Get the path for the Windows directory
IniFileName = $WINPATH
    ;add on the file name
strcat IniFileName "\PW4.INI"
    ;we don't have to open the file, just read it
    ;the topic is [Settings] and the item is PW Large Icons
    ;The value is an integer so we read it as an integer
profilrd IniFileName "Settings" "PW Large Icons" IniValue

    ;report the value
usermsg "The value for PW Large Icons is %i" IniValue

endproc
```

Exercise 8a



Create a dialogue window to input 5 names, area codes, telephone numbers and company names. For each set of entries, create a comma-delimited line in a file. (A check on how well you do this exercise is to import this file into the Connection Directory.)

Chapter 9 - File Transfers

A common use of Procomm is to automate file transfers. This will simplify transferring data on a regular basis. All of the controlling parameters for a transfer protocol can be set using Set and Fetch commands.

getfile and sendfile

getfile protocol | index | string | DEFAULT [filespec[filespec]]

sendfile protocol | index | string | DEFAULT [filespec [filespec]]

These commands are identical in their syntax. The transfer protocol can be specified by name, index, string variable or DEFAULT. Depending on the protocol, the local and remote filespec may be specified.

Both **getfile** and **sendfile** are asynchronous; this means that the script will go on to the next command after the file transfer has been initiated. Usually, it is desirable to find out whether or not the file transfer was successful and then act upon that result. If the script needs to wait for the fax to complete, it is necessary to monitor the \$XFERSTATUS with a WHILE loop.

The values for \$XFERSTATUS are:

- | | | | |
|---|----------------------------|---|-------------------------------|
| 0 | not busy | 2 | successfully sent or received |
| 1 | busy, sending or receiving | 3 | transfer aborted |

```
iTemp = 1 ;Set iTemp value to 1 (for loop.)
sendfile ZMODEM "sendfile.txt"
while iTemp == 1 ;While faxing is going on.
    iTemp = $XFERSTATUS ;Store value in $XFERSTATUS in iTemp
    yield ;Yield processor time.
endwhile
if iTemp == 2 ;Test the value of iTemp for success
    usermsg "File successfully sent."
else ;or failure of file
    errormsg "File transfer failed!."
endif
```

The reason for writing the WHILE-loop this way is so that we can determine if the fax was successful or not. Once \$FAXSTATUS is set to 2 (successful) or 3 (failed) within the WHILE loop and it is accessed by the script, \$XFERSTATUS is set back to zero. This is why it is important to use the parameter iTemp; it preserves the next to the last value of \$XFERSTATUS. When the fax has completed, the value of \$XFERSTATUS is zero (not busy) but we want to know if the transmission was successful or not.

The yield command inside of the while loop releases CPU time to the transfer process. Otherwise, the computing power is tied up in the while loop and the transfer is much slower.

Exercise 9a



Connect to the Symantec BBS using the Connection Directory to make the modem connection. Go to the Top 10 list. Download the first file using Xmodem transfer protocol.

Chapter 10 - DDE

DDE – Dynamic Data Exchange -- is a means by which two Windows programs can communicate with each other. Procomm Plus' ability to use DDE to communicate with other applications increases its power and versatility. For example, a Microsoft Access macro can retrieve a list of names and fax numbers then send a sequence of commands to Procomm Plus to fax a sales information sheet to each name in the list. Conversely, a Procomm Plus script could retrieve those names and telephone numbers then send a fax to each one of them. One major requirement is that the user be familiar with the DDE commands for both Procomm Plus and Access.

To make use of DDE, one program, known as the client, initiates a DDE session with another program, known as the server. This is accomplished by sending the server application a DDE initiate message. After the session is established, the client application sends DDE commands to the server application. The client can send as many commands as desired to the server during a single DDE conversation. When the client is finished communicating with the server application, the client is required to close the link. This is accomplished by sending a DDE terminate message to the server application.

Note that the client always sends the commands to the server, not vice versa. The server can send responses to queries by the client but it cannot tell the client to perform any operations. If the server application wants to send commands to the client application, a separate DDE link will need to be established. For example, if Procomm Plus initiates a DDE link with Microsoft Access, Procomm Plus becomes the client while Microsoft Access becomes the server. Procomm Plus can tell Microsoft Access to do a variety of things, including entering data into the data base. However, Microsoft Access cannot tell Procomm Plus to dial a phone number. In order to do this, a second DDE link would have to be created where Microsoft Access is the client and Procomm Plus is the server. Each DDE link is given a different identification number, known as

the DDE channel number, so multiple DDE links between two applications can exist simultaneously. Procomm Plus can act as either client or server.

DDE Session

These steps must take place:

- Client initiates the DDE link to the Task or Window
- Client sends commands to the Host and may receive responses
- Client terminates the DDE link. It may also end the Host program.

DDE Initiation

ddeinit chanID TaskName Topic [intvar]

This command will return the identification number for the DDE channel in the long variable **chanID**. The **TaskName** is the name that the Host is known to DDE. This is in the DDE documentation for a particular application; for Procomm Plus, it is "PW4". The **Topic** is one of the topics that the Host recognizes. These are also given in the DDE documentation. Most applications will respond to a topic of "System". Procomm Plus also supports a script name as a topic. Thus, you can initiate a DDE link to Procomm Plus and run a script with one command.

If there is more than one instance of an application that is to be the Host, then we need to specify the instance. The **intvar** is the Window ID of the main window of the instance of the application. The following snippet of code illustrates how to obtain and use this information.

```
.....  
.....  
run "excel.exe"  
pause 7  
WindowID = $MAINWIN  
ddeinit chanID "Excel" "System" WindowID
```

Execution of Commands

After a DDE link has been established, the Client can send commands to the Host. Note that the syntax of the client application's DDE command will vary from application to application. For more information on how to program other clients to perform DDE, refer to that application's user guide.

Procomm Plus as a Host

Other programs or another instance of Procomm Plus can initiate a DDE link with Procomm Plus as a Host. There are a limited number of commands that can be transmitted directly to the Host. This number is expanded greatly by the command ASPECTCMD; almost any valid ASPECT command can be transmitted.

ASPECTCMD string – executes an ASPECT command. This can be used to perform almost all of the available ASPECT tasks and settings.

Capture ON/OFF – toggles data capture on/off

DIAL dialclass [GROUP] name – dials or connects the specified Connection Directory entry

Dialload filespec -- loads the specified Connection Directory

Execute filespec -- Executes a specific script file

Getfile protocol filespec -- begins a file download with the specified transfer protocol.

Halt -- halts the currently executing script

Hangup -- disconnects the active data line

Pwexit -- exits the Host session of Procomm Plus

Sendfile protocol filespec -- begins a file upload with the specified transfer protocol.

Transmit string -- sends the specified string out the active port.

Polling commands

Procomm Plus also offers a set of internal items that can be polled by a client application. These topic command keywords are:

TOPICS -- Returns "System" and the name of the currently executing script file, if any. The items are returned in a string, each separated by a tab character.

SYSITEMS -- Lists the available items for the topic and a list of ASPECT predefined variables that can be requested. The items are returned in a string, each separated by a tab character.

FORMATS -- Displays the Windows Clipboard formats available for DDE exchange. Only the text format is supported by Procomm Plus.

STATUS -- Returns a block of information in a string, with each item separated by a comma:

DIAL DIR {directory name},
{script file | NO SCRIPT},
OFFLINE | ONLINE,
CAPTURE ON | OFF,
PRINTER ON | PRINTER OFF,
{TERMINAL | {FILE XFER} | DIALING},
FAX IDLE | FAX BUSY

We've placed the items on separate lines for clarity. This status commands is the easiest method to query a Host session of Procomm Plus. Since the system variables that we would normally use are not available, this provides a way of obtaining information about the current activities of the Host session

HELP -- Displays brief help text describing Procomm Plus' DDE server support.

ASPECT examples:

```
ddeexecute l0 "ASPECTCMD usermsg "Hello"" ;Note how quotes are used.  
ddeexecute l0 "capture ON"  
ddeexecute l0 "DIAL WWW SYMANTEC" ;Dialing entry name is case sensitive!!!  
ddeexecute l0 "transmit "ATDT 4017^M""  
ddeexecute l0 "PWEXIT"  
ddeexecute l0 "status" strvar
```

Advise

When Procomm Plus is the Host, a DDE ADVISE command can be used to create a "hot link" which monitors the value of a predefined ASPECT variable. These variables consist of integers (i0-i9), strings (s0-s9), floats (f0-f9) and longs (l0-l9).

Procomm Plus as a Client

The following commands are available in ASPECT when you are using Procomm Plus as the DDE Client

ddeinit

ddeterminate

ddeadvise

ddeunadvise

ddepoke

dderequest

ddeexecute

The following examples illustrate the use of these DDE Client commands. The first script is starting a second instance of Procomm Plus (Host). The Host session of Procomm Plus is told to run a script to download a file then notify the Client session with the name of the downloaded file.

Client script

```
string FileDown           ;name of the downloaded file

proc main

string pathname           ;path to PW4.EXE
integer pwhwnd            ;window ID of the Host session
long pwchanid            ;DDE channel ID

pathname = $pwtaskpath      ;Get PW's path.
addfilename pathname "PW4.EXE" ;Add Procomm's EXE.
```

```

run pathname          ;start up Host session of PW4
pause 4               ;wait for PW4 to start and get focus

pwhwnd = $MAINWIN     ; retrieve the window ID

                    ;initiate the DDE link and start the
                    ;downloading script
ddeinit pwchanid "pw4" "ddednld.wax" pwhwnd
if success             ;confirm that the link was established
    usermsg "ddeinit ok"
else
    usermsg "ddeinit not ok"
endif

ddeadvise pwchanid "s0" FileDown ;tell the DDE link to notify the
                                ;Client when s0 changes
when $ddeadvise call reporter ;when s0 changes, call the procedure
                                ;to report the change

while 1                ;keep looping while we are waiting
    yield              ; for s0 to change
endwhile

endproc

proc reporter          ;called procedure when s0 changes

    usermsg "success %s" FileDown ;report the name of the downloaded file
    ddeexecute pwchanid "pwexit" ;exit the Host session of PW
    exit               ;exit the Client session of PW

endproc

```

Host Script

```

...
...
    log on to the remote system and choose the file to be downloaded
...
...
getfile ZMODEM "*.txt" ;start the download process
pause 5                ; wait to give it time to start
s0 = $XFERFILE          ;set s0 to the name of the fill being transferred
while $XFERSTATUS       ; pause the script while the file downloads
    yield
endwhile

....
.....
    log off remote system
...
....

```

Exercise 10a



One instance of Procomm will be the Client. It will run a script that opens a second instance of Procomm and displays a user message in the second instance.

Exercise Source Codes

These ASPECT scripts are intended only as a sample of ASPECT programming. Symantec makes no warranty of any kind, express or implied, including without limitation, any warranties of commercial practicality and/or fitness for a particular purpose. Use of these programs is at your own risk.

Exercise 2a



Create a script to display a standard message box with 3 buttons: yes, no and cancel. If the user presses "YES", then the second message box says "You pressed YES". If the user presses "NO", the second message box says "You pressed NO". If the user presses "CANCEL", the script exits immediately.

```

;*Exercise2a.was Sample script
;*****
;
;
;*
;* MAIN
;
;* The Main procedure displays a standard message box with 3 buttons: yes,
;* no and cancel. If the user presses "YES", then the second message box
;* says "You pressed YES". If the user presses "NO", the second message
;* box says "You pressed NO". If the user presses "CANCEL", the script
;* exits immediately.
;*
;*
;* Calls: none
;*
;* Modifies globals: none
;*
;*****
;
proc main
    ;variables for the results of the message boxes
    integer result1, result2
    ;first message box
    sdlmsgbox "Exercise2a" "Press a button" QUESTION YESNOCANCEL result1
    ;actions based on the results of which button was pressed
    switch result1
        ;YES button
        case 6
            sdlmsgbox "title" "You pressed YES" INFORMATION ok result2
        endcase
        ;NO button
        case 7
    
```

```

        sdlmsgbox "title" "You pressed NO" STOP OK result2
    endcase
        ;CANCEL button
    case 2
        exit
    endcase
endswitch

endproc

```

Exercise 2b



Create a script to display a standard input box asking for a general directory specification. Using that directory specification, open a standard file listing box and let the user choose one file. Report the name of the file that was chosen.

```

;Exercise2b.was Sample script
;*****
;
;*
;
;* MAIN
;
;* The Main procedure displays a standard input box asking for a general
;* directory specification. Using that directory, it opens a standard
;* file listing box and lets the user choose one file. It reports the
;* name of the file that was chosen..
;*
;*
;* Calls: none
;* Modifies globals: none
;*
;*****
proc main
    ;directory specification
    string dirspeg
        ;file chosen by user
    string specfile
        ;result of button press in message box
    integer result1
        ;query user for the directory specification
    sdlginput "title" "Please enter a directory specification" dirspeg
        ;display selected directory and chose a file
    sdlgfopen "title" dirspeg SINGLE specfile
        ;display chosen file name
    sdlmsgbox "title" specfile INFORMATION OK result1

endproc

```

Exercise 3a



Input 2 strings then test to see if they are identical. The output should report if they are identical or not.

```

;*Exercise3A.was Sample script
;*****
;
;*
;
;* MAIN
;
;* The Main procedure performs the input, comparison and output
;
;*
;* Calls: none
;
;* Modifies globals: none
;*
;*****
;
proc main
    ;strings for the input variables
    string InputLine1, InputLine2
    ;integer to report the result of the comparison
    integer ComparisonResult
    ;integer to report which button was pressed in dialog box
    integer MsgBoxButton

    ;Request the user to input 2 strings
    sdlginput "First Input" "Please type in your first string" InputLine1
    sdlginput "Second Input" "Please type in your second string" InputLine2
    ;compare the 2 input lines
    strcmp InputLine1 InputLine2 ComparisonResult
    ;output the results of the comparison
    if ComparisonResult == 0
        sdlgmsgbox "Comparison Results" "The 2 Lines Match" INFORMATION OK MsgBoxButton
    else
        sdlgmsgbox "Comparison Results" "The 2 Lines Don't Match" EXCLAMATION OK MsgBoxButton
    endif
endproc

```

Exercise 3b



Input a list of names. Report whether or not the list contains a "J".

```

;Exercise3b.was Sample Script
;*****
;
;*
;
;* MAIN
;
;* The Main procedure asks the user to input a list of names. Then it
; reports whether or not the list contains a "J". *
;
;*
;* Calls: none
;
;* Modifies globals: none
;*
;*****
;
proc main
    ;variable for the names
    string ListName
    ;result button for the message box

```



```

integer msgbutton

        ;request input from the user
sdlginput "Name List" "Please input your list of names" ListName
        ;look for a "J" in the input string
strfind ListName "J"
        ;report the results of the string search
if SUCCESS
    sdlgmsgbox "J Report" "I found a J" EXCLAMATION OK MsgButton
else
    sdlgmsgbox "J Report" "I didn't find a J" STOP OK MsgButton
endif

endproc

```

Exercise 3c



Input a list of names. Check to see if there are any J's in the list and replace them with QD's. Report the resulting list.

```

;Exercise3c.was sample script
;*****
;
;
;
; * MAIN
; * The Main procedure asks the user to input a list of names. Then it
; * checks to see if there are any J's in the list and replace them with
; * QD's. It reports the resulting list.
; *
; *
; * Calls: none
; * Modifies globals: none
; *
; *****
proc main

        ;variable for the input name list
string ListName
        ;result button for the message box
integer MsgButton
        ;index for the location of "J" in the names list
integer FindResult

        ;request the name list
sdlginput "Name List" "Please input your list of names" ListName
        ;search the string for a J
strfind ListName "J" FindResult
        ;if a J was found, replace it
if SUCCESS
    strreplace ListName "J" "QD"
    sdlgmsgbox "New Name List" ListName INFORMATION OK MsgButton
else
    sdlgmsgbox "J Report" "I didn't find a J" STOP OK MsgButton
endif

endproc

```

Exercise 3d



Input a comma-delimited list of names. Output the 3rd name in the list

```

;*Exercise3D.was Sample Script
;*****
;
;*
;
;* MAIN
;
;* The Main procedure asks the user to input a comma-delimited list of
;* names. Then it outputs the 3rd name in the list
;*
;*
;* Calls: none
;*
;* Modifies globals: none
;*
;*****
proc main

    ;variable for the names list
    string ListName
    ;variable for the 3rd name
    string OneName
    ;result button for the message box
    integer MsgButton

    ;request the name list
    sdlginput "Name List" "Please input your list of names" ListName
    ;extract the 3rd name
    strextract OneName ListName "," 2
    ;if a name is found, report it
    if nullstr OneName
        sdlgmsgbox "Problem" "I didn't find a third name in the list" STOP OK MsgButton
    else
        sdlgmsgbox "Third Name" OneName INFORMATION OK MsgButton
    endif

endproc

```

Exercise 3e



Input 2 strings. Output the strings and their lengths individually. Concatenate them and report the combined string and its length.

```

;*Exercise3e.was Sample script
;*****
;
;*
;
;* MAIN
;
;* The Main procedure asks the user to input 2 strings. Then it outputs
;* the strings and their lengths individually. The script concatenates
;* them and reports the combined string and its length.
;*
;*
;* Calls: none
;*
;* Modifies globals: none
;*****

```

```

.*
;
;*****
;
proc main

    ;first input string
    string InputString1
    ;second input string
    string InputString2
    ;concatenated string for output
    string OutputString
    ;result button for message window
    integer MsgBoxButton
    ;length of the first string
    integer Len1
    ;length of the second string
    integer Len2

    ;request the first string
    sdlginput "String Input" "Please input your first string" InputString1
    ;request the second string
    sdlginput "String Input" "Please input your second string" InputString2
    ;determine the length of each string
    strlen InputString1 Len1
    strlen InputString2 Len2
    ;format the String 1 output into 2 lines
    strfmt OutputString "%s\n\rLength=%d" InputString1 Len1
    sdlgmsgbox "Input String 1" OutputString INFORMATION OK MsgBoxButton
    ;format the String 2 output into 2 lines
    strfmt OutputString "%s\n\rLength=%d" InputString2 Len2
    sdlgmsgbox "Input String 2" OutputString INFORMATION OK MsgBoxButton
    ;concatenate string 2 onto string 1
    strcat InputString1 InputString2
    ;determine the combined length
    strlen InputString1 Len1
    ;format and output the combined string
    strfmt OutputString "%s\n\rLength=%d" InputString1 Len1
    sdlgmsgbox "Combined String" OutputString INFORMATION OK MsgBoxButton

endproc

```

Exercise 3f



Input 3 numbers. Add them together and report the sum.

```

.*Exercise3f.was Sample Script
;*****
;
.*
;
.* MAIN
;
.* The Main procedure asks the user to input 3 numbers. The script adds
.* them together and reports the sum.
;
.*
;
.* Calls: none
.* Modifies globals: none
.*
;
;*****
;
proc main

    ;string variables for the 3 input numbers

```

```

string OneNumber, TwoNumber, ThreeNumber
    ;string variable for the output number
string OutputString
    ;integer variables for the 3 input numbers
integer Numb1, Numb2, Numb3
    ;integer variable for the sum
integer SumAll
    ;result index for the message box
integer MsgButton

    ;request the first number
sdlginput "String Input" "Please input your first number" OneNumber
    ;request the second string
sdlginput "String Input" "Please input your second number" TwoNumber
    ;request the third string
sdlginput "String Input" "Please input your third number" ThreeNumber
    ;convert the input strings into integers
atoi OneNumber Numb1
atoi TwoNumber Numb2
atoi ThreeNumber Numb3
    ;add the numbers together
SumAll = Numb1 + Numb2 + Numb3
    ;format output
strfmt OutputString "%d + %d + %d = %d" Numb1 Numb2 Numb3 SumAll
sdlgmsgbox "Sum 3 Numbers" OutputString INFORMATION OK MsgButton

endproc

```

Exercise 3g



Input 3 floating point numbers. Report the average to 2 decimals.

```

;*Exercise3g.was Sample script
;*****
;
;*
;
;* MAIN
;
;* The Main procedure asks the user to input 3 floating point numbers. The
;* script reports the average to 2 decimals.
;*
;*
;* Calls: none
;* Modifies globals: none
;*
;*****
;
proc main

    ;string variables for the 3 input numbers
string OneNumber, TwoNumber, ThreeNumber
    ;string variable for the output average
string OutputString
    ;float variables for the 3 input numbers
float Numb1, Numb2, Numb3
    ;float variable for the average
float Average
    ;result index for the message box
integer MsgButton

    ;request the first number
sdlginput "String Input" "Please input your first number" OneNumber

```

```

        ;request the second string
sdlginput "String Input" "Please input your second number" TwoNumber
        ;request the third string
sdlginput "String Input" "Please input your third number" ThreeNumber
        ;convert the input strings into integers
atof OneNumber Numb1
atof TwoNumber Numb2
atof ThreeNumber Numb3
        ;average the numbers
Average = (Numb1 + Numb2 + Numb3) / 3.0
        ;format output
strfmt OutputString "The average of %8.3f, %8.3f and %8.3f is %8.2f" Numb1 Numb2 Numb3 Average
sdlgmsgbox "Sum 3 Numbers" OutputString INFORMATION OK MsgBoxButton

endproc

```

Exercise 4a



Repeat Exercise 3f (Input 3 numbers. Add them together and report the sum.) but now do the input of all three numbers on the same dialogue box.

```

;*Exercise4a.was Sample Script
;*****
;
;*
;* MAIN
;
;* The Main procedure uses a dialogue box to input 3 numbers. The script
;* adds them together and reports the sum.
;* This is a Type 67 Dialogue Box--1(centered)+ 2(movable)+ 64 (Suspend
;* script execution until the dialog box is destroyed) = 67.
;*
;*
;* Calls: none
;* Modifies globals: none
;*
;*****
proc main

```

```

        ;string variables for input numbers
string OneNumber, TwoNumber, ThreeNumber
        ;string variable for output number
string OutputString
        ;integer variables for the input numbers
integer Numb1, Numb2, Numb3
        ;integer variable for the sum
integer SumAll
        ;result index for the user message
integer MsgBoxButton

        ;dialog box for input
dialogbox 0 8 20 175 124 67 "Numbers Input"
    text 1 30 20 46 11 "First Number" left
    text 2 30 47 61 11 "Second Number" left
    text 3 30 74 56 11 "Third Number" left
    editbox 4 104 20 34 11 OneNumber
    editbox 5 104 47 34 11 TwoNumber
    editbox 6 104 74 34 11 ThreeNumber
    pushbutton 7 125 98 34 14 "OK" OK

```

```

enddialog
;convert the input strings to numbers
atoi OneNumber Numb1
atoi TwoNumber Numb2
atoi ThreeNumber Numb3
;add the numbers together
SumAll = Numb1 + Numb2 + Numb3
;format output
strfmt OutputString "%d + %d + %d = %d" Numb1 Numb2 Numb3 SumAll
sdlgmsgbox "Sum 3 Numbers" OutputString INFORMATION OK MsgButton BEEP

endproc

```

Exercise 4b



Present a lunch menu to the user consisting of at least 7 items. Let the user choose 3 items then list the choices.

```

;*Exercise 4b.was Sample Script
;*****
;
;*
;
;* This script presents a lunch menu to the user consisting of at least 7
;* items. The user chooses 3 items then the script lists the choices
;*
;*****
;*****
;
;* GLOBAL VARIABLES
;*****
string MyLunch ;global string to hold lunch selections

;*****
;
;*
;* MAIN
;* The Main procedure calls LunchMenu to display the dialog box with the
;* menu items. It also initializes the WHEN command that waits for an event
;* in Dialog Box 0.
;*
;*
;* Calls: LunchMenu
;* Modifies globals: none
;*
;*****
proc main
    when dialog 0 call Event0Handler
        LunchMenu ()
endproc

;*****
;
;*
;* LunchMenu
;* The procedure LunchMenu displays the dialog box 0 with a list box
;* containing the menu items. The user can choose several items then click
;* on the OK button.
;*
;*
;* Calls: none
;* Called by: Main
;* Modifies globals: MyLunch
;*
;*****
;

```

```

proc LunchMenu
  string LunchItems
    ;list of items to be displayed
  LunchItems = "Tuna Salad on Whole Wheat,Tongue on Rye,Cobb \
  Salad,Coke,Coffee,Cheesecake,Brownie"
    ;dialog box with lunch choices
  dialogbox 0 135 60 193 124 67 "Lunch Menu"
    text 1 47 6 100 11 "Choose 3 items from the menu:" left
    listbox 2 44 36 105 54 LunchItems multiple MyLunch sort
    pushbutton 3 77 103 40 13 "OK" OK
  enddialog
endproc

;*****
;
;*
;
;* Event0Handler
;* The procedure Event0Handler processes the events in Dialog Box 0. If the user
;* has clicked on the OK button, then the menu items are separated and
;* output.
;*
;*
;* Calls: none
;* Called by: Main (WHEN command)
;* Modifies globals: none
;*
;*****
proc Event0Handler
  ;index for the event
  integer Event0
  ;individual lunch items
  string MyLunch1, MyLunch2, MyLunch3

  ;query for which event happened in dialog box 0
  dlgevent 0 Event0
  ;don't do anything except for the OK button (event 3)
  switch Event0
  case 3
    strextract MyLunch1 MyLunch "," 0
    strextract MyLunch2 MyLunch "," 1
    strextract MyLunch3 MyLunch "," 2
    ;display the user's lunch choices
    usermsg "Your Lunch Items:`n`r%s`n`r%s`n`r%s`n`r" MyLunch1 MyLunch2 MyLunch3
  endcase
  endswitch
endproc

```

Exercise 4c



Continue with Exercise 4b by asking the user to confirm the choices and giving the option to make the choices again.

```

;*Exercise4c.was Sample script
;*****
;
;*
;* This script presents a lunch menu to the user consisting of at least 7
;* items. The user chooses 3 items then the script lists the choices.
;* The user is asked to confirm the choices and given the option to make
;* the choices again.
;*
;

```

```

*****
;

*****
;
;* GLOBAL VARIABLES
;
*****
;
string MyLunch    ;global string to pass choices from the dialog window to the event handler

*****
;
;*
;
;* MAIN
;
;* The Main procedure LunchMenu to display the lunch menu. It also
;* initializes the WHEN commands to process the events in the Dialog
;* Windows.
;*
;*
;* Calls: LunchMenu
;* Modifies globals: none
;*
*****
;
proc main
    ; when statements to process the events in the dialog windows
    when dialog 0 call Event0Handler
    when dialog 1 call Event1Handler
    ;call the dialog window that displays the lunch menu
    LunchMenu ()
endproc

*****
;
;*
;
;* LunchMenu
;
;* The procedure LunchMenu displays the Dialog Window with the lunch items.
;* The user can choose 3 items then click on OK.
;*
;*
;* Calls: none
;* Called by: Main
;* Modifies globals: none
;*
*****
;
proc LunchMenu
    ;string to hold the total lunch items
    string LunchItems
    LunchItems = "Tuna Salad on Whole Wheat,Tongue on Rye,Cobb Salad,Coke,Coffee,Cheesecake,Brownie"
    ;blank the choices list
    MyLunch = ""
    ;display the lunch items and let user choose
    dialogbox 0 135 60 193 124 67 "Lunch Menu"
        text 1 47 6 100 11 "Choose 3 items from the menu:" left
        listbox 2 44 36 105 54 LunchItems multiple MyLunch sort
        pushbutton 3 77 103 40 13 "OK" OK
    enddialog
endproc

*****
;
;*
;
;* Event0Handler
;
;* The procedure Event0Handler processes the events in Dialog Window 0. If
;* the OK button was pressed, the lunch choices are displayed in Dialog
;* Window 1. The user is given the choice to accept these items or return
;* to Dialog Window 0.
;*
;*
;* Calls: none
;* Called by: Main
;* Modifies globals: none
;*
*****
;
proc Event0Handler
    ;integer to identify which event happened in the dialog window
    integer Event0
    ;strings to hold each lunch choice
    string MyLunch1, MyLunch2, MyLunch3

```



```

        ;find out which event happened
    dlgevent 0 Event0
        ;do something only if the OK button is clicked
    switch Event0
    case 3
        ;separate the choices into individual variables
        strextract MyLunch1 MyLunch "," 0
        strextract MyLunch2 MyLunch "," 1
        strextract MyLunch3 MyLunch "," 2
        ;display the choices for the user to accept
        dialogbox 1 158 35 184 122 67 "Lunch Choices"
            text 2 26 9 86 11 "You have chosen:" left
            text 3 26 29 115 11 MyLunch1 left
            text 4 26 48 115 11 MyLunch2 left
            text 5 26 67 115 11 MyLunch3 left
            pushbutton 6 26 99 40 13 "Cancel"
            pushbutton 7 105 99 40 13 "OK" OK DEFAULT
        enddialog
    endcase
endswitch
endproc

;*****
;
;
;
; * Event1Handler
; * The procedure Event1Handler processes the events from Dialog Window 1. It
; * gives the user the option to accept the current lunch items or go back to
; * choose something different.
;
; *
; * Calls: LunchMenu
; * Called by: Main
; * Modifies globals: none
;
;
; *****
proc Event1Handler
    ;integer to identify which event happened in the dialog window
    integer Event1
        ;find out which event happened
    dlgevent 1 Event1
        ;Take action only if OK or Cancel was clicked
    switch Event1
        ;cancel -- choose again
    case 6
        LunchMenu ()
    endcase
        ;OK -- phone in the order
    case 7
        usermsg "Your order has been phoned to Pizza Hut`n`rlt will be delivered soon"
        exit
    endcase
endswitch
endproc

```

Exercise 4d



Create a dialogue box with 3 radio buttons: left, center and right. Depending on the user's choice, display a window at the left, center or right of the screen.

```

;*Exercise4d.was Sample script
;*****
;
;*
;
;* This script creates a dialogue box with 3 radio buttons: left, center
;* and right. Depending on the user's choice, the script displays a window
;* at the left, center or right of the screen.
;*
;*****

;*****
;
;* GLOBAL VARIABLES
;*****
integer Radio1 ;integer for which radio button is clicked

;*****
;
;* MAIN
; The Main procedure calls ThreeRadioButtons to display the dialog window
; with 3 radio buttons.
;*
;* Calls: ThreeRadioButtons
;* Modifies globals: none
;*
;*****
proc main
    ; when statement to process the event in the dialog window
    when dialog 0 call Event0Handler
    ;call the dialog window to display the choices
    ThreeRadioButtons ()
endproc

;*****
;
;*
;
;* ThreeRadioButtons
;* The procedure ThreeRadioButtons displays Dialog Box 0 which contains
;* the three radio buttons for the user to select the subsequent display.
;*
;* Calls: none
;* Called by: Main
;* Modifies globals: Radio1
;*
;*****
proc ThreeRadioButtons
    ;display the choices
    dialogbox 0 172 68 169 126 67 "Three Radio Buttons"
    radiogroup 1 Radio1
    radiobutton 2 22 34 42 11 "Left"
    radiobutton 3 22 50 42 11 "Middle"
    radiobutton 4 22 66 42 11 "Right"
    endgroup
    groupbox 7 11 18 72 66 "Choose one:"
    pushbutton 5 19 103 40 13 "OK" OK
    pushbutton 6 90 103 40 13 "Exit" CANCEL
    enddialog
endproc

;*****
;
;*
;
;* Event0Handler
;* The procedure Event1Handler processes events in Dialog Box 0. If the
;* EXIT button was pressed, the script exits. If the OK button was pressed,
;* then check the radio buttons to see which one was pressed then call the
;* appropriate procedure to display the requested window.
;*
;*
;* Calls: LeftWindow, MiddleWindow,RightWindow
;* Called by: Main (WHEN statement)

```

```

.* Modifies globals: none
.*
.*
.*****
proc Event0Handler
    ;integer to identify which event happened in the dialog window
    integer Event0
    ;find out which event happened
    dlgevent 0 Event0
    ;do something only if the OK or Exit button is clicked
    switch Event0
        ;Exit button so exit the script
        case 6
            exit
        endcase
        ;OK -- find out which radio button is clicked
        case 5
            switch Radio1
                case 2
                    LeftWindow ()
                endcase
                case 3
                    MiddleWindow ()
                endcase
                case 4
                    RightWindow ()
                endcase
            endswitch
        endcase
    endswitch
endproc

.*
.*
.*
.* LeftWindow
.* The procedure LeftWindow displays a window on the left side of the screen.
.*
.*
.* Calls: ThreeRadioButtons
.* Called by: Event0Handler
.* Modifies globals: none
.*
.*
.*****
proc LeftWindow
    ;display the window on the left
    dialogbox 2 10 20 122 64 70 "Left Window"
    text 1 18 5 82 11 "This window is on the left" left
    pushbutton 2 37 32 40 13 "OK" OK DEFAULT
    enddialog
    ;go back for another choice
    ThreeRadioButtons ()
endproc

.*
.*
.*
.* MiddleWindow
.* The procedure MiddleWindow displays a window in the middle of the screen.
.*
.*
.* Calls: ThreeRadioButtons
.* Called by: Event0Handler
.* Modifies globals: none
.*
.*
.*****
proc MiddleWindow
    ;display the window in the middle
    dialogbox 2 200 20 122 64 70 "Middle Window"
    text 1 13 5 104 11 "This window is in the middle" left
    pushbutton 2 37 32 40 13 "OK" OK DEFAULT
    enddialog
    ;go back for another choice
    ThreeRadioButtons ()

```

```

endproc

.*****
;
;
;
; * RightWindow
; * The procedure RightWindow displays a window on the right side of the screen.
;
;
; * Calls: ThreeRadioButtons
; * Called by: Event0Handler
; * Modifies globals: none
;
;
;*****
proc RightWindow
    ;display the window on the right
    dialogbox 3 380 20 122 64 70 "Right Window"
        text 1 12 6 104 11 "This window is on the right" left
        pushbutton 2 37 32 40 13 "OK" OK DEFAULT
    enddialog
    ;go back for another choice
    ThreeRadioButtons ()
endproc

```

Exercise 5a



Create a dialogue window to input name, area code, fax number and company name. Create a Connection Directory entry for this name. You should allow the user to choose to enter another name or exit the program.

```

;*Exersice 5a  Sample Script
;*****
;
;
; *
;
; * Create a dialogue window to input name, area code, fax number and
; * company name. Create a Connection Directory entry for this name.
; * You should allow the user to choose to enter another name or exit
; * the program
;
;
;*****

;*****
;
; * GLOBAL VARIABLES
;
;*****
;
; * ;variables for the input connection directory entries
string NameIn, AreaIn, PhoneIn, CompanyIn

;*****
;
; *
;
; * MAIN
; * The Main procedure calls DirectoryEntry to display the dialog box which
; * allows the user to input information. It also initializes the WHEN
; * statements to process events in Dialog Boxes 0 and 1.
;
;
; * Calls: DirectoryEntry
; * Modifies globals: none
;
;

```

```

*****
;
proc main
    when dialog 0 call Event0Handler
    when dialog 1 call Event1Handler
    DirectoryEntry ()
endproc

*****
;
;
;
;* DirectoryEntry
;
;* The procedure DirectoryEntry displays...
;
;*
;
;* Calls:
;
;* Called by: Main, WhateverElse
;* Modifies globals: Var1
;
;
*****
;
proc DirectoryEntry

    ;blank out the entry edit boxes
    Nameln = ""
    Arealn = ""
    Phoneln = ""
    Companyln = ""

    ;display the dialog box
    dialogbox 0 136 68 202 125 67 "Connection Directory Entry"
        text 1 17 16 34 11 "Name" left
        text 2 17 36 34 11 "Area Code" left
        text 3 15 56 52 11 "Phone Number" left
        text 4 15 76 34 11 "Company" left
        editbox 5 79 16 102 11 Nameln
        editbox 6 79 36 102 11 Arealn
        editbox 7 79 56 102 11 Phoneln
        editbox 8 79 76 102 11 Companyln
        pushbutton 9 140 107 40 13 "OK" OK DEFAULT
    enddialog
endproc

*****
;
;
;* Event0Handler
;
;* The procedure Event0Handler processes the events from Dialog Box 0.
;* If the OK button was pressed, the contents of the edit boxes are
;* entered into the Connection Directory. Then the user is given the
;* choice of another entry or exit.
;
;*
;
;* Calls: AnotherEntry
;* Called by: Main (WHEN statement)
;* Modifies globals: none
;
;
*****
;
proc Event0Handler
    ;variable for the event in Dialog Box 0
    integer Event0
    ;query for which event occurred
    dlgevent 0 Event0
    ;only take action if the OK button was pressed
    switch Event0
        case 9
            ;create a new connection directory entry
            dialadd DATA Nameln
            set dialentry access DATA Nameln
            set dialentry areacode Arealn
            set dialentry phonenummer Phoneln
            set dialentry company Companyln
            dialsave
            ;give the user a choice of another entry or exit
            AnotherEntry ()
        endcase
    endproc

```

```

endswitch
endproc

*****
;
;*
;
;* AnotherEntry
;* The procedure AnotherEntry gives the user a choice of making another
;* entry or exiting the program.
;*
;*
;* Calls: none
;* Called by: Event0Handler
;* Modifies globals: none
;*
*****
proc AnotherEntry
dialogbox 1 187 130 151 57 67 "Another Entry?"
    pushbutton 1 9 22 54 13 "&Another Entry"
    pushbutton 2 89 22 40 13 "&Exit"
enddialog
endproc

*****
;
;*
;
;* Event1Handler
;* The procedure Event1Handler precesses the events from dialog box 1. This
;* either exits or goes back to create another connection directory entry.
;*
;*
;* Calls:
;* Called by: Main, WhateverElse
;* Modifies globals: Var1
;*
*****
proc Event1Handler
    ;variable for which event occurred
    integer Event1
    ;query for the event
    dlgevent 1 Event1
    ;switch on the event
    switch Event1
        ;another entry button
        case 1
            DirectoryEntry ()
        endcase
        ;exit button
        case 2
            exit
        endcase
    endswitch
endproc

```

Exercise 5b



Display the first 3 names (with their companies) in the Data section of the default Connection Directory. The user should be able to choose one and have it dialed. After the connection is made, the script should exit. If Procomm is unable to make a connection, the user should be given the option to exit or dial another number.

```

;*Exercise5b.was Sample Script
;*****
;
;*
;
;* This script will display the first 3 names (with their companies) in
;* the Data section of the default Connection Directory. The user is
;* able to choose one and have it dialed. After the connection is made,
;* the script will exit. If Procomm is unable to make a connection, the
;* user is given the option to exit or dial another number.
;*
;*****
;

;*****
;
;* GLOBAL VARIABLES
;*****
;
;variables for the 3 entries from the connection directory
string EntName[3], EntCompany[3]
;integer flag to indicate which entry was chosen
integer chose

;*****
;
;*
;
;* MAIN
;
;* The Main procedure calls DataDialer to display the 3 possible names to
;* call. It also initializes the WHEN statement to process the events
;* in Dialog Box 0
;*
;
;* Calls: DataDialer
;* Modifies globals: none
;*
;*****
;
proc main

    when dialog 0 call Event0Handler
    DataDialer ()

endproc

;*****
;
;*
;
;* DataDialer
;
;* The procedure DataDialer displays the dialog box with the 3 entry choices
;* for the user to choose. Before that, it counts the total number of
;* entries to be sure we don't display an empty entry. Then it fetches the
;* name and company for each entry and displays the first 3.
;*
;
;* Calls: none
;* Called by: Main
;* Modifies globals: EntName, EntCompany
;*
;*****
;
proc DataDialer
    ;count the total number of entries in the connection dir.
    integer NumEnts
    ;counter for the FOR loop
    integer Count
    ;maximum value for the Count
    integer CountMax

    ;count entries so we don't try to access more than exist
    dialcount DATA NumEnts
    ;set the CountMax to 2 or number of entries
    if NumEnts < 3
        CountMax = NumEnts - 1
    else
        CountMax = 2
    endif

```

```

        ;retrieve the information for the entries
for Count = 0 upto Countmax    ; dialname zero-based.
    dialname DATA Count EntName[Count]    ; Get name of entry.
    set dialentry access DATA EntName[Count]
    fetch dialentry company EntCompany[Count]
    set dialentry access OFF
endfor

        ;display the entries and let the user choose
dialogbox 0 6 61 452 121 67 "Connection Directory Entries"
    pushbutton 9 206 102 40 13 "Exit" OK
    text 2 15 32 101 11 EntName[0] left
    text 3 15 48 94 11 EntCompany[0] left
    text 4 160 32 94 11 EntName[1] left
    text 5 160 48 94 11 EntCompany[1] left
    text 6 325 32 94 11 EntName[2] left
    text 7 325 48 94 11 EntCompany[2] left
    pushbutton 10 14 71 62 13 "Dial this entry"
    pushbutton 11 176 73 62 13 "Dial this entry"
    pushbutton 12 339 74 62 13 "Dial this entry"
    groupbox 13 2 19 130 76 "Entry #1"
    groupbox 14 155 19 130 76 "Entry #2"
    groupbox 15 308 19 130 76 "Entry#3"
enddialog

endproc

;*****
;
;*
;
;* Event0Handler
;
;* The procedure Event0Handler processes the events from Dialog Box 0.
;* The Dialer procedure will be called with a flag set for which entry
;* is to be dialed.
;*
;*
;* Calls: Dialer
;* Called by: Main ( WHEN statement)
;* Modifies globals: chose
;*
;*****
proc Event0Handler
    ;variable for which event occurred
    integer Event0
    ;query to see which event occurred
    dlgevent 0 Event0
    ;
    switch Event0
        ;push button 10 so dial the first entry
        case 10
            dlgdestroy 0 OK
            chose = 0
            Dialer ()
        endcase
        ;push button 11 so dial the second entry
        case 11
            dlgdestroy 0 OK
            chose = 1
            Dialer ()
        endcase
        ;push button 12 so dial the third entry
        case 12
            dlgdestroy 0 OK
            chose = 2
            dialer ()
        endcase
    endswitch
endproc

;*****
;
;*
;
;* Dialer

```



```

;* The procedure Dialer dials the chosen entry. If the remote system answers,
;* the script will exit. If there is no answer, the user may choose another
;* entry.
;*
;* Calls: DataDialer
;* Called by: EventOHandler
;* Modifies globals: none
;*
;*****
;
proc Dialer

    dial DATA EntName[chose]          ; Dial the current entry.
        while $DIALING                ; Pause while dialing.
            yield
        endwhile
        pause 4                        ; Pause to wait for carrier.
        if $CARRIER
            exit
        else
            DataDialer ()
        endif
endproc

```

Exercise 6a



Write a script that lets the user list fax files in a fax outbox directory and choose one fax to send. Then the user is queried for the name and fax number of the recipient in order to send the fax. After the fax transmission ends, report to the user whether or not it was successful.

;demonstrate selecting and sending a fax v1.00

```

;*****
;
;*
;* EXERCISE6a.WAS
;* Copyright (C) 1999 Symantec Corporation
;* All rights reserved.
;*
;* This script is an example of the FAXSEND command using $FAXSTATUS to
;* determine if the fax was successful or not.
;*
;*****

;*****
;* GLOBAL VARIABLES
;*****
string FaxOutBox      ;full path for Fax OutBox
string FilesToFax     ;text file that contains the names of files to fax
string FaxRecipient   ;name of fax recipient
string FaxDial        ;Number of fax recipient

```

```

;*****
;

```

```

.*
.*
.* MAIN
.* The Main procedure fetches the current path for the fax outbox. It also
.* identifies the file to hold the names of the files to fax. Then it calls
.* ListFiles to display the possible files for the user.
.*
.*
.* Calls:ListFiles
.* Modifies globals: FaxOutBox, FilesToFax
.*
.*
.*****

proc main

;these statements direct the script when an event occurs in a dialog box
    when dialog 0 call Handler0
    when dialog 1 call Handler1

;get the current path for the fax out box
    fetch fax path outbox FaxOutBox
;define a name for the file which holds the list of files to be faxed
    FilesToFax = "Faxes.txt"

;call the dialog box to display the list of files
    ListFiles ()

endproc

.*****
.*
.*
.* ListFiles
.* The procedure ListFiles creates the dialog box which presents the list
.* of possible files to fax.
.*
.*
.* Calls: none
.* Called by: Main
.* Modifies globals: none
.*
.*
.*****

proc ListFiles

    dialogbox 0 8 20 148 169 71 "Choose File to Fax"
    text 1 36 22 68 12 "Files in Fax OutBox" left
    dirlistbox 3 40 50 68 68 FaxOutBox Multiple FilesToFax
    pushbutton 4 22 140 40 13 "OK" OK
    pushbutton 5 86 140 40 13 "Exit" CANCEL
    enddialog

endproc

.*****
.*
.*
.* Handler0
.* The procedure Handler0 processes the events from dialog box 0. If the
.* clicks on OK, then we go on to next window. If the user clicks on Exit,
.* then we exit the script.
.*
.*
.* Calls: GetRecipient
.* Called by: Main
.* Modifies globals: none
.*
.*
.*****

proc Handler0
    integer Event0 ;holds the event number for dialog box 0

;determine which event happened

```

```

dlgevent 0 Event0

switch Event0
;OK button so continue to next step
    case 4
        GetRecipient ()
    endcase
;exit button so exit script
    case 5
        exit
    endcase
endswitch

endproc

.*****
;
;*
;
;* GetRecipient
; The procedure GetRecipient displays a dialog box to obtain the
;* name and fax number. The user can then click on OK or Exit as before.
;*
;
;* Calls: none
; Called by: Handler0
;* Modifies globals: FaxRecipient, FaxDial
;*
;*****
proc GetRecipient

    dialogbox 1 8 20 230 132 71 "Recipient"
    text 1 24 31 54 11 "Fax Recipient" left
    text 2 24 72 48 11 "Fax Number" left
    editbox 3 92 31 110 12 FaxRecipient
    editbox 4 92 72 110 12 FaxDial
    pushbutton 5 52 109 40 13 "OK" OK
    pushbutton 6 147 109 40 13 "Exit" CANCEL
    enddialog

endproc

.*****
;
;*
;
;* Handler1
; The procedure Handler1 processes the events from Dialog Box 1. It checks
; to be sure the name and number were entered. Then it sends the fax.
; The scripts reports on the success or failure of the fax then deletes
; the text file which held the names of the file to fax.
;*
;
;* Calls:
; Called by: Main, WhateverElse
;* Modifies globals: Var1
;*
;*****
proc Handler1
    integer Event1          ;holds the event number for dialog box 1
    integer StatFax = 0      ;flag for the fax status -- initialized to 0

;obtain the event number
    dlgevent 1 Event1

    switch Event1
;OK button so prepare to send the fax
        case 5
;check to see if the name was entered -- if not then go back and get it
            if nullstr FaxRecipient
                usermsg "The name of the fax recipient is blank"

```

```

        GetRecipient ()
    endif
;check to see if the fax number was entered -- if not then go back and get it
    if nullstr FaxDial
        usermsg "The fax number of the recipient is blank"
        GetRecipient ()
    endif
;send the fax using the FIRST available modem
    faxsend First FaxRecipient FaxDial Multiple FilesToFax
;wait here until the fax has completed sending
    while StatFax <= 1
        statfax = $FAXSTATUS
        yield
    endwhile
;report whether the fax was successful or failed
    if statfax = 3
        usermsg "Fax Sent Successfully"
    else
        usermsg "Fax Failed"
    endif
;delete the text file with the files to be faxed and exit
    delfile FilesToFax
    exit
endcase
;exit button so exit without sending the fax
    case 6
        exit
    endcase
endswitch

endproc

```

Exercise 6b



Create a dialog box that reports the current settings for Fax Retries, Fax Retry Delay Interval, whether or not faxes are automatically printed upon receipt, and whether or not faxes are deleted after sending.

;Demonstration of fax fetch commands

```

*****
;
;*
;
;* NAME.WAS
;* Copyright (C) 1999 Symantec Corporation
;* All rights reserved.
;*
;
; This script is a demonstration of fetch commands and a simple dialog box.
;*
;
*****

*****
;
;*
;
;* MAIN
;* The Main procedure performs fetches to retrieve some of the fax settings,
;* formats the information into strings and then displays the information in
;* a simple dialog box.

```

```

.*
;
.* Calls: none
.* Modifies globals: none
.*
;
.*****
;
proc main

    integer NumRetry      ;numeric and string values for number of retries
    string StrRetry
    integer DelayInterval ;numeric and string values for retry interval
    string StrDelay
    integer PrintOnReceipt ;numeric and string values for printing faxes
    string StrPrint
    integer DeleteOnSend  ;numeric and string values for deleting faxes
    string StrDeleteOnSend

;fetch the current Setup Options for faxing
    fetch fax retries NumRetry
    fetch fax retrydelay DelayInterval
    fetch fax recvprint PrintOnReceipt
    fetch fax delpages DeleteOnSend

;format the current Setup Options for faxing
    strfmt StrRetry "Number of Retries is %i" NumRetry
    strfmt StrDelay "Retry Every %i Mlnutes" DelayInterval
    if PrintOnReceipt
        StrPrint = "Faxes Will Be Printed on Receipt"
    else
        StrPrint = "Faxes Will not Be Printed on Receipt"
    endif
    if DeleteOnSend
        StrDeleteOnSend = "Faxes Will Be Deleted after Sending"
    else
        StrDeleteOnSend = "Faxes Will not Be Deleted after Sending"
    endif

;display the current Setup Options for faxing
;The OK button destroys the window so we don't need to process the button event
    dialogbox 0 8 20 264 169 71 "Set / Fetch Examples"
    text 6 90 8 82 11 "Selected Fax Information" left
    text 1 22 31 212 11 StrRetry left
    text 2 21 55 212 11 StrDelay left
    text 3 21 79 212 11 StrPrint left
    text 4 21 103 212 11 StrDeleteOnSend left
    pushbutton 5 112 144 40 13 "OK" OK
enddialog

endproc

```

Exercise 7a



Write a script that will find your AUTOEXEC.BAT and report its size and date. Check to see if MYAUTOEXEC.BAT exists in the ASPECT directory. If it exists, rename it to MYAUTOEXEC.OLD. Then copy AUTOEXEC.BAT to the ASPECT directory and rename it to MYAUTOEXEC.BAT.

```

;Exercise6a.was Sample Script
.*****
;

```

11/23/99

COPYRIGHT © 1999 SYMANTEC

INTERMEDIATE ASPECT SCRIPTING.DOC

ALL RIGHTS RESERVED.

```

.*
.*
.* MAIN
.* The Main procedure finds your AUTOEXEC.BAT and report its size and date.
.* The script then checks to see if MYAUTOEXEC.BAT exists in the ASPECT
.* directory. If it exists, it will be renamed to MYAUTOEXEC.OLD.
.* Then the script will copy AUTOEXEC.BAT to the ASPECT directory and
.* rename it to MYAUTOEXEC.BAT.
.*
.*
.* Calls: Proc1, Proc2
.* Modifies globals: Var1
.*
.*
.*****
proc main

    ;variables to hold the date and size of the file
    string MyAutoDate
    long MyAutoSize
    ;locate the desired file
    findfirst "c:\autoexec.bat"
    ;if not found, notify the user and exit the script
    if FAILURE
        usermsg "Could not find autoexec.bat"
        exit
    endif

    ;convert the system variables to regular variables
    MyAutoSize = $FSIZE
    MyAutoDate = $FDATE
    ;report on the date and size of the desired file
    usermsg "The size of AUTOEXEC.BAT is %ld'n'rThe date is %s" MyAutoSize MyAutoDate
    ;look for myautoexec.bat in the current directory
    findfirst "myautoexec.bat"
    ;if it exists, rename it to .OLD
    if SUCCESS
        rename "myautoexec.bat" "myautoexec.old"
    endif

    ;copy autoexec.bat to the current directory
    copyfile "c:\autoexec.bat" "myautoexec.bat"
    ;report on the success or failure of the copy command
    if SUCCESS
        usermsg "autoexec.bat was copied successfully and renamed to myautoexec.bat"
    else
        usermsg "The file copying process failed"
    endif
endproc

```

Exercise 8a



Create a dialogue window to input 5 names, area codes, phone numbers and company names. For each set of entries, create a comma delimited line in a file. (A check on how well you do this exercise is to import this file into the Connection Directory.)

```

.*Exercise8a.was Sample Script
.*****
.*
.*
.* This script creates a dialogue window to input 5 names, area codes,
.* phone numbers and company names. For each set of entries, the script

```

```

;* creates a comma delimited line in a file. (A check on how well this
;* exercise is done is to import this file into the Connection Directory.)
;*
;
;*****
;
;variables to store the input information
string NameIn, AreaIn, PhoneIn, CompanyIn

;*****
;
;*
;
;* MAIN
;* The Main procedure creates the NAMES.TXT file to hold the comma delimited
;* list of names. Then it calls NamesInput to input the names and store them.
;* It also initializes the WHEN statement to handle events in Dialog Box 0.
;*
;
;* Calls: NamesInput
;* Modifies globals: none
;*
;*****
proc main
;integer counter for the FOR loop
integer counter
;command to process the events in Dialog Box 0
when dialog 0 call Event0Handler
;open the file for the names list. CREATE it if
;doesn't exist. It is a TEXT file so add CR/LF to each line
fopen 0 "names.txt" CREATE TEXT
;loop through 5 entries
for counter = 1 upto 5
NamesInput ()
endfor
;close the file
fclose 0
endproc

;*****
;
;*
;
;* NamesInput
;* The procedure NamesInput creates a Dialog Box for the input of the
;* information for each entry.
;*
;
;* Calls: none
;* Called by: Main
;* Modifies globals: NameIn, AreaIn, PhoneIn, CompanyIn
;*
;*****
proc NamesInput
;blank out the edit boxes
NameIn = ""
AreaIn = ""
PhoneIn = ""
CompanyIn = ""
;allow the user to input the names info
dialogbox 0 136 68 203 125 67 "Names Input"
text 1 17 16 34 11 "Name" left
text 2 17 36 34 11 "Area Code" left
text 3 15 56 52 11 "Phone Number" left
text 4 15 76 34 11 "Company" left
editbox 5 79 16 102 11 NameIn
editbox 6 79 36 102 11 AreaIn
editbox 7 79 56 102 11 PhoneIn
editbox 8 79 76 102 11 CompanyIn
pushbutton 9 82 106 40 13 "OK" OK
enddialog
endproc

;*****
;
;*
;
;* Event0Handler
;* The procedure Event0Handler processes the events in Dialog Box 0. If the

```

```

;* OK button was pressed, the contents of the name and phone edit boxes
;* are formatted and written out to the file.
;*
;*
;* Calls: none
;* Called by: Main (WHEN statement)
;* Modifies globals: none
;*
;*****
proc Event0Handler
    ;variable for which event happened
    integer Event0
    ;string to hold the formatted line
    string FileLine
    ;query to see which event occurred
    dlgevent 0 Event0
    ;only take action if the OK button was pressed
    switch Event0
    case 9
        ;blank out the variable
        FileLine = ""
        ;format the information to a comma delimited list
        strfmt FileLine "%s, %s, %s, %s" NameIn AreaIn PhoneIn CompanyIn
        ;write the formatted information to the file
        fputs 0 FileLine
    endcase
    endswitch
endproc

```

Exercise 9a



Connect to the Symantec BBS using the Connection Directory to make the modem connection. Go to the Top 10 list. Download the first file using Xmodem transfer protocol.

;Sample for download and file I/O

```

proc main
    ;Dial the BBS using the Connection Directory entry
    dial DATA "Symantec"
    ;Don't do anything else until we finish dialing
    while $dialing
        yield
    endwhile
    ;set the desired protocol and its behavior
    set PROTOCOL XMODEM
    set XMODEM OVERWRITE ON
    ;Waitfor query for name then transmit it
    waitfor "ame: "
    transmit "no name^M"
    ;Waitfor and transmit for "Is this you?"
    waitfor "you? "
    transmit "y"

```



```

        ;Password
waitfor "sword: "
transmit "password^M"
        ;Connection data and then "press any key"
waitfor "Key-"
transmit "^M"
        ;1st menu -- choose File
waitfor " settings"
transmit "f"
        ;2nd menu -- choose "top 10"
waitfor "settings"
transmit "o"
        ;3rd menu -- choose "US products"
waitfor "menu"
transmit "1"
        ;4th menu -- choose option 1
waitfor "menu"
transmit "1"
        ;verify download -- press any key
waitfor "Key-"
transmit " "
        ;select transfer protocol -- choose zmodem. The
        ;transfer starts immediately
waitfor "Choose one (Q to Quit): "
transmit "x"
        ;wait for the remote host to tell us to start download
        ;procedure
waitfor "start"
getfile XMODEM "download1.txt"
while $xferstatus
    yield
endwhile
        ;When the transfer completes, there will be a message
        ;to "Press any Key". Wait for that message then press
        ;any key.
waitfor "Key"
transmit " "
        ;Wait for the next menu then press "g" for goodbye and
        ;"1" for exit.
waitfor " menu"
transmit "g"
pause 1
transmit "1"
        ;Pause for connection to clean up then clear the screen
pause 2
clear
endproc

```

Index

- \$faxstatus, 63
- \$XFERSTATUS, 77
- addfilename, 69
- Arrays, 20
- atof, 39
- atoi, 38
- atol, 39
- Breakpoint, 23
- Carriage return, 38
- Case, 18, 51
- chdir, 71
- checkbox, 54
- combobox, 55
- Compiling, 21
- Connection Directory*, 57
- copyfile, 70
- DDE, 79
- DDE Client
 - ddeadvise, 84
 - ddeexecute, 84
 - ddeinit, 84
 - ddepoke, 84
 - dderequest, 84
 - ddeterminate, 84
 - ddeunadvise, 84
- DDE Host
 - advise, 83
 - ASPECTCMD, 81
 - Capture, 81
 - dialload, 82
 - Execute, 82
 - formats, 82
 - getfile, 82
 - Halt, 82
 - help, 83
 - pwexit, 82
 - sendfile, 82
 - status, 83
 - sysitems, 82
 - topics, 82
 - transmit, 82
- DDE Host
 - Dial, 81
- DDE Session, 80
- ddeinit, 80
- Debugging, 21
 - Compile for debug option, 23
- delfile, 70
- Dialing commands, 59
- Dialog boxes
 - dlgevent, 51
 - Events, 50
 - Modal, 46
 - Modeless, 46
 - Programming, 47
 - Styles, 45
- Dialog editor, 40
 - Align controls, 44
 - Controls, 41
 - Layout assistance, 44
 - Make same size, 44
 - Properties, 41
 - Space evenly, 44
 - Tab order, 44

dialogbox, 52	Help, 11
dirlistbox, 56	isfile, 68
dirpath, 56	itoa, 39
dlgdestroy, 53	Layout assistance, 44
dlgevent, 51	Line feed, 38
dlgsave, 53	listbox, 55
dlgupdate, 53	Logical operators, 13
dos, 72	Loops, 14
DOS file commands, 65	ltoa, 39
editbox, 53	makepath, 70
Editor, 11	mkdir, 71
enddialog, 52	Modal, 46
endgroup, 55	Modeless, 46
Escape sequences, 38	numtostr, 39
Events, 50	Operators, 13
Failure/Success, 14	Parameter passing, 19
fax commands, 64	profilerd, 75
faxsend, 61	profilewr, 75
faxstatus, 63	pushbutton, 54
fclose, 73	radiogroup, 55
fcombobox, 55	rename, 71
File I/O Procedures, 73	Reserved words, 12
File Transfers, 77	rmdir, 72
fileget, 69	rstrcmp, 31
fileset, 69	run, 72
findfirst, 65	sdlglopen, 27
findnext, 65	sdloginput, 25
flistbox, 55	sdlgmsgbox, 26
fopen, 73	sdlgsaveas, 28
Format specifiers, 37	sendfile, 77
fputs, 73	Set / Fetch
fstrfmt, 75	dialentry, 57
ftext, 53	dialentry access, 57
ftoa, 39	dialentry areacode, 58
fullpath, 70	dialentry company, 58
getdir, 71	dialentry country, 58
getfile, 77	dialentry dialnumberonly, 58
getfilename, 69	dialentry longdistance, 58
getpathname, 69	dialentry phonenumber, 58

- dialentry scriptfile, 59
- shell, 72
- shortpath, 70
- splitpath, 70
- Standard dialog boxes, 25
 - sdlgopen, 27
 - sdlginput, 25
 - sdlgmsgbox, 26
 - sdlgsaveas, 28
- strcat, 36
- strchr, 33
- strcmp, 30
- strdelete, 33
- strextact, 34
- strfind, 32
- strfmt, 36
- strcmp, 31
- String comparisons
 - rstrcmp, 31
 - strcmp, 30
 - strcmp, 31
 - strcmp, 31
 - strcmp, 31
- String formatting
 - strfmt, 36
- String miscellany
 - strcat, 36
 - strlen, 36
- String parsing
 - strextact, 34
 - strtok, 35
- String replace or insert
 - strdelete, 33
 - strinsert, 33
 - strreplace, 33
 - strupdt, 33
 - substr, 33
- String searches
 - strchr, 33
 - strfind, 32
 - strrchr, 33
 - strsearch, 33
- strinsert, 33
- strlen, 36
- strcmp, 31
- strcmp, 31
- strchr, 33
- strreplace, 33
- strsearch, 33
- strtok, 35
- strtonum, 39
- strupdt, 33
- Styles, 45
- substr, 33
- Success/Failure, 14
- Switch, 18, 51
- text, 53
- When, 16
 - Dialog, 17, 50
 - Elapsed, 18
 - Iskey, 17
 - Quiet, 17
 - Target, 17
 - Userexit, 18
- While loops, 14