About Seven Hills Software

No Copy Protection

We don’t believe in copy protection—all it does is impair the honest user’s ability to use software to its fullest. We strive to provide high quality products at reasonable prices. We hope you will support our efforts by not allowing your family or friends to copy this software.

Postage-Paid Registration

Be sure to complete and return the postage-paid registration card so we can notify you as new versions of this program become available. Updates are always reasonably priced.

Questions and Comments

We always welcome feedback—if you have any questions, comments, or suggestions for improving this product, please let us know! In addition, we would like to hear your ideas for new programs.

Contacting Us

For orders and product information contact us electronically:

E-Mail: <sales@myseource.com>.

For technical questions about a specific product contact us electronically:

E-Mail: <support@myesource.com>

To contact us the “old-fashioned way,” write to:

My eSource
2310 Oxford Road
Tallahassee, FL 32304-3930

Copyrights and Trademarks

Table Of Contents

Script Commands

Spectrum Scripting

Scripts are extremely useful for telecommunications. By using scripts you can automate simple tasks (e.g. typing a password) or complex ones (logging onto a system, sending and receiving mail, downloading files, then logging off). The limits of a script are up to the imagination and skill of the script author.

This section provides basic information about writing and using scripts. Before trying to write scripts you should be familiar with using Spectrum.

Writing A Script

Spectrum's scripting language is very powerful, yet relatively easy to understand because most commands (built-in instructions that tell Spectrum to perform some action) are simple English phrases. For example, can you guess what the command *Play Sound "Welcome"* does? If you guessed that it plays a sound named "Welcome" then you won't have much trouble learning to write scripts for Spectrum!

The best way to learn how to write a script is by doing it! Sit down with this manual and just go through it page by page, learning what each script command does. Many examples are provided in this manual…try them, and modify them to do something slightly different!

To actually write a script you create a text file using Spectrum's built-in editor. In that file you simply write one or more Spectrum commands, placing each command on a line that ends with a Return character. Or you can place several commands on a single line by inserting a semicolon (;) between each command.

For example, this script…

```
Display "^LType your name then press Return: "
Get Line 2
Display "^M^J^JHello there $2!^G^M^J^J"
Stop Script
```

…and this script…

```
Display "^LType your name then press Return: "; Get Line 2; Display "^M^J^JHello there
   $2!^G^M^J^J"; Stop Script
```

…work exactly the same.

The only limitation to combining commands on a single line is that the line (everything up to a Return character) cannot exceed 636 characters after all the replacements are made (replacements are explained shortly).

There are several features that don't affect how the scripts are executed, but they do let you format your scripts to be more readable. The following formatting features can be used:

- Blank lines
- Spaces and/or Tabs at the beginning of a line
- Different fonts, styles, and sizes
- Comments (text you can read but that will be ignored when the script is run)

Running A Script

When Spectrum is told to run a script, it loads the specified script file into memory and begins interpreting the information in that file. A black box appears at the upper-right corner of the menu bar while a script is running.

If an unrecognized command is encountered, Spectrum stops and displays an error message[1]. The error message is displayed for approximately 30 seconds (clicking the mouse or pressing a key will dismiss the error sooner). If the error box disappears automatically, Spectrum hangs up the line. This is a safety feature for scripts that run unattended (by hanging up, online charges are kept to a minimum).

To stop a script press Escape, R, or choose Stop Script from the Script menu. Stopping a script closes any open script files and returns control to the user[2]. *NOTE: Some script commands temporarily block the use of the menu bar; if one method does not work, try another.*

1 Unless an On Error Goto command was previously encountered.

2 If an On Escape Goto command was previously encountered, pressing Escape will jump to the specified label instead of stopping the script. However, choosing Stop Script from the Scripts menu (or pressing R) will stop the script even if an On Escape Goto command was used.

Script Language Definitions

The Spectrum script language consists of three main parts:

- Built-in *commands* that tell Spectrum to do something.
- *Parameters* you supply that tell a command exactly what to do.
- *Special characters* that are treated differently than normal characters.

The following sections describe each of these components (in reverse order because you need to know about the special characters before the parameter descriptions will make sense, and you need to know about parameters before the command descriptions will make sense).

Throughout this manual, “host” refers to *any* system to which you are connected (e.g. if you are connected to GEnie, GEnie is the “host” system). If your friend calls your computer, he is still the host because he is the system to which you are connected (from his perspective, you are the host).

Specially-Treated Characters

The following sections discuss the “special” characters used when writing scripts.

" (String Delimiter)

The double quote mark (") is used to indicate a “string” (a sequence of characters). For example:

An empty string (no characters):	""
A single character string:	"A"
A several character string:	"Greetings, Earthling!"
A string that happens to be a number:	"1234"

NOTE: There is a script command that lets you redefine the string delimiter to be some other character, but normally it should not be changed.

^ (Control Character)

The caret (^) tells Spectrum that the following letter is a control character. You use control characters to enter carriage returns, linefeeds, and other characters that cannot be entered directly into a script command. At script runtime <u>^*Letter*</u> is replaced by the actual control character. *NOTE: Capitalization of control characters does not matter (^b and ^B are both the same character).*

To display the caret (^) from within a script, enter it twice (^^) or use the *$^* replacement item (described in the next section). *NOTE: There is a script command that lets you redefine the control character indicator to be some other character, but normally it should not be changed.*

# (Comment Character)

The number sign (#) tells Spectrum that the rest of the line is a comment. Comments are not executed when a script is run Comments are useful to document what your script is doing so it will be easier to modify in the future. When typing a comment you must include a space after the # sign, then type the comment.

If the number sign is at the beginning of a line, the first word after the sign is also a *Label* (described later). If you want to include a comment but do not want it to be a *Label* then you can use the alternate comment indicator *Rem* or *Remark*.

Example:

```
    # This is a comment, and "This" is also a Label
    Rem This is just a comment
    Remark This is just a comment
```

$ (Replacement Item)

The dollar sign ($) tells Spectrum that what follows is a replacement item. A replacement item works by completely replacing the *$Item* with that item’s current contents, just as if you had actually typed the contents.

For example, the *$Date* replacement item gets replaced by the current date, in the form of “dd mmm yy”. If today’s date is November 19, 1993, when this script is run…

```
    Display "Today is $Date.^M^J"
```

…it works *exactly* as if you had actually typed:

```
    Display "Today is 19 Nov 93.^M^J"
```

Of course, the advantage to using the *$Date* replacement item instead of actually typing the date is that when the date changes, so will the message that is displayed.

Note that no spaces are inserted when you use a replacement item—the *$Item* is replaced *exactly* as if you had typed it from the keyboard!

To demonstrate this point further, if these assignments have been made…

```
    Set Variable 0 "12" # $0 is now 12
    Set Variable 1 "14" # $1 is now 14
    Set Variable 2 "12,14" # $2 is now 12,14
    Set Variable 3 "GotoXY 12,14" # $3 is now GotoXY 12,14
```

…then all of these statements are exactly identical when the script is run:

```
    GotoXY 12,14
    GotoXY $0,$1 # Same as typing GotoXY 12,14
    GotoXY $2 # Same as typing GotoXY 12,14
    $3 # Same as typing GotoXY 12,14
```

The following replacement items are available in Spectrum. *NOTE: Capitalization of replacement items does not matter—$RATE, $rate, $Rate, and $rAtE are all the same item.*

$$

Gets replaced by a single $.

Example:

```
    Display "The cost is $$12.34^M^J"
```

$^

Gets replaced by a caret (^).

Example:

```
    Display "^LSee what $^L does?^M^J"
```

$#

# must be replaced by a number from 0 to 9

Gets replaced by the contents of the specified variable (see the *Set Variable* command).

Example:

```
    Set Variable 5 "Hello there!^M^J"
    Display "$5"
```

$Length#

# must be replaced by a number from 0 to 9

The length of the contents of variable number #.

Example:

```
    Set Variable 6 "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    Display "There are $Length6 characters in '$6'.^M^J"
```

$FKey#

# must be replaced by a number from 0 to 9

Gets replaced by the contents of the specified FKey (see the *Set FKey* command).

Example:

```
    Set FKey 3 "a frequently-used phrase"
    Display "FKey 3 is '$FKey3'. Try pressing OpenApple-3.M^J"
```

$Version

Gets replaced by the software’s name and version number (e.g. Spectrum 1.0).

$UserName

Gets replaced by the personalization that was entered when Spectrum was installed (the same name that is displayed in the About dialog box).

$OnlineDisplay

Gets replaced by the current online display name, as displayed in the Online Display dialog box.

$DisplayVersion

Gets replaced by the version number of the currently-chosen online display.

$DateTimeStamp

Gets replaced by a ProDOS-compatible *Filename* that represents the current date and time (e.g. D17Sep94T1040).

$Date

Gets replaced by the current date (e.g. 3 Nov 93, 17 Sep 94).

$Month

Gets replaced by the current month number (01-12).

$MonthText

Gets replaced by the current month abbreviation (Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec).

$Day

Gets replaced by the current day number (01-31).

$DayText

Gets replaced by the current day abbreviation (Sun Mon Tue Wed Thu Fri Sat Sun).

$Year

Gets replaced by the current year number (00-99).

$Time

Gets replaced by the current clock time with no seconds or am/pm indication (e.g. 10:40).

$FullTime

Gets replaced by the current clock time with seconds and am/pm indication (e.g. 10:40:59 am).

$Hour

Gets replaced by the current hour (01-23).

$Minute

Gets replaced by the current minute (00-59).

$Second

Gets replaced by the current second (00-59).

$Rate

Gets replaced by the rate that was set by the *Set Rate* command.

$Timer

Gets replaced by the current timer value, in the form “00:01:32”.

$Cost

Gets replaced by the current cost of a call. The cost is calculated as *$Rate* divided by 60 times *$Timer* seconds, and is shown with two decimal places (e.g. 12.34).

Example:

```
    Set Rate 10; Set Timer On; Pause 10
    Display "The current cost is $$$Cost^M^J"
```

Note that in this example the first *$$* gets replaced by a single $, and *$Cost* gets replaced by the current cost. Thus something like the following is displayed on the screen:

```
    The current cost is $1.67
```

$Matched

Gets replaced by the item number that was matched in a *WaitFor* command.

$MatchString

Gets replaced by the string that was matched in a *WaitFor* command.

Example:

```
    WaitFor String "cat" "dog"
    Display "Got '$MatchString' (choice #$Matched).^M^J"
```

$ErrorMsg

Only meaningful in an “On Error Goto” routine

Gets replaced by the error message that would have been shown to the user if the script aborted normally.

$PTimer

Gets replaced by the current prompt timer value that was set by the *Set PTimer* command.

$CurrentX

Gets replaced by the current horizontal cursor position.

$CurrentY

Gets replaced by the current vertical cursor position.

$StoredX

Gets replaced by the horizontal cursor position that was stored by the *Store XY* command.

$StoredY

Gets replaced by the vertical cursor position that was stored by the *Store XY* command.

$PhoneEntries

Gets replaced by the number of phonebook entries listed in the Dial Number dialog box.

$ForValue#

# must be replaced by a number from 0 to 9

The current counter value of loop number #.

Example:

```
    For 7 11 37; Display "Value is $ForValue7.^M^J"; Next 7
```

$SFPrefix

Gets replaced by the current prefix 8 (usually the *Foldername* that was last used in one of Spectrum's "Open" or "Save" dialog boxes).

$Boot

Gets replaced by the *Volumename* of the disk you used to start the system.

$SpectrumPath

Gets replaced by the *Foldername* from which Spectrum was launched.

$SpectrumFile

Gets replaced by the *Filename* of Spectrum (it will be "Spectrum" unless you have renamed it on disk).

$ScriptPath

Gets replaced by the *Foldername* of the script that is currently running.

$ScriptFile

Gets replaced by the *Filename* of the script that is currently running.

$MenuPath

Gets replaced by the *Foldername* of the current menu file. This is the same folder that is searched when an Option-keypress is used to run a script. The *$MenuPath* is set by loading a menu file from the desired folder (see *Load MenuFile*). *NOTE: $ScriptPath and $MenuPath do not necessarily indicate the same folder. $ScriptPath indicates the folder of the* ***currently-running*** *script. If the user selected a script manually (by choosing Run a Script from the Script menu), or if a script command was used to run a script, $ScriptPath could be different from $MenuPath. If you write a script that accesses items that should be stored in the same folder as your script, always use $ScriptPath.*

$MenuFile

Gets replaced by the *Filename* of the current menu file.

$LogonFile

Gets replaced by the *Filename* of the script (if any) that is attached to a phonebook entry. Valid only after a *Dial Entry* or *Dial Service* command.

$FileXferPath

Gets replaced by the *Foldername* used for file transfers (set by the user in the File Transfer dialog on the Settings menu, or by the *Set FileXferPath* command).

$AutoSavePath

Gets replaced by the *Foldername* where the capture buffer will be saved automatically (see *Set AutoSave*).

$LastPath

Gets replaced by the *Foldername* of the last file that was loaded, saved, or sent.

$FrontmostApp

Gets replaced by the *FoldernameFilename* of the frontmost application. If *$FrontmostApp* is equal to *$SpectrumPath$SpectrumFile* then Spectrum is the foreground application, otherwise The Manager is active and Spectrum is in the background.

Color Values

Colors are numbered from 0 to 15. The following replacement items are available to simplify commands that use a color value:

$Black	0	$DarkGreen	8
$Blue	1	$Aqua	9
$Brown	2	$BrightGreen	10
$Gray1	3	$PaleGreen	11
$Red	4	$Gray2	12
$Purple	5	$Periwinkle	13
$Orange	6	$Yellow	14
$Pink	7	$White	15

Parameters

When you use a command there are usually additional “parameters” you must specify so that the command knows exactly what to do. For example, in the statement *Pause 5*, “Pause” is the command and “5” is a parameter that tells the command how long to pause.

In this manual, command parameters are displayed in *italics* so you can see exactly what part of the command you may need to complete. Study this partial example of a command description:

Pause *Value*
Value is optional; if used it can be from 0 to 65535

On the first line the required part of the command is in **boldface** and parameters you might supply are in *italics*. The second line contains special notes about the parameter (it tells you that you don’t *have* to specify a value, but if you do that it must be a number from 0 to 65535).

There are several parameters that are commonly used. Those common parameters are not redefined or noted in a command’s description unless further explanation is required.

The common parameter types which are described in this section are:

- VarNum
- Value
- Character

- String
- Volumename, Foldername, and Filename
- Label
- Statement

VarNum

Variables are used to store information in memory. Ten variables are available, represented by the numbers 0 through 9.

Wherever a *VarNum* parameter is specified you can use one of the following:

- A number from 0 through 9
- Any replacement item that results in a number from 0 through 9

Value

A *Value* is a positive integer (no negative numbers and no fractions or decimals). The exact restriction for a *Value* parameter depends upon the command, so it is noted in each command's description.

Wherever a *Value* parameter is specified you can use one of the following:

- A number (4, 2, 7, etc.)
- A string that represents a number ("4", "2", "7", etc.)
- Any replacement item that results in a *Value*

Character

A *Character* is any single character available on the Apple IIGS.

Wherever a *Character* parameter is specified you can use one of the following:

- A single character, with or without quote delimitation marks (A, g, 7, “R”, “k”, “3”, “^M”, “^J”)
- A value—without the quote delimitation marks—corresponding to the desired character’s ASCII code. The value can be specified as a decimal number (10 through 255 only; 0 through 9 would be considered a single character) or as a hexadecimal value ($00 through $FF). *NOTE: To specify a hexadecimal value you must use $$xx (as in “Set Quote $$22”).*
- Any replacement item that results in a *Character*

Examples:

```
Set Quote \     or  Set Quote "\"
Set Quote 92    or  Set Quote "92"
Set Quote $$5C  or  Set Quote "$$5C"
```

Note the use of *two* dollar signs to indicate a hexadecimal number. Remember, this is required because a single $ indicates a “replacement item”…when Spectrum first expands the *Set Quote $$5C* command, the *$$* gets replaced by the single $, so Spectrum actually sees $5C.

String

A *String* is composed of 0 to 128 *Characters*.

When used in a script, strings should be enclosed in delimitation marks (they are required if the string contains spaces; otherwise the marks are optional). *NOTE: The “delimitation mark” is usually the double quote (") mark. Although the string delimitation mark can be changed by a script, the examples presented in this manual all use the standard double quote mark.*

Wherever a *String* parameter is needed you can use one of the following:

- A single character, with or without quote delimitation marks (A, g, 7, “R”, “k”, “3”, “^M”, “^J”)
- A word, with or without quote delimitation marks (dog, CAT, “house”, “Bread”)
- Two or more words within quote delimitation marks (“wild dog”, “Pretty blue cat^M^J”)
- An empty string (""), which has zero characters
- Any replacement item that results in a *String*

Volumename, Foldername, and Filename

Volumename: The name of a disk (it does not include any folder or file names). For example:

```
:Hard:
:Macintosh Disk:
```

Foldername: Either just the name of a disk or the name of a disk plus the names of one or more folders (or directories) on that disk. For example:

```
:Hard:
:Hard:Spectrum:Spectrum.Script:
:Macintosh Disk:Files to send/receive:
```

Filename: The name of a file (it does not include a *Volumename* or *Foldername*). For example:

```
Capture.File
Today’s Email
```

Remember these important notes about pathnames:

- Always include pathnames in the string delimiter mark (normally the double-quote mark). Although the marks are optional if a pathname has no spaces in it, spaces are entirely possible when using AppleShare or HFS disks. If you don’t include the string delimiter marks, a script that works on a ProDOS disk could fail if it is used on an HFS disk.
- Always use colons (:) at the beginning of a *Volumename*, and to separate the names of each folder in a *Foldername*. Although Spectrum allows the old ProDOS-style “/” indicator, it is better to use the GS/OS-style “:” indicator.
- Wherever a *Volumename*, *Foldername*, or *Filename* is needed you can use any replacement item that results in the proper parameter.
- Wherever a *Volumename* or *Foldername* is needed you can use a GS/OS prefix number if it is set correctly. See the *Set GSPrefix* command for more information.
- Wherever a *FoldernameFilename* parameter is indicated you can omit the *Foldername* if prefix 8 is set to the proper folder. *NOTE: It is safer to always include the desired Foldername because prefix 8 can be changed by the user at almost any time.*

Label

Wherever a *Label* is needed you must specify a label that is defined somewhere in your script.

To define a label in your script, type the comment character (#) at the beginning of a line, press the Spacebar, then type a word that will be the label. If you also want to add a comment, press the Spacebar then type the comment.

Look at the following examples in the left column to see if you can determine which items are valid labels. The right column shows the correct answer and reason for each item:

`# What is a label?`	*What* (the first word after the # at the beginning of the line)
`# Say_Hello`	*Say_Hello* (the first word after the # at the beginning of the line)
`Display "Bye!" # The End`	(no label; # is not at the beginning of the line)
`# AskQuestion -- Asks the user a question`	*AskQuestion* (the first word after the # at the beginning of the line)
`# AskQuestion--Asks the user a question`	*AskQuestion--Asks* (the first word after the # at the beginning of the line)
`# WhatLabel?ThisOne!`	*WhatLabel?ThisOne!* (the first word after the # at the beginning of the line)

Statement

Wherever a *Statement* is needed you can insert any valid Spectrum command, including multiple commands separated by semicolons. You can even use any replacement item that results in a *Statement.*

Script Commands

This section describes every Spectrum script command. The function of each command is explained, but in-depth descriptions from the Spectrum Reference manual are *not* repeated here. For example, the *Set Baud* command states that it controls the port setting's baud rate, but it does *not* explain what baud rate is or how it affects telecommunications. For that information, refer to the Spectrum Reference manual.

Each command is described in the following format:

Pause *Value*
Value is optional; if used it can be from 0 to 65535

Pauses script execution for *Value* seconds. A value of 0 pauses forever; if no value is given there is a one second pause.

Example:
```
Display "Hold your breath..."
Pause 5; Display "^Gbreathe normally now!^M^J"
```

The first line displays the required parts of the command in **boldface** and parameters that you might supply in *italics*. In the sample above, **Pause** is the command and *Value* is the only parameter.

If a parameter needs explanation or clarification, it appears below the command name. In the sample above, it is clarified that *Value* is optional, but if specified it must be a number from 0 to 65535.

Following these items is a description of the command as well as any other notes of interest. From the sample above we now know that Pause will cause a delay of *Value* seconds, and we know what happens if no value is specified, or if a value of 0 is used.

An example is often provided to clarify and spark ideas (they are not necessarily useful, as the example above illustrates).

Set Debug *State*

State can be Screen, Scrollback, or Off

Helps you trace exactly what is happening when a script is run so you can detect errors or omissions. Each script statement is displayed or recorded after all replacements have been made, just prior to executing the expanded statement.

Screen: Statements are displayed on the screen. In some online displays the commands are shown in inverse text to help separate them from the rest of the screen. This option is useful for short scripts that display very little on the screen. It can be used for longer scripts, but the displayed statements may go by too quickly to effectively trace what is happening.

Scrollback: Statements are placed into the scrollback buffer. This option is usually more useful than Screen because it provides a written record of all the statements that were executed, as well as showing *when* they were executed.

Off: When debugging is off, scripts execute with no debug information being displayed or recorded.

Example:

```
Clear Scrollback; Set Debug Scrollback
Display "How^M^Jdoes^M^Jthis^M^Jlook?^M^J"
Set Debug Off
```

Clear Scrollback

Clears the scrollback buffer, which is sometimes useful during debugging (especially when sending debug commands to the scrollback buffer). *NOTE: This command should never appear in a completed script!*

Set ShowControls *State*

State can be Off or On

When on, control characters are displayed as ^*letter*, which is sometimes useful for debugging scripts. For example, if you're waiting for "Hello" but the host is transmitting "Hel^Tlo" you may not know why the script isn't working; by turning on ShowControls you will be able to see the problem.

For this command to be most useful, create and select character filter tables that do no key translations. *NOTE: Some online displays do not support this command. This command should never appear in a completed script!*

Fundamental Commands

Display "*String*"

Displays *String* on the screen. If you also want to send the string out to the port, use the *Transmit* command.

No characters are added automatically by the *Display* command. If you want to display a word followed by a carriage return you must include the carriage return in the *Display* command (see the example below).

Because some online displays do not automatically add a linefeed (^J) when a carriage return (^M) appears, you should always display the "^M^J" combination to ensure the cursor is at the first position of the next line. Displays that do automatically linefeed will ignore the first ^J after a ^M.

Example:

```
    Display "^LThis is "
    Display "a test.^M^J"
    Display "^GSee how these^M^Jlines look?^M^J"
```

The following control characters behave in a standard way when used in a *Display* command:

Key	Result
^E	Show the cursor
^F	Hide the cursor
^G	Play the system beep sound
^H	Move the cursor one character to the left
^J	Move the cursor down one line
^K	Clear the screen from the cursor on down, without changing the cursor position
^L	Clear the display and move the cursor to the top-left of the display
^M	Move the cursor all the way to the left of the current line (and sometimes down one line too)
^Y	Move the cursor to the top-left of the display without clearing the display
^]	Clear from the cursor to the end of the line
^^	GotoXY (the ASCII value of the next two characters are interpreted as X+32 and Y+32)

NOTE: To insure that you are at the beginning of the next line, always display the combination "^M^J".

NOTE: You cannot directly display "^^" followed by two letters, but you can do it with the following script:

```
    Set Token \ # change the control character token to be \
    Display "\^!*" # same as GotoXY 1,10
    Set Token ^ # change token back to ^
```

The following additional control characters are not supported in all online displays, but if supported they behave as described:

Key	Result
^I	Move the cursor right, to the next “tab stop” (usually at positions 0, 8, 16, 24, and so on)
^N	Normal text (turns off Inverse, and sometimes MouseText too)
^O	Inverse text
^V	Scroll the screen down one line without changing the cursor position
^W	Scroll the screen up one line without changing the cursor position
^X	Turn off MouseText (and sometimes Inverse too)
^[	Turn on MouseText (uppercase letters A-Z are displayed as “MouseText” characters)
^\	Move the cursor right one character
^_	Move the cursor up one line
^Z	Clear the entire line that the cursor is on

NOTE: Displays that support MouseText have different requirements. To ensure MouseText is on, always display the combination “^O^[“; to ensure MouseText is off, always display the combination “^X^N”.

DisplayRecord "*String*"

Displays *String* on the screen and records it into the capture buffer (even if the screen and buffer have been turned off). If you also want to send the string out to the port, use the *Transmit* command.

Example:

```
    DisplayRecord "^M^J[Begin online session on $Date at $FullTime.]^M^J"
```

In the example, note that the ^M^J combination is there for the benefit of the “Display” part of the command…recording to the capture buffer does not need a linefeed.

Record "*String*"

Records *String* directly into the capture buffer (even if the buffer has been turned off). If you also want to send the string out to the port, use the *Transmit* command.

Example:

```
    Record "^M[Begin online session on $Date at $FullTime.]^M"
```

In the example, note that only ^M is used…the capture buffer does not need a linefeed.

Transmit "*String*"

Shortcut: **Xmit "***String***"**

Sends *String* directly out to the port. If the port is set for half duplex the string is also displayed on the screen; with full duplex the string is displayed only if the host echoes it.

No characters are added automatically by the *Transmit* command. If you want to transmit a word followed by a carriage return you must include the carriage return in the *Transmit* command (see the example).

Example:

```
    Transmit "MyPassword^M"
```

Break

Sends a “break” signal to the port.

Load MenuFile "*FoldernameFilename*"

Loads the specified file as if the user selected it by choosing Load Menu File from the Script menu.

Clear MenuFile

Removes any script names from the bottom of the Scripts menu, and resets the menu path to the Spectrum.Script folder.

Set FKey *Value* "*String*"

Value can be from 0 to 9

Sets an “FKey” to the specified *String*. From then on pressing # (where # is a number from 0 to 9) will type the string just as if you typed it from the keyboard (if a display window is open and in front; it does not work in the Editor, in NDAs, etc.).

The FKeys are available even after the script has stopped running, which makes them very useful for quickly typing common phrases.

Example:

```
Set FKey 1 "For more Spectrum information contact:^M   Seven Hills Software^M   2310 Oxford
   Road^M   Tallahassee, FL 32304^M" # From now until Spectrum is quit (or another Set FKey 1 is
   encountered) pressing OpenApple-1 will type the complete address when the online display is
   open.
```

Settings

Save Settings

Saves Spectrum’s current settings to disk. Settings are also saved when Spectrum is quit (unless the user has checked the “Don’t save settings” option in the Status dialog box).

Load Settings

Loads Spectrum’s settings from disk.

Store Settings

Remembers Spectrum’s settings in memory.

Every script command that controls a checkbox or popup menu in a Spectrum dialog box is affecting choices the user has made. Whenever possible you should not permanently alter the user’s settings of those items. This example shows one way to change settings without permanently affecting the user’s choices:

```
Store Settings # hold current settings in memory
Set AutoReceive OFF # turn autoreceive off during logon
Set Duplex FULL; Set Echo OFF # don't show logon information
...logon commands here...
Restore Settings # restore the AutoReceive, Duplex, and Echo settings to whatever they were
  originally
```

Restore Settings

Restores from memory the previously-stored program settings.

Set SmartPaste *State*

Controls whether Spectrum’s built-in editor will use “smart” cutting and pasting. If you change this option, the editor window must be closed (W) then re-opened (E) before the change takes effect.

With SmartPaste turned on, cutting and pasting will insert and delete spaces automatically in an effort maintain the proper spacing between words. However, this option is normally turned off because it can be annoying when editing scripts.

Port Settings

NOTE: The commands described in this section control settings related to the communications port. Most commands affect options that the user can set in the Port Settings dialog box.

Set Baud *Rate*

Rate can be Default, 50, 75, 110, 135, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200, 38400, or 57600

Sets the "Baud Rate" option to the specified rate.

Set DFormat *Format*

Format can be 7E2, 7O2, 7N2, 7E1, 7O1, 7N1, 8E2, 8O2, 8N2, 8E1, 8O1, or 8N1

Sets the "Data Format" option to the specified data bits (7 or 8), parity (None, Even, or Odd), and stop bits (1 or 2).

Set Duplex *DuplexKind*

DuplexKind can be Half or Full

Controls the "Half Duplex" checkbox. With half duplex, outgoing data is displayed on the screen as it is transmitted; with full duplex outgoing data is *not* displayed on the screen. Setting full duplex and setting the screen off

Set Echo *State*

State can be Off or On

Controls the "Echo" checkbox. If echo is on, incoming characters are echoed back out to the port. *NOTE: If Echo is on and the host echoes characters back to you, you may find yourself in a loop with the same characters constantly being echoed between the two systems. Turn echo off to stop the loop.*

Also, commands which normally are displayed only on the screen will be sent out to the port (e.g. Get Line, Show Catalog, Show File, etc.). *NOTE: When echoing information out to the port, characters are not passed through the keyboard filter (as they would be if they were sent using the Transmit command).*

Set SendLFs *State*
State can be Off or On

Controls the "Send LFs" checkbox.

Set Handshake *State*
State can be Off or On

Controls the "H'ware Handshake" checkbox.

Set XonXoff *State*
State can be Off or On

Controls the "Xon/Xoff Flow" checkbox.

Set CharDelay *Value*
Value can be from 0 to 9

Controls the "Character Delay" value. *Value* is the delay in 1/60ths of a second (0 is no delay; 9 is a .15 second delay).

The delay occurs after each character is sent out to the port and is used at all times except for non-text file transfer protocols.

Set LineDelay *Value*
Value can be from 0 to 9

Controls the "Line Delay" value. *Value* is the delay in 16/60ths of a second (0 is no delay; 9 is a 2.4 second delay).

The delay occurs after a Return character is sent out to the port and is used at all times except for non-text file transfer protocols.

Set DCD *State*

State can be Off or On

Controls the “DCD Handshake” checkbox.

Set Port *PortKind SlotNumber*

PortKind can be IIGS or Slot
SlotNumber can be a number from 1 to 2 if *PortKind* is IIGS; from 1 to 7 if *PortKind* is Slot

Controls the port connection options:

IIGS: Indicates the connection should be through the serial port on the back of the IIGS computer. To specify the Printer Port use 1 for *SlotNumber*; to specify the Modem Port use 2.

Slot: Indicates the connection should be through the card that is plugged into the specified slot.

Examples:

```
    Set Port Slot 2 # a Super Serial Card in slot 2
    Set Port Slot 4 # an AE DataLink in slot 4
    Set Port IIGS 2 # the IIGS Modem Port
```

Online Display Settings

NOTE: The commands described in this section control settings related to the online display. Most commands affect options that the user can set in the Online Display Settings dialog box.

Set OnlineDisplay "*Display*"

Display can be the name of any online display as it appears in the Online Display Settings dialog box

Selects and opens the specified online display. The Spectrum SHR Fast, Spectrum SHR Normal, and Spectrum Text displays will always be available, but others can be added or removed by the user at any time. You can test the Failed flag to see if the display was opened or not.

Example:

```
Set OnlineDisplay "ANSI"
If Failed Then Display "^LThis script requires the ANSI display, which could not be
  opened.^M^J^J"; Stop Script
```

Close OnlineDisplay

Shortcut: **Offline**

Closes the current online display and stops script execution as if the *Stop Script* command was encountered. *Close OnlineDisplay* is useful for ending a script that uses a custom display—the command will close the display and return to the 640 mode desktop with Spectrum's menu bar.

TIP: Use the Set OnlineDisplay command to select a different online display without causing the script to stop.

Set DeleteBack *State*

State can be Off or On

Controls the “Delete key = Backspace” checkbox. When on, pressing the delete key sends a backspace (^H, ASCII $08). When off, pressing the delete key sends a true delete character (ASCII $7F).

Set LowASCII *State*

State can be Off or On

Controls the “Convert To Low ASCII” checkbox.

Set ScriptKeys *State*

State can be Off or On

Controls the “Recognize Script Keys” checkbox.

Set RemoveLFs *State*

State can be Off or On

When on, linefeed characters (^J, ASCII $0A) are not stored in the capture buffer or seen by scripts. This option is usually on.

Set Sound *State*

State can be Off or On

Controls the “Sounds” checkbox.

Character Filter Settings

NOTE: The commands described in this section control settings related to the character filter. Most commands affect options that the user can set in the Character Filter Settings dialog box.

Set DisplayFilter *Value*

Value can be from 0 to 16

Sets the Display Character Filter to the specified table number. Table 0 is “Default” (which uses the translation setting specified in the General CDEV). Other table numbers correspond to tables that have been added using the Character Filter Editor.

Set KeyFilter *Value*

Value can be from 0 to 16

Sets the Keyboard Character Filter to the specified table number. Table 0 is “Default” (which uses the translation setting specified in the General CDEV). Other table numbers correspond to tables that have been added using the Character Filter Editor.

Set Country *Value*

Value can be from 0 to 7

Sets the IIGS Control Panel’s Display Language and Keyboard Layout to the specified *Value*:

```
Set Country 0 # U.S.A.   # @ [ \ ] ` { | } ~
Set Country 1 # U.K.     £ @ [ \ ] ` { | } ~
Set Country 2 # French   £ à ˙ ç § ` é ù è ¨
Set Country 3 # Danish   # @ Æ Ø Â ` æ ø å ~
Set Country 4 # Spanish  £ § ¡ Ñ ¿ ` ˙ ñ ç ~
Set Country 5 # Italian  £ § ˙ ç é ù à ò è ì
Set Country 6 # German   # § Ä Ö Ü ` ä ö ü ß
Set Country 7 # Swedish  # @ Ä Ö Å ` ä ö å ~
```

This is useful for foreign services that use common character codes to display foreign characters. The Control Panel’s Display Language and Keyboard Layout affect only the standard 40/80 column text displays…they do not affect what you see on the Super Hires screen or on a printout.

If you frequently call a foreign service it would be better to create a Character Filter table that translated one character into another and use a Super High Res online display. For example, a filter could translate:

$7B to $8E so an incoming { character would be changed to é, and

$8E to $7B so an outgoing é character would be changed to {

To view standard “option” characters on the Super High Res screen, the “Convert to low ASCII” online display option must be off. However, characters intentionally converted to high ASCII characters using a filter table will display correctly even if the “Convert to low ASCII” option is off.

File Transfer Settings

NOTE: The commands described in this section control settings related to file transfers. Most commands affect options that the user can set in the File Transfer Settings dialog box.

Set FileXferPath "*FoldernameFilename*"

Filename is optional

Sets the default folder and filename for file transfers. “Auto Receive” transfers will be received to or sent from this folder. *NOTE: If your script is receiving a file and does not care where it goes, set the prefix to $FileXferPath.*

Set PadCR *State*

State can be Off or On

Controls the “Pad Empty Lines” checkbox. Some hosts use a blank line to indicate the end of a document. When this option is on, blank lines in the file being sent are padded with a space.

After a file has been sent with this option on, you may need to Transmit “^M” to finish the transfer because CRs within the file have been padded.

Set Prompt *Character*

Establishes the prompt character used when sending text files, and turns on prompting. If *Character* is ^@ or ^M then prompting is turned off.

Set PTimer *Value*

Value can be from 0 to 256

The value indicates how many seconds Spectrum will wait for the prompt character during a “prompted” text file upload. If the prompt is not received within *Value* seconds after sending a line of text, the upload continues.

Set ULTextShow *State*

State can be Off or On

Controls the “Display text and save to buffer” checkbox in the Send Text File dialog box. If this option is on when a text file is uploaded, the text is displayed on the screen and captured just as if it was being typed at the keyboard. If this option is off, the file transfer dialog box appears while the file is being sent.

Set AutoResume *State*

State can be Off or On

Controls the “Resume Transfers” checkbox. When on, interrupted/aborted CIS B+ and Zmodem transfers will be resumed if possible.

Set AutoReceive *State*

State can be Off or On

Controls the “Auto Receive” checkbox. When on, Spectrum watches incoming data to see if the host wants to send a file via CIS B+ or Zmodem.

Because CIS B+ transfers are started by a single character (^E), it is fairly easy to trigger a false CIS B+ transfer. To prevent false triggers you could write a script that turns auto receive off and on when necessary.

Set BinaryII *State*

State can be Downloads, Uploads, Off, or On

Controls the “Binary II Down” and “Binary II Up” checkboxes.

Downloads: If a downloaded file has a Binary II wrapper, the wrapper will be removed from the file as it is downloaded. This option also turns off the “Resume Transfers” checkbox.

Uploads: If a file doesn’t already have a Binary II wrapper, one will be added as the file is uploaded.

Off: Turns off both checkboxes.

On: Turns on both checkboxes, and turns off the “Resume Transfers” checkbox.

Set SendAhead *State*

Controls the “Packet Send Ahead” checkbox. If this option is on then CIS B+ and Zmodem transfers will not wait for an acknowledgment of receipt of the previous packet before sending another packet. This can decrease file transfer time, but should be used only with a good connection.

Set ProDOSX *State*

State can be Off or On

Controls the “ProDOS Xmodem” checkbox. When on, extra information is transmitted about a ProDOS file (the file’s length, file type, creation/modification date, etc.). Useful mainly when sending a file from one Apple to another or with systems that support this feature.

Set RelaxedXfers *State*

State can be Off or On

Triples the normal wait times for all protocol transfers during this session of using Spectrum (the normal wait times are re-established the next time Spectrum is started).

This command is useful when connected to a host that is slow to respond during file transfers. If you find transfers are getting aborted due to timeout errors, this command can help (it cannot help if the problem is line noise).

Set Turbo *State*

State can be Off or On

Sets “Turbo” mode for Ymodem file transfers. When off, Receive Ymodem uses regular Ymodem protocol; when on it uses Ymodem-g. Ymodem-g avoids the typical delays between transmittal of each block, but must be used only with a good connection because if a single bad packet is encountered the whole transfer is cancelled and must be restarted from scratch.

Set ZErrors *Value*

Value can be from 0 to 65536

Sets the number of errors that will be allowed during a Zmodem transfer. The transfer will be aborted if more than *Value* number of errors occur.

Dialing

NOTE: If you're writing a script that will be attached to a phonebook entry in the Dial Number dialog, or that uses one of the Dial commands, be aware that the script should not wait for things like "CONNECT" or "BUSY"—Spectrum's dialing routines are complete (if they do not fail then you are connected).

Set ConnectWait *Value*

Value can be from 1 to 999 seconds

Specifies how long to wait for a connection before cancelling the dial sequence.

Example:

```
    Set ConnectWait 60
```

Get PhoneEntry *Value* *VarNum*

Value can be a number from 1 to *$PhoneEntries*

Extracts the name of the specified phonebook entry and stores it in the specified variable.

Dial String "*String*"

Prepares the given number for dialing by adding "ATDT" or "ATDP" if needed, then dials the number. Because scripts have greater control over redialing, this command does not redial automatically.

You can check the Failed flag to determine whether or not a connection was made. *NOTE: If the user pressed the ESC key to cancel dialing, the Keyboard value will be ^[(ESC).*

Example:

```
Dial String "555-1234"
If NOT Failed Then Display "[CONNECTED AT $Time ON $Date]^M^J"; Stop Script
If Keyboard "^[" Then Display "Changed your mind, eh?^M^J"
```

Dial Service "*PhonebookEntry*"

PhonebookEntry can be the name of any phonebook entry that appears in the Dial Number dialog box

Extracts the specified entry, sets the port settings, then dials the entry's phone number. Because scripts have greater control over redialing, this command does not redial automatically.

You can check the Failed flag to determine whether or not a connection was made. *NOTE: If the user pressed the ESC key to cancel dialing, the Keyboard value will be ^[(ESC).*

If the phonebook entry has a script attached to it, that script is *not* automatically run. Instead, the *$LogonFile* replacement item is set to the name of the attached script. This way your script can decide whether to use an entry just for connecting to a service, or it can connect and run the script attached to the phonebook entry (see the example below).

Example:

```
Dial Service "GEnie"; If Failed Then Stop Script
Set SFPrefix "$MenuPath" # This is the folder that the last menu file was loaded from, which is
  where the logon script should be located
If Exists "$LogonFile" Then Run "$LogonFile" # If the logon file doesn't exist then no error
  appears
```

Dial Entry *Value*

Value can be a number from 1 to *$PhoneEntries*

Dials phonebook entry number *Value* exactly like the *Dial Service* command. The *Dial Entry* command is useful for automatically dialing one entry after another until one connects. For dialing a specific service the *Dial Service* command should be used because phonebook entries can change position whenever an entry is added or deleted.

Example:

```
If Equal "$PhoneEntries" "0" Then Stop Script # because there are no entries to dial
For 5 1 $PhoneEntries; Get PhoneEntry $ForValue5 3
Display "Trying entry number $ForValue5 ($3)...^M^J"; Dial Entry $ForValue5; If Not Failed Then
   Clear For 5; Display "^LConnected to $3!^M^J"; Stop Script
Next 5; Display "Couldn't connect.^M^J"; Stop Script
```

Hangup

Hangs up the modem without asking for confirmation.

Set Password *Value* "*String*"

Value can be from 0 to 9

Encrypts *String* and saves it with Spectrum. The stored password may be transmitted using the *Send Password* command.

The following table shows the recommended use for the password values:

0	The response given at GEnie’s “U#=” prompt
1	The response given at CompuServe’s “Password” prompt

By using stored passwords and the *Send Password* command, generic scripts can be written and posted online without having to remove your private password information.

Send Password *Value*

Value can be from 0 to 9

Decrypts the stored password and sends it out to the port with no screen echo (unless the host echoes it).

Set Rate *Value*

Value can be from 0 to 65535

Sets the value used to calculate the *$Cost* replacement item. The value should be the cents (or pence, kopeks, etc.) per minute. For example, if a service costs $6.00 per hour, the cost per minute is 10 cents (**6.00***100/60).

Set Timer *State*

State can be Off or On

Turns the timer on and off. The current timer can be displayed using the *$Timer* replacement item. A charge based on the timer can be calculated by first setting the rate *(Set Rate)* then displaying the *$Cost* replacement item.

Clear Timer

Clears the timer back to 00:00:00; it does not turn the timer on or off.

Play Sound "*Name*"

Plays the specified sound at the volume specified in the Sounds CDEV. *NOTE: In order to play sounds, the Sounds CDEV must be installed and active, and the "Sounds" checkbox must be on in the Online Display Settings dialog box.*

Name is case-sensitive ("welcome" and "Welcome" are *not* the same sound). The Failed flag is set if the specified sound was not found (Spectrum will play sounds in the Spectrum.Sounds file, or in any of the sound files found in the System:Sounds folder). *NOTE: System 6.0.0 contains a bug that sometimes prevents the correct sound from being played (e.g. playing "You Have Mail (HAL)" will actually play "You Have Mail"). System 6.0.1 works correctly.*

To avoid one sound cutting another sound short, Spectrum waits until no sound is playing before it asks the Sounds CDEV to play the requested sound. This means that sounds can be "combined" by playing one right after another.

Two sound files come with Spectrum: SP.Snds.Main and SP.Snds.Aux.

The sounds in the **SP.Snds.Main** file are played automatically by Spectrum at the appropriate time:

Spectrum Welcome	File-Launch	File-Quit
Phone-Connected	Phone-No Connection	
Phone-Hangup		
Send/Receive-Good	Send/Receive-Bad	
Chatline Warning	SystemBeep	
Saving Screen	Saving Screen Failed	
Key-Return	Key-Spacebar	Key-Any

The sounds in the **SP.Snds.Aux** file are not played by Spectrum, but scripts can play them when appropriate:

Welcome	Goodbye	
You Have Mail	You Have Mail (HAL)	
Reading Mail	Sending Mail	
Receiving Files	Sending Files	
Via Text	Via Xmodem	
Via Ymodem	Via Zmodem	Via CIS B+
Entering Forum	Entering Roundtable	
Reading Messages	Posting Replies	
Loading	Saving	
Capture Buffer	File	
Cleared	On	Off

Example:

```
# NOTE: Although something like this is possible, it can get annoying if overused...
Get File 2 0; If Failed Then Stop Script
Play Sound "Loading"; Play Sound "File"; Load ScriptEditor "$0"
Apply LowASCII; Apply RemoveControls; Apply LFsToCRs; Apply Format 3
Play Sound "Saving"; Play Sound "File"; Save ScriptEditor "$0"; Clear ScriptEditor
Play Sound "Sending A File"; Play Sound "Via Text"; Send "$0" Text
```

Script and Program Control

Stop Script

Stops running the current script.

Example:

```
Display "Thanks for using this script!^M^J"
Stop Script
Display "You won't see this.^M^J"
```

Pause *Value*

Value is optional; if used it can be from 0 to 65535

Pauses script execution for *Value* seconds. A value of 0 pauses forever; if no value is given there is a one second pause.

Example:

```
Display "Hold your breath..."
Pause 5; Display "^Gbreathe normally now!^M^J"
```

WaitFor Time "*Time*"

Waits until the specified time then continues executing the script. The time must be specified in 24-hour format (e.g. “19:40”, “12:34”).

Example:

```
WaitFor Time "19:40"; Display "It is now 7:40pm!^M^J"
```

Run "*FoldernameFilename*"

Runs the specified script, just as if the user chose Run a Script from the Script menu, then selected the *FoldernameFilename* script. Whenever any script is run, the following items are set to a known state:

```
Set Quote "
Set Token ^
Set CaseSensitive OFF
Set Timeout 0
On Escape Goto ""  # Spectrum handles an Escape
On Error Goto ""  # Spectrum handles errors
Set Turbo OFF
Set ScreenBlank OFF
Set Screen ON
Set Debug OFF
Set Rate 0
Set Flush ON
Set ULCapture OFF
# - ALSO:
# - All ten variables ($0-$9) are cleared to ""
# - The "If Keyboard" value is cleared
# - The Failed flag is cleared to indicate "not failed"
# - $Matched is cleared to 0
```

```
    # - $MatchString is cleared to ""
    # - All files opened by the previous script are closed (including any ScriptEditor file)
```
Example:
```
    Display "Set up <C>ompuServe or <G>Enie? "; WaitFor Keyboard
    If Keyboard C Then Run "$MenuPathCIS.Setup"
    If Keyboard G Then Run "$MenuPathGEnie.Setup"
    Display "No system with that letter.^M^J"; Stop Script
```

Chain "*FoldernameFilename*"

Passes control (or "Chains") to the specified script, *preserving* all the settings that are normally reset when a script is Run (see above).

Example:

At the beginning of a script named "Chain.1":
```
    Display "Do you want to <D>elete or <S>how a file? "
    # KeyLoop1
    WaitFor Keyboard
    If Keyboard D Then Set Var 9 "Delete"; Chain "Chain.2"
    If Keyboard S Then Set Var 9 "Show"; Chain "Chain.2"
    Display "^G"; Goto KeyLoop1
```

At the beginning of the script named “Chain.2”:

```
Display "^M^J^JNow in the 'Chain.2' script.^M^J^J"
If Null 9 Goto NotChained # if variable number 9 is empty then this script was RUN, otherwise the
   script was CHAINed to and variable number 9 has the name of a label in this file
Goto $9

# NotChained
Display "This script was RUN (not CHAINed to).^M^J"; Stop Script

# Delete
Display "This script was CHAINed to, and you wanted to DELETE a file.^M^J"; Stop Script

# Show
Display "This script was CHAINed to, and you wanted to SHOW a file.^M^J"; Stop Script
```

Launch "*FoldernameFilename*"

Spectrum hangs up the modem (see “Hangup”) then quits to the specified application. Quitting that application will restart Spectrum. *NOTE: If The Manager is active Spectrum will not be quit; it will just attempt to launch (or activate) the specified application.*

The user is not asked if he wants to save the capture buffer, but a *Save Buffer* command will be issued automatically if the AutoSaveBuffer option is on.

Example:

```
Display "Launch <G>raphicWriter III? "
WaitFor Keyboard
If Keyboard Y Then Launch ":Hard:GWIII:GraphicWriter"
```

Exit "*FoldernameFilename*"

FoldernameFilename is optional; if given it must reference a IIGS application

Spectrum attempts to leave the communication port active (if you are online you should not be disconnected), then…

If *FoldernameFilename* is **not** given then Spectrum quits to the application that launched it. Quitting that application will not restart Spectrum.

If *FoldernameFilename* **is** given then Spectrum quits to that application. Quitting that application will restart Spectrum. *NOTE: If The Manager is active Spectrum will not be quit; it will just attempt to launch (or activate) the specified application.*

In either case the user is not asked if he wants to save the capture buffer, but a *Save Buffer* command will be issued automatically if the AutoSaveBuffer option is on.

Example:

```
Display "^GWARNING: Exiting does not hang up the line; be sure you are not connected to an
   expensive service!^M^J"
Display "Exit to <G>raphicWriter III or <F>ont Factory GS? "
WaitFor Keyboard
If Keyboard G Then Exit ":Hard:GWIII:GraphicWriter"
If Keyboard F Then Exit ":Hard:FFGS:Font.Factory.GS"
```

Quit "*FoldernameFilename*"

FoldernameFilename is optional; if given it must reference a IIGS application

Spectrum hangs up the modem (see “Hangup”) then…

If *FoldernameFilename* is **not** given then Spectrum quits to the application that launched it. Quitting that application will not restart Spectrum.

If *FoldernameFilename* **is** given then Spectrum quits to that application. Quitting that application will not restart Spectrum.

In either case the user is not asked if he wants to save the capture buffer, but a *Save Buffer* command will be issued automatically if the AutoSaveBuffer option is on.

Variables

Set Variable *VarNum* **"***String***"**

Shortcut: **Set Var** *VarNum* **"***String***"**

Set the contents of the specified variable number to be the given string.

Example:

```
    Set Variable 0 "Spectrum is cool!^M^J"
    Display "$0$0$0$0"
```

Store Variables

Shortcut: **Store Vars**

Stores the current values of the ten numbered variables.

Restore Variables

Shortcut: **Restore Vars**

Restores the ten numbered variables to the previously-stored values.

Clear Variables

Shortcut: **Clear Vars**

Clears the ten numbered variables to "".

Concatenate "*String1*" "*String2*" *VarNum*

Appends *String2* onto the end of *String1* and places the result into the specified variable number. If the length of *String1* and *String2* is greater than 128 characters, the result is truncated to 128 characters and the Failed flag is set.

Example:

```
Display "What adjective describes Spectrum? "
Get Line 0
Concatenate "Spectrum is " "$0" 1
Concatenate "$1" "!^M^J" 1
Display "$1$1$1$1"
```

NOTE: Only a single Concatenate command was really needed; we used two steps to illustrate that it is perfectly acceptable to specify a variable for one of the strings, then to store the result back into that same variable.

Position "*String*" "*ToFindString*" *Start VarNum*

Start can be from 1 to 128

Sets variable number *VarNum* to indicate the character position at which *ToFindString* is found in *String*. *Start* indicates which character to start searching from. If *Start* is 0, or if *Start* is greater than the length of *String*, or if *ToFindString* is not found in *String*, then *$VarNum* is set to 0.

Example:

```
Display "Type your first and last name: "; Get Line 0
Position "$0" " " 1 1 # looks for a space starting at character 1; stores the position in $1
If Equal "$1" "0" Then Display "You didn't enter a space between your first and last name."; Stop
   Script
```

Substring "*String*" *Start Length VarNum*

Start can be from 1 to 128
Length can be from 1 to 128

Sets *$VarNum* to the specified portion of *String*. If *Start* is 0, or if *Start* is greater than the length of *String*, then *$VarNum* is set to "". If *Length* is greater than the length of *String* then *$VarNum* contains the characters from *Start* to the last character of *String*.

Example:

```
Display "Type your first and last name: "; Get Line 0
Position "$0" " " 1 1 # looks for a space starting at character 1; stores the position in $1
If Equal "$1" "0" Then Display "You didn't enter a space between your first and last name."; Stop
   Script
Decrement 1 # the character before the space
Substring "$0" 1 $1 2 # everything to the left of the space is put into $2
Increment 1; Increment 1 # the character after the space
Substring "$0" $1 128 3 # everything to the right of the space is put into $3
Display "^M^JYour first name is: $2^M^J"; Display "Your last name is: $3^M^J"; Stop Script
```

Increment *VarNum*

Shortcut: **INC** *VarNum*

Increments the contents of the specified variable number by 1. The variable must contain a valid number in order for this command to work. The number will not increment above 4,294,967,295.

Example:

```
Set Variable 0 "12345"
Display "Starting at: $0^M^J"
Increment 0
INC 0 # the shortcut version
Display " Result is: $0^M^J"
```

Decrement *VarNum*

Shortcut: **DEC** *VarNum*

Decrements the contents of the specified variable number by 1. The variable must contain a valid number in order for this command to work. The number will not decrement below 0.

Example:

```
    Set Variable 0 "12345"
    Display "Starting at: $0^M^J"
    Decrement 0
    DEC 0 # the shortcut version
    Display " Result is: $0^M^J"
```

Get Random *Value VarNum*

Value can be from 1 to 65535

Places a random number from 1 to *Value* into the specified variable number.

Add *Value1 Value2 VarNum*

Adds *Value1* to *Value2* and places the result into the specified variable number.

Subtract *Value1 Value2 VarNum*

Subtracts *Value2* from *Value1* and places the result into the specified variable number.

Multiply *Value1 Value2 VarNum*

Multiplies *Value1* by *Value2* and places the result into the specified variable number.

Divide *Value1 Value2 VarNum*

Divides *Value1* by *Value2* and places the result into the specified variable number. NOTE: Because values are positive integers, any remainder from the division is thrown away (e.g. the result of 5 divided by 2 is 2, not 2.5).

Example:

```
Set Variable 0 "100"; Set Variable 1 "25"
Add $0 $1 2;      Display "$0+$1=$2^M^J"
Subtract $0 $1 2; Display "$0-$1=$2^M^J"
Multiply $0 $1 2; Display "$0*$1=$2^M^J"
Divide $0 $1 2;   Display "$0/$1=$2^M^J"
```

Compare *Value1 Value2 VarNum*

Compares *Value1* to *Value2* and places the result into the specified variable number (1 means *Value1<Value2*; 2 means *Value1=Value2*; 3 means *Value1>Value2*)

Example:

```
Get Random 100 0
Display "^LI picked a number from 1-100. Now you try to guess what it is!^M^J^J"
# GuessLoop
Display "What is your guess? "; Get Line 1
Compare $1 $0 2 # compare the guess to the random number
On $2 GotoNext Less, Equal, More
# Less
Display "^M^JToo low!^M^J^J"; Goto GuessLoop
# Equal
Display "^G^M^JYou guessed right!^M^J^J^G"; Stop Script
# More
Display "^M^JToo high!^M^J^J"; Goto GuessLoop
```

Swap *VarNum1 VarNum2*

Swaps the contents of the two variables.

Example:

```
    Set Var 1 "One"; Set Var 2 "Two"
    Swap 1 2; Display "1=$1 2=$2^M^J^J"
```

Getting Input

Set Timeout *Value*

Value can be from 0 to 65535

When waiting for input using any of the “Getting Input” commands, Spectrum waits for *Value* seconds before it gives up and continues executing the script. If *Value* is 0 Spectrum will wait forever.

If you use a timeout for an input command, test to see if the input “failed” (timed out) before using the input.

Example:

```
Display "What is your favorite letter? "
Set Timeout 5; WaitFor Keyboard
If Failed Then Display "^M^JApparently you can't decide what your favorite is!^M^J"; Stop Script
Display "^M^JYes, '$MatchString' is a nice letter!^M^J"; Stop Script
```

Set KeyLock *State*

State can be Off or On

Locks out keyboard entry that would be sent directly to the port, and prevents keyboard input to the *Get Key* and *Get Line* commands. Turning this option on prevents the user from interfering with the data that a script is transmitting or waiting for.

WaitFor Keyboard

Waits for the next keypress (Spectrum processes characters coming in from the port, but this command ignores them). When a keypress is received the Failed flag is cleared, *$MatchString* is set to the key that was pressed, and *$Matched* is set to 1. You can test for which key was pressed by using the *If Keyboard* command.

If a timeout was used and time ran out, the Failed flag is set, *$MatchString* is cleared, and *$Matched* is set to 0.

WaitFor String "*String1*" "*String2*" … "*String7*" "*String8*"

At least one string is required; up to eight may be specified. Commas are optional between each string.

Spectrum processes characters coming in from the port and watches for the specified string (the CaseSensitive setting affects the comparison). If one of the strings is found then the Failed flag is cleared, *$MatchString* is set to the string that was found, and *$Matched* is a number that indicates which string was matched.

If a timeout was used and time ran out, the Failed flag is set, *$MatchString* is cleared, and *$Matched* is set to 0.

Example:

```
Set Timeout 15; Set CaseSensitive Off
Transmit "ATDT555-1234^M"
WaitFor String "Connect" "Busy" "No Carrier"
If Failed Then Stop Script
On $Matched Goto Logon, Redial, Redial
```

Get Key *VarNum*

Gets one character from the port or the keyboard (if KeyLock is off) then stores it into the specified variable.

If a timeout was used and time ran out, the Failed flag is set and the specified variable is cleared.

Get Line *VarNum*

Accepts up to 128 characters from the port or the keyboard (if KeyLock is Off). When the Return key is pressed the line is stored into the specified variable.

If a timeout was used and time ran out, the Failed flag is set and the specified variable is cleared.

Input Keyboard

Waits for the next keypress. If Flush is Off then characters coming in from the port are not processed; if Flush is On then incoming characters are processed, but this command ignores them (it watches the keyboard only).

When a keypress is received the Failed flag is cleared, *$MatchString* is set to the key that was pressed, and *$Matched* is set to 1. You can test for which key was pressed by using the *If Keyboard* command.

If a timeout was used and time ran out, the Failed flag is set, *$MatchString* is cleared, and *$Matched* is set to 0.

Input Key *VarNum*

Gets one character from the keyboard and stores it into the specified variable. If Flush is Off then characters coming in from the port are not processed; if Flush is On then incoming characters are processed, but this command ignores them (it watches the keyboard only).

If a timeout was used and time ran out, the Failed flag is set and the specified variable is cleared.

Input Line *VarNum*

Accepts up to 128 characters from the keyboard. When the Return key is pressed the line is stored into the specified variable. If Flush is Off then characters coming in from the port are not processed; if Flush is On then incoming characters are processed, but this command ignores them (it watches the keyboard only).

If a timeout was used and time ran out, the Failed flag is set and the specified variable is cleared.

Ask1 "*Question***" "***Button1***"** *VarNum*
Ask2 "*Question***" "***Button1***" "***Button2***"** *VarNum*
Ask3 "*Question***" "***Button1***" "***Button2***" "***Button3***"** *VarNum*

Question is a string up to 68 characters long
Button# is a string up to 12 characters long

Displays an alert window on the super hires screen that contains the question and the buttons. After this statement *VarNum* contains 1, 2, or 3 to indicate which button number was chosen.

The "#" and "*" are special characters in an alert, so to display them you must use them twice (see the example below). *NOTE: No substitution array is defined so using embedded "*" codes will do nothing.*

The strings can include these alert replacement items:

#0 OK	#2 Yes	#4 Try Again	#6 Continue
#1 Cancel	#3 No	#5 Quit	

The beginning of one button string can include "^^" to indicate that button should be the default choice.

Example:

```
Ask3 "Special ## characters **?" "#1 This" "#3 Way" "^^#2!" 0
Display "You chose button number $0.^M^J"
```

Branching and Loops

Set Labels *State*

State can be Off or On

This command controls whether pressing Option-# (where # is a number from 0 to 9) will attempt to “Gosub” that label (the given routine should include a Return or Pop). *NOTE: No error is generated if a numbered label does not exist. The label search begins at the top of the script each time, so control passes to the first matching label.*

Goto *Label*

Script control jumps to the commands after the first occurrence of *Label* (the label search begins at the top of the script).

Example:

```
Goto Begin
Display "This will never be displayed!"
# Begin
Display "Hello!^M^J"; Stop Script
```

GotoNext *Label*

Script control jumps to the commands after the next occurrence of *Label* (the label search begins on the line after the current statement).

Because searching does not begin at the top of the script, GotoNext is slightly faster than Goto. It is also useful in creating “modular” subroutines…if the subroutine is structured to use GotoNext instead of Goto, you can cut and paste the subroutine without fear of any label names conflicting with labels in the script you’re pasting into.

Gosub *Label*

Script control temporarily transfers to the commands after the first occurrence of *Label* (the label search begins at the top of the script). When the *Return* command is encountered, script control returns to the statement following the *Gosub* command.

Gosub is short for “go subroutine.” Subroutines are very useful (but not required). Imagine you were writing a script that needed to use the same ten commands three different places in a script. Instead of writing those ten commands each time, write the ten commands only once as a “subroutine.” Whenever those ten commands are needed you simply “gosub” to that subroutine.

Up to 16 *Gosub* commands can be active, which means one “subroutine” can call another.

Example without a subroutine:

```
Set Var 6 "This is an example"; Display "Variable 6 is '$6'.^M^J"; Display "The length is
   $Length6.^M^J"
Set Var 6 "of how subroutines"; Display "Variable 6 is '$6'.^M^J"; Display "The length is
   $Length6.^M^J"
Set Var 6 "can be useful."; Display "Variable 6 is '$6'.^M^J"; Display "The length is
   $Length6.^M^J"
```

Example with a subroutine:

```
Set Var 6 "This is an example"; Gosub ShowVar6
Set Var 6 "of how subroutines"; Gosub ShowVar6
Set Var 6 "can be useful."; Gosub ShowVar6

# ShowVar6
Display "Variable 6 is '$6'.^M^J"; Display "The length is $Length6.^M^J"
Return
```

Although the second script might look longer, it actually contains 100 characters less than the first script! Also, imagine what would happen if you wanted to know whether or not variable 6 contained the letter “a”…instead of adding commands to three separate locations you simply have to add this to the subroutine (just above the *Return* command):

```
Position "a" "$6" 1 1; If Not Equal "$1" "0" Then Display "There is at least one 'a' in that
  phrase.^M^J"
```

GosubNext *Label*

Script control temporarily transfers to the commands after the next occurrence of *Label* (the label search begins on the line after the current statement). When the *Return* command is encountered, script control returns to the statement following the *Gosub* command. Because searching does not begin at the top of the script, *GosubNext* is slightly faster than *Gosub*.

Return

When the *Gosub* or *GosubNext* command is encountered, Spectrum remembers where the statement is. When the *Return* command is encountered, Spectrum returns control to that point.

Example:

```
Gosub th; Display "ree "; Gosub th; Display "in "; Gosub th; Display "ings!"; Stop Script
# th
Display "Th"; Return
```

Pop

When the *Gosub* or *GosubNext* command is encountered, Spectrum remembers where the statement is. Using the *Pop* command makes Spectrum forget the most recent Gosub, which causes the next *Return* command to return control to the statement following the *second* most recent Gosub. *NOTE: This can be confusing and is intended for advanced script authors only*.

Example:

```
Gosub One; Display "Finished.^M^J"; Stop Script
# One
Display "Inside ONE..."; Display "Leaving ONE..."; Gosub Two; Display "Back in ONE..."; Return
# Two
```

```
Display "Inside TWO..."; Display "Leaving TWO..."; Gosub Three; Display "Back in TWO..."; Return
# Three
Display "Inside THREE..."
Pop # this forgets about "Gosub Three"
Pop # this forgets about "Gosub Two"
Return # which means this returns to the statement after "Gosub One"
```

Pop All

Similar to the *Pop* command, except *Pop All* makes Spectrum forget all *Gosub* commands. Most often useful in a generic error handler routine. *NOTE: This can be confusing and is intended for advanced script authors only.*

On *Value* Goto *Label1*, *Label2 … Label7*, *Label8*

At least one label is required; up to eight may be specified

Performs a Goto based on the *Value* (e.g. if *Value* is 1 then script control passes to the first label; if *Value* is 5 then script control passes to the fifth *Label*). *NOTE: The label search begins at the top of the script each time, so control passes to the first matching label.*

If *Value* is 0 then script control “falls through” to the next statement and the Failed flag is set.

Example:

```
Set Timeout 10; WaitFor String "CONNECT" "BUSY"
If Failed Goto Abort
On $Matched Goto Logon, Redial
```

On *Value* GotoNext *Label1*, *Label2 … Label7*, *Label8*

At least one label is required; up to eight may be specified

Similar to *On…Goto* except that the label search begins at the current statement, so control passes to the first matching label after this statement.

On *Value* **Gosub** *Label1*, *Label2 … Label7*, *Label8*

At least one label is required; up to eight may be specified

Performs a Gosub based on the value (e.g. if *Value* is 1 then script control passes to the first label; if *Value* is 5 then script control passes to the fifth label). When a *Return* command is encountered control returns to the statement after the *On value Gosub* command. *NOTE: The label search begins at the top of the script each time, so control passes to the first matching label.*

If *Value* is 0 then script control "falls through" to the next statement and the Failed flag is set.

Up to 16 *Gosub* commands can be active, which means one "subroutine" *can* call another.

Example:

```
# Options
Display "Type a number from 1-3: "; WaitFor Keyboard
On $MatchString Gosub One, Two, Three; Goto Options
# One
Display "One^M^J"; Return
# Two
Display "Two^M^J"; Return
# Three
Display "Three^M^J"; Return
```

On *Value* **GosubNext** *Label1*, *Label2 … Label7*, *Label8*

At least one label is required; up to eight may be specified

Similar to On…Gosub except that the label search begins at the current statement, so control passes to the first matching label after this statement.

For *LoopNumber Start Stop Increment*

LoopNumber can be from 0 to 9
Start is a value from 0 to 65535
Stop is a value from 0 to 65535
Increment is optional; if used it is a value from 1 to 65535

The “For” loop lets you easily repeat a sequence of statements a number of times. The loop is controlled by a counter. The *For* command initializes the counter to *Start*, and the counter is incremented each time a *Next* command is encountered. When the counter passes *Stop* the loop stops. Unless specified, *Increment* is 1. The current counter value can be determined by using *$ForValue#* (where # is a number from 0 to 9).

Example:

```
Display "^LFor/Next loops are much faster than loops done manually!^M^J^J"
Display "Counting with a manual loop: "; Store XY
Set Var 0 "1"
# ManualLoop
Restore XY; Display "$0^M^J"
If Not Equal "$0" "100" Then Inc 0; Goto ManualLoop
Display "Counting with a For/Next loop: "; Store XY
For 0 1 100
Restore XY; Display "$ForValue0^M^J"
Next 0
Display "Done!^M^J"
```

Next *LoopNumber*

LoopNumber can be from 0 to 9

Tests to see if the loop is finished. If not, control returns to the statement after the *For* command. Otherwise control continues to the statement immediately after the *Next* command.

Clear For *LoopNumber*

LoopNumber can be from 0 to 9

Clears loop number *LoopNumber* and sets the counter value to 0. If a *Next* command is encountered, control continues to the statement immediately after the *Next* command.

Store ForLoops

Remembers the current state of all the *For* loops.

Restore ForLoops

Restores the saved state of all the *For* loops.

Conditional Tests

If Equal "*String1*" "*String2*" Then *Statement*

If *String1* and *String2* are exactly the same then *Statement* is executed, otherwise control drops to the next line.

Example:

```
If Equal "THIS" "this" Then Display "The CaseSensitive option is OFF.^M^J"; Stop Script
Display "The CaseSensitive option is ON.^M^J"
```

If Not Equal "*String1*" "*String2*" Then *Statement*

If *String1* and *String2* are **not** exactly the same then *Statement* is executed, otherwise control drops to the next line.

Example:

```
Display "What is your password? "; Get Line 0
If Not Equal "$0" "Garfblat" Then Display "Wrong password!^G^M^J"; Stop Script
Display "Correct!^M^J"
```

If Contains "*LongString*" "*ShortString*" Then *Statement*

If *ShortString* is found anywhere in *LongString* then *Statement* is executed, otherwise control drops to the next line.

Example:

```
If Contains "This is a test" "this" Then Display "YES"; Stop Script
Display "NO"
```

If Not Contains "*LongString*" "*ShortString*" Then *Statement*

If *ShortString* is **not** found anywhere in *LongString* then *Statement* is executed, otherwise control drops to the next line.

Example:

```
# Ask
Display "Do you want to continue? "; WaitFor Keyboard
If Not Contains "YN" "$MatchString" Then Display "^M^JTry Again!^M^J"; Goto Ask
```

If Keyboard *Character* Then *Statement*

If the key pressed in the most recent *WaitFor Keyboard* command is the same as *Character* then *Statement* is executed, otherwise control drops to the next line.

Example:

```
# Continue
Display "Do you want to continue? "; WaitFor Keyboard
If Keyboard Y Then Return
If Keyboard N Then Pop; Display "^M^JGoodbye!^M^J"; Stop Script
Display "^G"; Goto Continue
```

If Not Keyboard *Character* Then *Statement*

If the key pressed in the most recent *WaitFor Keyboard* command is **not** the same as *Character* then *Statement* is executed, otherwise control drops to the next line.

Example:

```
# Wait
Display "Press the Spacebar to continue."; WaitFor Keyboard
If Not Keyboard " " Then Display "^G"; Goto Wait
```

If Exists "*FoldernameFilename*" Then *Statement*

If the specified file exists then *Statement* is executed, otherwise control drops to the next line.

Typically you would check to see if a file exists before you attempt to delete or work with the file, thus avoiding errors.

Example:

```
If Exists "$SpectrumPathCapture.File" Then Show File "$SpectrumPathCapture.File"; Goto NextThing
Display "^GThere is no capture file to examine!^M^J"
# NextThing
Display "That's all for now!^M^J"; Stop Script
```

If Not Exists "*FoldernameFilename*" Then *Statement*

If the specified file does **not** exist then *Statement* is executed, otherwise control drops to the next line.

Typically you would make sure a file does not exist so you won't accidentally overwrite an important file.

Example:

```
Gosub GetCaptureName; Display "Capturing to $0..."; Stop Script
# GetCaptureName
Display "Capture text to what filename? "; Get Line 0
If Not Exists "$SpectrumPath$0" Then Return
Display "^M^J^GThat name is already used--"; Goto GetCaptureName
# In reality you should use the Get File command to let the user specify a name and location of
   the file.
```

If Failed Then *Statement*

If the Failed flag is set then *Statement* is executed, otherwise control drops to the next line. *NOTE: "If Failed" is the same as "If Not Found".*

The Failed flag is set or cleared by various commands (e.g. *Dial*, *WaitFor*, etc.). The *If Failed* command can be used to determine if a command failed. *NOTE: The Failed flag is correct only following a command that sets or clears it…test the flag immediately after these commands (do not put statements between the command and the "If Failed" test).*

Example:

```
# Ask
```

```
    Display "^M^JPress a key to continue..."
    Set Timeout 5; WaitFor Keyboard
    If Failed Then Display "Hurry up!^G^M^J"; Goto Ask
    Return
```

If Not Failed Then *Statement*

If the Failed flag is clear then *Statement* is executed, otherwise control drops to the next line. *NOTE: "If Not Failed" is the same as "If Found".*

The Failed flag is set or cleared by various commands (e.g. *Dial*, *WaitFor*, etc.). The *If Not Failed* command can be used to determine if a command was successful. *NOTE: The Failed flag is correct only following a command that sets or clears it…test the flag immediately after these commands (do not put statements between the command and the "If Not Failed" test).*

Example:

```
    Dial String "555-1234"
    If Not Failed Then Goto Connected
    Display "The dial command failed.^M^J"; Stop Script
```

If Found Then *Statement*

If the Found flag is set then *Statement* is executed, otherwise control drops to the next line. *NOTE: "If Found" is the same as "If Not Failed".*

The Found flag is set or cleared only by the *WaitFor* commands that use a Timeout (Found is set if the *WaitFor* command succeeds).

Example:

```
    # Ask
    Display "^M^JPress a key to continue..."
    Set Timeout 5; WaitFor Keyboard
    If Found Then Return
    Display "Hurry up!^G^M^J"; Goto Ask
```

If Not Found Then *Statement*

If the Found flag is clear then *Statement* is executed, otherwise control drops to the next line. *NOTE: "If Not Found" is the same as "If Failed".*

The Found flag is set or cleared only by the *WaitFor* commands that use a Timeout (Found is cleared if the *WaitFor* command times out).

Example:

```
    # Ask
    Display "^M^JPress a key to continue..."
    Set Timeout 5; WaitFor Keyboard
    If Not Found Then Display "Hurry up!^G^M^J"; Goto Ask
    Return
```

If Null *VarNum* Then *Statement*

If the given variable number is empty ("") then *Statement* is executed, otherwise control drops to the next line.

Example:

```
    Display "What is your name? "; Get Line 0
    If Null 0 Then Display "We'll call you 'Fred' because you didn't type anything!^M^J"; Set
      Variable 0 "Fred"
    Display "Hello there, $0!^M^J"
```

If Not Null *VarNum* Then *Statement*

If the given variable number is **not** empty then *Statement* is executed, otherwise control drops to the next line.

Example:
```
    Display "What is your name? "; Get Line 0
    If Not Null 0 Then Display "Hello there, $0!^M^J"
```

If Even *VarNum* **Then** *Statement*

If the value of the given variable number is an even number then *Statement* is executed, otherwise control drops to the next line. *NOTE: "If Even" is the same as "If Not Odd".*

If Not Even *VarNum* **Then** *Statement*

If the value of the given variable number is not an even number then *Statement* is executed, otherwise control drops to the next line. *NOTE: "If Not Even" is the same as "If Odd".*

If Odd *VarNum* **Then** *Statement*

If the value of the given variable number is an odd number then *Statement* is executed, otherwise control drops to the next line. *NOTE: "If Odd" is the same as "If Not Even".*

If Not Odd *VarNum* **Then** *Statement*

If the value of the given variable number is not an odd number then *Statement* is executed, otherwise control drops to the next line. *NOTE: "If Not Odd" is the same as "If Even".*

If CarrierDetect Then *Statement*

NOTE: This command is reliable only if your modem properly controls the DCD signal, and you have a properly-wired modem cable, and the "DCD Handshake" option is on.

If the modem is currently connected to a host then *Statement* is executed.

If Not CarrierDetect Then *Statement*

NOTE: This command is reliable only if your modem properly controls the DCD signal, and you have a properly-wired modem cable, and the "DCD Handshake" option is on.

If the modem is not currently connected to a host then *Statement* is executed.

Example:

```
# - Run this script when you are NOT online
Transmit "AT&C1^M" # - a common modem command so the modem will adjust the DCD line to indicate
  whether a remote modem's data carrier tone is present
Set DCD On # - set Spectrum so it trusts the DCD signal
Display "Your setup apparently does "
If CarrierDetect Then Display "NOT "
Display "support DCD.^M^J"
```

If Debug Then *Statement*

If debugging is currently "Screen" or "Scrollback," then *Statement* is executed.

If Not Debug Then *Statement*

If debugging is currently off then *Statement* is executed.

If TheManager Then *Statement*

If Spectrum is currently being run under *The Manager* (Seven Hills Software's multi-tasking environment for the Apple IIGS) then *Statement* is executed.

If Not TheManager Then *Statement*

If Spectrum is not currently being run under *The Manager* then *Statement* is executed.

Screen Appearance

Set ChatLine *State*

State can be Off or On

Controls whether the chat line is visible or not. *NOTE: Some online displays do not support a chat line.*

Set AutoChat *State*

State can be Off or On

Normally when the chat line is turned on, the port is automatically set to full duplex. When the chat line is turned off the original duplex setting is restored. If you turn AutoChat off, Spectrum will not automatically change the duplex setting when the chat line is turned on and off.

Set StatLine *State*

State can be Off or On

Controls whether the status line is visible or not. *NOTE: Some online displays do not support a status line.*

Store XY

Stores the current cursor position and sets the values for *$StoredX* and *$StoredY*.

Restore XY

Restores the cursor position that was last stored using the *Store XY* command.

GotoXY *X*,*Y*

X (horizontal position) is a value from 0 to 79
Y (vertical position) is a value from 0 to 23

Attempts to move the cursor to the specified screen position. The limits for each value depend upon the online display being used. Be aware that the setting of the chat line and status line might also affect the range of acceptable values.

Draw Window *Left Right Top Bottom*

Left and *Right* are values from 0 to 79
Top and *Bottom* are values from 0 to 23

Draws a window extending from (*Left*, *Top*) to (*Right*, *Bottom*). The limits for each value depend upon the online display being used. Be aware that the setting of the chat line and status line might also affect the range of acceptable values. A script error occurs if a value is used that would place a window coordinate off the screen.

After drawing the window you can use the *GotoXY* and *Display* commands to display something within the window. *NOTE: The window is simply a visual effect—text you display can easily overwrite a window's frame.*

Print Screen

Prints the current screen to the printer, just as if Shift-4 was pressed.

Save Screen

Saves the current screen to disk, just as if Shift-3 was pressed.

Set Flush *State*

State can be Off or On

Incoming data is stored in the port buffer until Spectrum has time to deal with it. When Flush is On, data in the port buffer is processed constantly. When Off, data in the port buffer is processed only during *Get Key*, *Get Line*, *WaitFor Keyboard*, and *WaitFor String* commands (or when the script stops), which makes those commands more reliable.

Set Screen *State*

State can be Off or On

This command determines whether incoming or outgoing data is displayed on the screen when a script is running (incoming data will still be captured if the capture buffer is turned on, and it will still be seen by the *WaitFor*, *Get Line* , and *Get Key* commands). Commands that directly display to the screen, such *Display*, will continue to display on the screen.

Set ScreenBlank *State*

State can be Off, On, or Auto

Controls a built-in screen blanker that blanks the screen (except for the border color). The screen is blanked only while a script is being run…when the script stops the screen is unblanked. *NOTE: If the Twilight II screen blanker is active, Spectrum asks it to "background blank" the screen.*

Off: The screen is not blanked.

On: Blanks the screen immediately.

Auto: Blanks the screen only during *WaitFor* commands. In the example script below, the screen will be blanked until 7pm, at which time it will unblank (in case you happen to want to watch the online session), dial a service, send and receive mail, then log off. The final *WaitFor* command will blank the screen until you press a key.

Example:

```
    Set ScreenBlank Auto
    WaitFor Time "19:00" # wait until 7pm
    Dial Service "CompuServe" # dial CompuServe
    If Not Failed Then Gosub Login; Gosub SendMail; Gosub ReadMail; Hangup
    WaitFor Keyboard; Stop Script # wait until a key is pressed
```

Prefix Control

Set SFPrefix "*Foldername*"

Sets GS/OS prefix 0 and prefix 8 to the specified folder.

Set GSPrefix *Value* "*Foldername*"

Value can be 0, 2 through 8, or 10 through 31

Sets the specified GS/OS prefix number to the specified folder. Prefix numbers, followed by a colon, can be used as shortcuts wherever a *Foldername* is required. For example, *Show Catalog "8:"* will list the files stored in the prefix 8 folder.

Prefix 1 and 9 is the folder where Spectrum is located and therefore it cannot be changed. All the other prefixes are not used by Spectrum so they are available to scripts.

Prefix 0 is a working path; NDAs and applications change this frequently. Prefix 8 is the current prefix. The next time an “Open” or “Save” dialog box appears, the prefix 8 folder will appear. Prefix 0 and prefix 8 can be changed independently, but usually they are set together (use the *Set SFPrefix* command as a shortcut).

Prefix 0 and 8 are volatile—they can be changed by desk accessories, loading a file in the Spectrum editor, and so on. Therefore it is better to use higher numbered prefixes to remember a prefix for a long time.

Example:

```
Set GSPrefix 21 "$BootSystem" # The system folder on the disk we booted from (e.g. :Hard:System)
Set GSPrefix 22 "21:Desk.Accs" # this is identical to typing "$BootSystem:Desk.Accs"
Set GSPrefix 23 "21:Fonts"
Display "Your Desk Accessories:^M^J"; Show Catalog "22:"
Display "Your Fonts:^M^J"; Show Catalog "23:"
```

Capture Buffer Control

Set Buffer *State*

State can be Off, On, Auto, or Manual

Off/On: Determines whether or not incoming characters are saved into the capture buffer or the capture file (whichever is active). *NOTE: Script commands that record directly to the buffer will do so regardless of this setting.*

Auto/Manual: Controls the “Auto buffer control” checkbox in the Online Displays Settings dialog box. When this option is set to Auto, the host can turn your capture buffer on and off by sending a ^R or ^T, respectively. *NOTE: Auto buffer control works only if the capture buffer has been turned off.*

Clear Buffer

Clears the capture buffer so it contains no characters.

Open CaptureFile "*FoldernameFilename*"

Deletes the specified file if it exists, then creates a new file and begins capturing incoming data to it instead of to memory.

Append CaptureFile "*FoldernameFilename*"

Opens the specified text file and begins capturing incoming data to the end of it.

Close CaptureFile

Closes the capture file and resumes storing incoming data into memory.

Set Append *State*

State can be Off or On

Determines whether data is appended to the end of an existing file or whether data overwrites any existing file when using the *Save Buffer* or *Write Buffer* commands (see those commands for more information).

Set AutoSave "*FoldernameFilename***"**

Filename must be at least three characters long

Specifies the file to use when saving the capture buffer using the *Save Buffer* command. This is used as a shortcut…you can establish this AutoSave filename and whenever you need to save the capture buffer just issue a *Save Buffer* command (as opposed to using *Write Buffer "FoldernameFilename"* each time you need to save the capture buffer).

Set AutoSaveBuffer *State*

State can be Off or On

Determines what occurs when the capture buffer completely fills. When on, Spectrum automatically issues a *Save Buffer* command; when off the user is presented with a dialog box from which he can clear or save the capture buffer.

Save Buffer

If the Append flag is off, the AutoSave filename is incremented by 1 and a new file is created. *NOTE: If there is no number as the last character of the filename then a "1" is inserted (if a number is there it gets incremented). When the number at the last position gets to 9 then the second to last character is set to 1 or increments if it's already a number. The maximum is 99 files, at which point you get an error "name no longer valid."*

If the Append flag is on, the AutoSave filename is not changed and the capture buffer contents are appended to the end of the existing AutoSave filename (if the file doesn't exist yet it is created).

After saving the contents of the capture buffer, the buffer is cleared.

Write Buffer "*FoldernameFilename*"

If the Append flag is off, the specified file is deleted if it already exists, then the capture buffer contents are saved.

If the Append flag is on, if the file exists it is *not* deleted (if *Filename* does not exist it is created). The capture buffer contents are appended to the end of the existing file.

After writing the contents of the capture buffer, the buffer is cleared.

Load Buffer "*FoldernameFilename*"

Loads the file into the capture buffer.

Transferring Files

Send File "*FoldernameFilename*" *Protocol*

Protocol can be Text, ProDOS, Xmodem, 1KXmodem, 4KXmodem, BPlus, Ymodem, or Zmodem

Sends the specified file using the specified protocol. A file transfer dialog box appears for all protocols except possibly Text (which depends upon the ULTextShow setting).

See the “File Transfer Settings” section for settings that apply to file transfers. For example, the “Turbo” option determines whether regular Ymodem or Ymodem-g is used.

If the transfer fails the Failed flag is set, otherwise it is clear.

Receive File "*FoldernameFilename*"

Spectrum automatically detects which protocol is being used, then it receives a file via Xmodem, 1K Xmodem, 4K Xmodem, Ymodem, or Ymodem-g and saves it as *FoldernameFilename*.

If the transfer fails the Failed flag is set, otherwise it is clear.

Receive File *Protocol*

Protocol can be BPlus or Zmodem

Receives a file using the specified protocol and saves it into the current file transfer folder *($FileXferPath)*. The filename is not needed because it is provided by the incoming data.

When receiving a Zmodem file the same process is used that is used for an “auto receive” file, except that the script can control how long Spectrum waits for the transfer to start. Set the wait time using *Set Timeout* with a value from 10 to 600 (10 seconds to 10 minutes).

If the transfer fails the Failed flag is set, otherwise it is clear.

OS Utilities

Delete File "*FoldernameFilename***"**

Permanently deletes the specified file. *Use with caution!*

Example:

```
    Set Var 0 "$ScriptPathTemporary.File"
    If Exists "$0" Then Delete "$0"
```

Rename File "*Foldername1Filename1***" "***Foldername2Filename2***"**

Renames the first item to the second name. The folder names can either refer to the same folder (which just renames *Filename*), or they can refer to different folders on the same disk (which moves *Filename* into the second folder). For example…

```
    Rename ":Hard:Spectrum:Capture.File" ":Hard:Archives:Capture"
```

…renames “Capture.File” to be called “Capture” *and* it moves it from the Spectrum folder into the Archives folder (which must already exist on the same disk).

Copy File "*Foldername1Filename1***" "***Foldername2Filename2***"**

Makes an exact copy of the first file. The copy is named *Filename2* and is stored in the *Foldername2* folder.

Create Folder "*FoldernameFilename***"**

Creates a new folder named *Filename* in the *Foldername* folder.

Get FileSize "*FoldernameFilename*" *VarNum*

The number of bytes in the specified file’s data and resource forks.

Get VolumeSize "*Volumename*" *VarNum*

The total size of the specified disk, in bytes.

Get VolumeFree "*Volumename*" *VarNum*

The number of bytes of free space on the specified disk.

Get FileInfo "*FoldernameFilename*" *VarNum*

Gets information about the specified file. The information is in the following format (use the *Substring* command to extract the desired pieces of information):

Start	Length	Information
1	31	Filename
33	3	3-letter abbreviation for common filetypes or $00
37	8	Length (data fork plus resource fork)
46	1	D=Can Destroy (blank space if not)
47	1	N=Can Rename (blank space if not)
48	1	B=Needs Backup (blank space if not)
49	1	W=Can Write (blank space if not)
50	1	R=Can Read (blank space if not)
52	2	Hexadecimal filetype
55	4	Hexadecimal auxtype

Example:

```
    Get FileInfo "$ScriptPath$ScriptFile" 0
    Substring "$0" 1 31 1 # $1 now has the name
    Substring "$0" 33 3 2 # $2 now has the type
    Substring "$0" 46 5 3 # $3 now has the file flags
    Display "Name: $1^M^JType: $2^M^JInfo: $3^M^J=====^M^J"
```

Show Catalog "*Foldername*"

Shows a listing of the items stored in the specified folder. If you also want to send the listing to the port, Set Echo On before showing it.

ShowRecord Catalog "*Foldername*"

A listing of the items stored in the specified folder is shown on the screen and recorded to the capture buffer. If you also want to send the listing to the port, Set Echo On before showing it.

Record Catalog "*Foldername*"

A listing of the items stored in the specified folder is recorded into the capture buffer.

Show File "*FoldernameFilename*"

Shows the specified AppleWorks Classic, Teach, or Text file on the screen. If you also want to send the listing to the port, Set Echo On before showing it.

Get File "*PromptString*" *Kind VarNum*

Kind can be 0 (any file), 1 (text only), or 2 (text, AppleWorks, or Teach)

Presents the standard “Open” file dialog box in which the user can select a file. *PromptString* is shown at the top of the dialog box (e.g. “Select the file to rename”). Only files matching the given *Kind* will be shown. *NOTE: If the script can be run unattended, do not use this command because it requires the user to interact with it (there is no “timeout”).*

If the user cancels the dialog box the Failed flag is set and the specified variable number is cleared to "". If the user did not cancel the dialog box, the specified variable number contains the name of the file that was highlighted and the prefix is set to the folder that contains the file *($SFPrefix)*.

This command does not open or load a file; it merely provides a standard way for the user to select a file.

Example:

```
Get File "Select the file to format:" 2 0 # select a text, AppleWorks, or Teach file
If Failed Then Stop Script # user clicked Cancel
Load ScriptEditor "$0" # load the file into the script editor
Apply LowASCII; Apply RemoveControls; Apply LFsToCRs; Apply Format 3 # format message for posting
Save ScriptEditor "$0" # save the formatted file
```

Put File "*PromptString*" "*NameString*" *VarNum*

NameString is optional; if not used “Untitled” is used

Presents the standard “Save” file dialog box in which the user specifies a filename and location to store a file. *PromptString* is shown above the name (e.g. “Save the file as…”). *NameString* is the suggested name that will appear in the dialog box. *NOTE: If the script can be run unattended, do not use this command because it requires the user to interact with it (there is no “timeout”).*

If the user cancels the dialog box the Failed flag is set and the specified variable number is cleared to "". If the user did not cancel, the specified variable number contains the name of the file they typed, and *$SFPrefix* indicates the folder they want the file stored in.

This command does not create a file; it merely provides a standard way for the user to specify a filename and location. If the command is successful you can be assured the returned *Filename* is a legal name.

The only special condition to be aware of is if the user specifies the name of a file that already exists on disk. In this case the system has already received permission to replace the existing file, but it has not deleted it. To avoid errors you should follow a successful *Put File* command with a statement to delete the file if it exists (see example).

Example:

```
Put File "Save mail as..." "Untitled" 0
If Failed Then Stop Script # user clicked Cancel
If Exists "$0" Then Delete "$0" # if the file already exists, delete it (the user has already
   given permission)
Write Buffer "$0" # save the capture buffer using the filename the user specified
```

Reading and Writing Files

Up to four files (numbered 0, 1, 2, and 3) may be open at a single time. After a file is opened for reading or writing (using either the *Open* or *Append* command) refer to the file using the *FileNumber*.

Open File *FileNumber* **"***FoldernameFilename***"**

FileNumber can be from 0 to 3

Opens the specified text file for reading or writing as file number *FileNumber*. If the file does not exist it is created. *NOTE: This command will also open Teach files, but only for reading (i.e. using Write File will cause an error).*

If you want to write a file from scratch you should delete it first (if it exists) because writing to a file does not shorten the file's length. For example, if a text file contains 100 characters and you open it, write 15 characters, then close it, the file will still contain 100 characters…15 new characters followed by the 85 old ones.

Append File *FileNumber* **"***FoldernameFilename***"**

FileNumber can be from 0 to 3

Opens the specified text file as file number *FileNumber* and sets the file marker to the end of the file so that writing will occur at the end of the file.

Read File *FileNumber* *VarNum*

FileNumber can be from 0 to 3

Reads the file and stores the read characters into variable number *VarNum*. Reading stops when 128 characters are read, when a CR is encountered, or when the end of the file is encountered.

If you attempt to read past the end of a file the Failed flag is set and variable number *VarNum* is set to "".

Write File *FileNumber* **"***String***"**

FileNumber can be from 0 to 3

Writes the string into the open file at the current position.

Close File *FileNumber*

FileNumber is optional; if used it can be from 0 to 3

Closes the specified file. If no *FileNumber* is used then all four files are closed (if open). If you forget to close a file you open, it will be closed automatically when the script stops.

Reading Catalogs

Up to four catalogs (numbered 0, 1, 2, and 3) may be opened at a single time. After a catalog is opened you refer to it using the *CatalogNumber*.

Open Catalog *CatalogNumber* **"***Foldername***"**

CatalogNumber can be from 0 to 3

Opens the specified *Foldername* for reading. *REMEMBER: A Foldername can be just a volume name, or a volume name plus one or more folder names.*

Read Catalog *CatalogNumber VarNum*

CatalogNumber can be from 0 to 3

Reads one catalog entry and stores information about it into variable number *VarNum*. If you attempt to read past the end of a catalog the Failed flag is set and variable number *VarNum* is set to "".

Each entry is in the format described under the *Get FileInfo* command.

Close Catalog *CatalogNumber*

CatalogNumber is optional; if used it can be from 0 to 3

Closes the specified catalog. If no *CatalogNumber* is used then all four catalogs are closed (if open). If you forget to close a catalog you open, it will be closed automatically when the script stops.

Script Editor

There is a text editor available just for scripts to use; it is entirely separate from the built-in text editor. The script editor can be used to load a file, apply formats to it, and send or save the file.

The script editor commands are ideal for automatically formatting a message for posting, or for automatically formatting incoming messages to be more readable.

Load ScriptEditor "*Item*"

Item can be a *FoldernameFilename*, or can be ::Scrollback, ::Buffer, or ::Clipboard

Loads the specified item into the script editor, replacing any existing script editor in memory. A script error occurs if the specified item could not be loaded.

If *Item* specifies a file on disk, the file must be a text, Teach, or AppleWorks Classic file.

If *Item* is *::Scrollback*, then contents of the current scrollback buffer are copied into the script editor. Likewise, *::Buffer* copies the current capture buffer contents and *::Clipboard* copies the system clipboard contents. *NOTE: As a shortcut you can also use ::S, ::B, or ::C (everything after the first letter is ignored).*

Append ScriptEditor "*Item*"

Item can be a *FoldernameFilename*, or can be ::Scrollback, ::Buffer, or ::Clipboard

Appends the specified item to the end of the current script editor. A script error occurs if the specified item could not be appended.

If *Item* specifies a file on disk, the file must be a text, Teach, or AppleWorks Classic file.

If *Item* is *::Scrollback*, then contents of the current scrollback buffer are appended onto the script editor. Likewise, *::Buffer* appends the current capture buffer contents and *::Clipboard* appends the system clipboard contents. *NOTE: As a shortcut you can also use ::S, ::B, or ::C (everything after the first letter is ignored).*

Save ScriptEditor "*FoldernameFilename*"

If a file has previously been loaded into memory, this command writes the file from memory to disk. If the specified file already exists on disk it is deleted before the new text file is written. *NOTE: Save ScriptEditor always creates a <u>text</u> file; if you load a Teach file into the script editor then save it with the same name, you will be deleting the Teach file and replacing it with a text file.*

Saving the ScriptEditor does *not* clear the file from memory, which means you can load a file, apply a format, save that version, apply another format, save that version, and so on. When you are done working with a file you should use the *Clear ScriptEditor* command to erase the file from memory.

Clear ScriptEditor

Clears the ScriptEditor text from memory, thus making the memory available for other uses. You should always clear the script editor when you are done using it. If you forget, the script editor will be cleared automatically when the script stops.

Send ScriptEditor

Sends the contents of the script editor via Text protocol. A file transfer dialog might appear (depends upon the “ULTextShow” setting). See the “File Transfer Settings” section for settings that apply to file transfers.

If the transfer fails the Failed flag is set, otherwise it is clear.

Apply Replace "*FindString*" "*ReplaceString*" *VarNum*
VarNum is optional

Replaces all occurrences of *FindString* with *ReplaceString*, and stores the number of changes made into variable number *VarNum*. The CaseSensitive option affects the search. *NOTE: Because Apply Replace can take a long time to complete, consider displaying a message to let the user know that your script is working.*

You can search for control characters by using a caret (^) and the letter of the control character (e.g. use ^M to find all carriage returns). To search for an actual caret character, use ^^.

Apply LowASCII

Converts the loaded text to low ASCII by stripping off the high bit.

Apply RemoveControls

Removes all control characters except for tabs, carriage returns, and linefeeds. If a backspace character (^H, ASCII $08) is removed, the one preceding character is also removed.

Apply LFsToCRs

Changes all carriage return/linefeed combinations into a carriage return, and changes standalone linefeed characters into carriage return characters.

Apply RemoveSpaces

Replaces two or more consecutive space characters with only a single space, and removes spaces before a carriage return

Apply Special *Value*
Value can be from 1-5

Applies one of the following special formats:

Value	Special Format
1	all lower case
2	ALL UPPER CASE
3	All Proper Names
4	Capitalize sentences
5	Convert Viewdata to Text

Apply Format *Value*

Value can be from 1-4

Applies one of the following formats:

Value	Format
1	CRs into spaces
2	Add line feeds
3	Lines into paragraphs
4	Paragraphs into lines

Example:

```
Load ScriptEditor "Message"; Display "Formatting..."
Apply Replace "…" "..." 1 # option-semicolon to ...
Apply Replace "—" "--" 1 # shift-option-dash to --
Apply LowASCII; Apply RemoveControls; Apply LFsToCRs; Apple Format 4
Display "done!^M^J"
Save ScriptEditor "Message"
Clear ScriptEditor
```

Error Control

On Escape Goto *Label*

If the user presses Escape while a script is running, Spectrum normally cancels the script with an error message stating that the script has been stopped. Some script authors might want to exit more gracefully if the user presses Escape, or perhaps confirm that the user really wants to stop the script.

If the *On Escape Goto* command has been encountered, instead of cancelling the script Spectrum jumps to the given label. If the label is not found then the script is cancelled in the usual way.

Although you can use *Resume* to continue the script from the point where the user pressed Escape, keep in mind that if he has pressed Escape then he probably wants to stop the script. Therefore the commands located at *Label* should let the user exit the script.

To turn off the *On Escape Goto* command, use an empty string ("") for the label.

Example:

```
On Escape Goto Quit # Also try running without this line to see what happens when you press ESC
# Loop
Display "Press ESC to quit..."; Goto Loop
# Quit
On Escape Goto ""
Display "^M^J^JThanks for using this script!^M^J"
Stop Script
```

On Escape GotoNext *Label*

Similar to the *On Escape Goto* command, except that Spectrum jumps to the next occurrence of *Label* (the search does not begin at the top of the script). This is useful for creating “local” Escape handlers.

On Error Goto *Label*

If a script error occurs while a script is running, Spectrum normally cancels the script and displays an error message. Some script authors might want to catch certain errors or exit more gracefully if an error occurs.

If the *On Error Goto* command has been encountered and an error occurs, Spectrum jumps to the given label instead of cancelling the script. If the label is not found then the script is cancelled in the usual way.

If you use the *On Error Goto* command, keep in mind that if a script error has occurred then the script probably should be stopped, so the commands located at *Label* should clean things up and exit the script. To report the error to the user a script can use the *$ErrorMsg* replacement item or the *Show Error* command.

To turn off the *On Error Goto* command, use an empty string ("") for the label.

Example:

```
# Loop
Display "Perform what script command? "; Get Line 3; If Null 3 Then Stop Script
On Error Goto BadCommand # turn on error checking
$0 # execute the command that was typed
On Error Goto "" # turn off error checking
Goto Loop
# BadCommand
Display "^M^J^GSpectrum doesn't recognize the command '$0'.^M^J^J"
Goto Loop
```

On Error GotoNext *Label*

Similar to the *On Error Goto* command, except that Spectrum jumps to the next occurrence of *Label* (the search does not begin at the top of the script). This is useful for creating “local” Error handlers.

Resume

Use only in an “On Escape Goto” or “On Error Goto” procedure

Continues running the script as if Escape was not pressed or as if the error did not occur. *NOTE: This can be confusing and is intended for advanced script authors only.*

Show Error

Use only in an “On Error Goto” procedure

Displays the same error box that would appear if no *On Error Goto* command was encountered, but does not stop the script. As usual, if the user does not respond to the error box within 30 seconds, the error box disappears automatically and Spectrum hangs up the line (if necessary). This is a safety feature for scripts that run unattended (by hanging up, online charges are kept to a minimum).

Script Interpretation

Set CaseSensitive *State*

State can be Off or On

The setting of the CaseSensitive option affects all text comparisons (e.g. *Goto*, *Gosub*, *WaitFor String*, *If Contains*, *Apply Replace*, etc.). It does *not* affect script commands themselves (i.e. *display*, *Display*, *DISPLAY*, *dIsPlAy* work identically regardless of the state of the CaseSensitive option). *NOTE: Each time a script is run (not chained to) CaseSensitive is turned off.*

Example:

```
Display "Type ON or OFF: "; Get Line 0
Set CaseSensitive $0
Goto LaBeL
# label
Display "CaseSensitive is OFF.^M^J"; Stop Script
# LaBeL
Display "CaseSensitive is ON.^M^J"; Stop Script
```

Set Quote *Character*

Sets the special character that is used to delimit a string parameter—usually the double quote (") character. *NOTE: This command must be placed on a line by itself!*

Example:

```
Set Quote A
Display A"You will see these quotes!^M^J"A
Set Quote "
```

Set Token *Character*

Sets the special character that is used to indicate a control character—usually the caret (^) character. *NOTE: This command must be placed on a line by itself!*

Example:

```
Set Token \
Display "\LDisplaying ^L clears the screen.\M\J"
Set Token ^
```

Advanced or Specialty Commands

NOTE: This section describes advanced or specialty commands that should be used with caution. Many of these commands will not work unless a specific online display is being used. Information for each online display can be found on disk in Spectrum's "Documentation" folder.

DirectDisplay "*String*"

Feeds *String* to the current online display as if the data were coming in from the port. The interpretation of *String* is entirely up to the current online display; use this command only when you know a particular display is in use and you want to take advantage of special features in that display.

DirectAction "*String*" *VarNum*

Passes *String* to the current online display for processing; the results are returned in *VarNum*. If the display does nothing then *VarNum* is cleared to "" and the Failed flag is set. If *VarNum* would be longer than 128 characters then a script error occurs.

The available actions that can be performed, as well as the meaning of the returned results, are entirely up to the current display.

Draw Line *Left Right Top Bottom Color Size*

Left and *Right* are values from 0 to 639
Top and *Bottom* are values from 0 to 186
Color is optional; if used it is a value from 0 to 15
Size is optional; if used it is a value from 1 to 15

Draws a line extending from (*Left*,*Top*) to (*Right*,*Bottom*). If *Color* is specified the line is drawn with the specified color (black if not specified). If *Size* is specified the line is *Size* pixels thick (1 pixel if not specified).

Draw Rectangle *Left Right Top Bottom Fill Frame*

Left and *Right* are values from 0 to 639
Top and *Bottom* are values from 0 to 186
Fill and *Frame* are optional; if used they are values from 0 to 15

Draws a rectangle extending from (*Left*,*Top*) to (*Right*,*Bottom*). If *Fill* is specified the rectangle is filled with the specified color (black if not specified). If *Frame* is specified the rectangle is framed with a border in the specified color (black if not specified). *TIP: If you need a different width border than the one automatically provided, just draw two rectangles—one with the frame color, then a smaller one with the fill color.*

Draw Circle *Left Right Top Bottom Fill Frame*

Left and *Right* are values from 0 to 639
Top and *Bottom* are values from 0 to 186
Fill and *Frame* are optional; if used they are values from 0 to 15

Draws a circle inside the area (*Left*,*Top*) to (*Right*,*Bottom*). If *Fill* is specified the circle is filled with the specified color (black if not specified). If *Frame* is specified the circle is framed with a border in the specified color (black if not specified). *TIP: If you need a different width border than the one automatically provided, just draw two circles—one with the frame color, then a smaller one with the fill color.*

Draw Icon *X*,*Y* "*Icon*" "*FoldernameFilename*"

X is a value from 0 to 639
Y is a value from 0 to 186
Icon is the name or resource number of the desired icon
FoldernameFilename is optional; if used it is the file that contains the desired resource

Draws the specified icon resource at coordinates (*X*,*Y*). *NOTE: This command works only on 640 mode super-hires screen displays.*

Spectrum searches for *Icon* in memory (if a file was opened it is searched first). A script error occurs if *Icon* is not found, or if a file was specified but could not be opened.

The following icons are available within Spectrum:

SP-Express Mail	SP-No Mail	SP-Have Mail
SP-In Tray	SP-Out Tray	SP-Mail 0
SP-Mail 1	SP-Mail 2	SP-Mail 3
SP-Mail 4	SP-Mail 5	SP-Mail 6
SP-Mail 7	SP-Mail 8	SP-Mail 9
SP-Stop	SP-Note	SP-Checkmark
SP-Spectrum	SP-Printer	SP-Phone
SP-Spectrum small	SP-Printer small	SP-Phone small
SP-Arrow 1 Down	SP-Arrow 1 Left	SP-Arrow 1 Right
SP-Arrow 1 Up	SP-Arrow 2 Down	SP-Arrow 2 Left
SP-Arrow 2 Right	SP-Arrow 2 Up	

The following icons are available in System 6.0.1's "Sys.Resource" file:

Caution	Note	Stop
Disk	Disk Swap	$$07FF0104
$$07FF0103	$$07FF0102	$$07FF0058

Example:

```
Set OnlineDisplay "Spectrum SHR Fast"; Display "^L"
For 0 0 9; Draw Icon 8,22 "SP-Mail $ForValue0"; Next 0
Draw Icon 1,20 "$$07FF0058"
```

Draw Picture *X*,*Y* "*Picture*" "*FoldernameFilename*"

X is a value from 0 to 639
Y is a value from 0 to 186
Picture is the name or resource number of the desired picture
FoldernameFilename is optional; if used it is the file that contains the desired resource

Draws the specified picture resource at coordinates (*X*,*Y*). *NOTE: This command works only on 640 mode super-hires screen displays.*

Spectrum searches for *Picture* in memory (if a file was opened it is searched first). A script error occurs if *Picture* is not found, or if a file was specified but could not be opened.

Pictures should be stored in the 640 mode screen format, and care should be taken to draw them at the proper *X* coordinate so the dithered colors come out correctly (usually *X* should be even).

Example:

```
    Set OnlineDisplay "Spectrum SHR Fast"; Display "^L"
    Draw Picture 118,35 "About Pic"
```

Set RTS *State*

State can be Off or On

Drops/raises the hardware handshaking line to tell the modem to stop sending information to Spectrum. Do not set RTS off for very long, as incoming data may overflow the modem’s buffer.

Set Init *State*

State can be Off or On

When on, the modem will be initialized before dialing. Normally you should not include this command because the option is automatically controlled (it is on at program startup and turned off after the modem is initialized).

Set InitString "*String*"

Sets the modem initialization string.

Initialize Modem

Initializes the modem if the “Init” option is on. If the modem is initialized successfully (or if the “Init” option was off) then the Failed flag is cleared, otherwise it is set to indicate an error (the modem is not responding).

Make ASCII *VarNum*

Determines the ASCII value of the first character stored in the specified variable, then sets the variable to contain that decimal number. *$VarNum* is empty the Failed flag is set.

Example:

```
    Set Var 3 "A" # the character "A"
    Display "The letter $3 = ASCII "
    Make ASCII 3 # converts "A" into its ASCII value (65)
    Display "$3.^M^J"
```

Make CHAR *VarNum*

Determines the value stored in the specified variable. If *$VarNum* is a number from 0 to 255 then the variable is set to contain the referenced character. If *$VarNum* is empty the Failed flag is set. If *$VarNum* is a number greater than 255 a script error occurs.

Example:

```
    Set Var 3 "65" # the ASCII value for the character "A"
    Display "ASCII $3 = the letter "
    Make CHAR 3 # converts "65" into its character (A)
    Display "$3.^M^J"
```

Expand Variable *VarNum*

Shortcut: **Expand Var** *VarNum*

Expands any replacement items within the given variable. If the expansion makes the length of variable number greater than 128, the Failed flag is set and the variable is not changed.

Example:

```
Display "Type '$Boot' and press Return: "; Get Line 1
Display "^M^JYou typed '$1' which expands to "
Expand Variable 1
Display "$1^M^J^J"
```

Clear PortBuffer

Clears the port buffer of all pending data. One use might be to eliminate unwanted “junk” characters that sometimes occurs when you log off a system.

Example:

```
Store Settings; Set Screen Off; Set Buffer Off
Transmit "Bye^M"; Hangup; Clear PortBuffer
Restore Settings
```

Index

This index attempts to reference all the topics you might look up. If you are looking for a particular topic and can’t find it in this index, please let us know so we can incorporate it in the next printing of the manual.