

FileLink[®]



Automated File Transfer for Windows PCs

User's Guide

-and-

Programmer's Reference

Serengeti Systems, Inc.

FileLink®

Automated File Transfer for Windows PCs

by Serengeti Systems Incorporated



www.serengeti.com

FileLink

Copyright (c) 2013 Serengeti Systems Incorporated

FileLink is sold as is. Serengeti Systems makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties for a particular purpose

Serengeti Systems shall have no liability for loss or damage caused or alleged to be caused directly by this computer program, including but not limited to interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use of this program.

Further, Serengeti Systems reserves the right to revise this publication and program from time to time without notice.

Serengeti Systems Incorporated

1108 Lavaca Street, Suite 110 PMB 431

Austin, Texas 78701 USA

www.robo-ftp.com

www.serengeti.com

Table of Contents

Notice to Evaluation Users	10
FileLink User's Guide	11
Notational Conventions	11
Introducing FileLink	11
FileLink Features Overview	13
General Features	14
File Transfer Features	15
Script Language Features	16
Running FileLink	16
Configuration for Current User vs. All Users	18
Command Line Switches	19
FileLink Console Window	22
FileLink Console Window Details	23
File Menu	25
Tools Menu	26
Scripts Menu	27
Applets Menu	28
Help Menu	29
System Menu	30
Using FileLink	31
Getting Online Help	31
Entering Commands in the Console Window	32
Scheduling File Transfers With FileLink	35
Modems and FileLink	37
Detecting Modems at Startup	38
Running FileLink With Prompt.s	39
Running FileLink as an Icon	40
FileLink and PGP Cryptography	40
PGP Public and Private Keys	41
PGP Passphrases	42
PGP ASCII Armoring	43
PGP Digital Signatures	44
FileLink's Implementation of PGP	45
Backing Up Your Keyring File Set	47
Using PGP With FileLink Step-By-Step Guide	48
Using the FileLink Configurator	49
File Configuration	51
Hardware Configuration	52
Modem Configuration	53
Terminal Configuration	54
Transfer Configuration	55
File Transfer Protocol Configuration	56
ASCII File Transfers	57
Kermit File Transfers	58
Xmodem File Transfers	59

Xmodem1K File Transfers	60
Ymodem File Transfers	61
Zmodem File Transfers	62
PGP Configuration	63
PGP Configuration: Create Key	64
PGP Configuration: Select Key	65
PGP Configuration: Manage Keys	66
User vs. Machine Configuration	67
Using the FileLink TTY Terminal Applet	68
Terminal Connection Menu	69
Terminal Settings Menu	70
Terminal File Transfer Menu	71
Terminal Help Menu	72
The FileLink Script File Editor	72
Script Programming	73
Script File Command Arguments	75
Script File Alphanumeric Constants	76
Script File Numeric Constants	77
Script File Variables	78
Script File Command Options	79
Labels in Script Files	80
Comments in Script Files	81
Debugging Script Files	82
Using Variables in Command Options	83
Using Functions	85
Performing Variable Arithmetic and Numeric Comparisons	89
Performing Date Arithmetic	90
Controlling Script Command Logging	91
Scheduling Script Operation	92
Sending and Receiving E-mail in Script Files	93
Using Shortcut Target Arguments in Script Files	94
Authorizing Remote Users in TTY Mode	95
Authorization File Format	96
Internal Script Variables	96
Using the %cr, %crlf, and %lf Variables	99
Using the %currentlocaldir Variable	100
Using the %date, and %datetime, and %time Variables	101
Using the %dbqueryrawresult, %dbqueryrows and %dbqueryvariables Variables	102
Using the %diffleid, %difffilename, and %difffiletext Variables	103
Using the %difffiles and %diffnum Variables	104
Using the %lasterror Variable	105
Using the %lasterrormsg Variable	106
Using the %lastfile and %lastpath Variables	107
Using the %nextcmd Variable	108
Using the %newport Variable	109
Using the %nextfile, %nextpath, and %nextfolder Variables	110
Using the %nextfiledate, %nextfiledatetime, and %nextfiletime Variables	111
Using the %port Variable	112
Using the %rcvfilecount and %sendfilecount Variables	113
Using the %snapshotfiles Variable	114
Using the %zipcount and %upzipcount Variables	115
Script File Command Overview	115

Script Commands Grouped by Function	119
ANSWER -- Wait for incoming telephone call	123
APPEND -- Append one local file to another	124
ARCHIVEDIR -- Define FileLink's archive folder	125
ASK -- Display dialog box with yes/no question	126
AUTHDATA -- Obtain user data from authorization file	128
AUTHPW -- Verify remote user password	129
AUTHUSER -- Verify remote user name	130
BEGINFUNCTIONS -- Begin function declaration section	131
BREAK -- Set a breakpoint location	132
BROWSE -- Display a pop-up open file dialog box	133
CALL -- Call another script file	134
CHAIN -- Transfer to another script file	135
CHGDIR -- Change local default folder	136
CONNECT -- Open direct connection	137
CONSOLE -- Control output to console window	138
COPY -- Copy one local file to another location	139
CREATEMAIL -- Create an e-mail message	140
CRON -- Schedule script operations	141
DATEADD-- Add specified number of days to date variable	144
DATESUB -- Subtract specified number of days from date variable	145
DBCLOSE -- Close and optionally delete SQL database file	146
DBGETRESULTS -- Get results from a SQL database query	147
DBQUERY -- Issue a SQL query	148
DBREWIND -- Reset query results pointer to first row of results	149
DBUSE -- Create and/or open a SQL database file	150
DEC -- Decrement a variable by one	151
DELDIR -- Delete an empty local folder	152
DELETE -- Delete a local file	153
DIAL -- Initiate modem auto-dialer	154
DIFF -- Look for differences in the local PC file system	155
DIFFREWIND -- Reset file pointer for GETDIFF command	156
DISPLAY -- Display all or a specified variable	157
DISCONNECT -- Disconnect the line	158
DOSCMD -- Execute an MS-DOS command	159
ENDFUNCTION -- End function declaration	160
ENDFUNCTIONS -- End function declaration section	161
EXEC -- Execute a external program	162
EXIT -- Quit FileLink	164
EXPORT -- Export Configuration Settings	165
FLUSH -- Flush characters from receive buffer	166
FUNCTION -- Begin a function declaration	167
GETDIFF -- Get specific changes within local PC file system	168
GETFILE -- Get file from folder structure on local PC	170
GETMAIL -- Get an e-mail message	171
GETNEXTFILE -- Get file or folder names on local PC	173
GETREWIND -- Reset file pointer for GETFILE command	176
GO -- Rerun the currently defined script file	177
GOTO -- Direct flow to label	178
IFDATE -- Conditional branch upon file date comparison	179
IFERROR -- Conditional branch after testing result code	181
IFFILE -- Conditional branch on file existence	182
IFNFILE -- Conditional branch on file non-existence	183
IFNO -- Conditional branch if 'No' is clicked in ASK dialog box	184

IFNSTRCMP -- Conditional branch when two string variables are not equal	185
IFNSUBSTR -- Conditional branch if sub-string is not found in string variable	186
IFNUM -- Conditional branch upon numeric variable comparison	187
IFSIZE -- Conditional branch upon file size comparison	188
IFSTRCMP -- Conditional branch when two string variables are equal	189
IFSUBSTR -- Conditional branch if sub-string is found in string variable	190
IFTIME -- Conditional branch upon time comparison	191
IFYES -- Conditional branch if 'Yes' is clicked in ASK dialog box	193
IMPORT -- Import Configuration Settings	194
INC -- Increment a variable by one	195
LINEIN -- Read one or more characters from COM port	196
LINEOUT -- Write one or more characters to COM port	198
LISTDIR -- List local directory to a file	200
LOG -- Control the script log file	201
LOGMSG -- Write a message to the script log file	203
LOGNTEVENT -- Write a message to the NT application event log	204
LOOPIF -- Conditional branch used in conjunction with LOOPCOUNT	205
LOOPTO -- Unconditional branch used in conjunction with LOOPCOUNT	206
LOOPCOUNT -- Set maximum loop repetition	207
MAILTO -- Send an e-mail message via default e-mail client	208
MAKEDIR -- Create a new local folder	209
MAKEFILENAME -- Create a unique, non-existent file name	210
MESSAGEBOX -- Display text in message box	211
MINIMIZE -- Minimize FileLink window	213
MODEMCMD -- Send AT command string to modem	214
MODEMDEFAULTS -- Set modem to factory defaults	215
MODEMDETECT -- Locate first available modem and/or COM port	216
MODEMRESET -- Send reset to modem	217
MODEMRESP -- Read modem response	218
MOVE -- Move one local file to another location	219
NATO -- Specify a no activity time-out	220
PAUSE -- Pause for specified length of time or until specified hour:minute	221
PERFORM -- Execute script command contained in character string or variable	222
PGPCOMMAND -- Send a "raw" GnuPG command	223
PGPDECRYPT -- Decrypt a PGP encrypted file	225
PGPENCRYPT -- Encrypt a file using PGP	228
PGPIMPORT -- Import a PGP key	232
PLAYSOUND -- Play a sound (.wav) file	233
PRESSANYKEY -- Suspend script execution pending a key press	234
PRINT -- Print a file	235
PROMPT -- Display message box with title and prompt, and accept user input	237
PROTOCOL -- Specify default file transfer protocol	238
RCVFILE -- Receive one or more files	240
READFILE -- Read string variable value from text file	242
REMOTECMD -- Perform a script command received via a COM port	244
RENAME -- Rename a file	245
RESTORE -- Restore minimized FileLink window to original size	246
RESUME -- Resume script execution from a breakpoint	247
RETURN -- Return from a called script file or function	248
SENDCMD -- Send script command (same as LINEOUT)	250
SENDFILE -- Send one or more files	251
SENDMAIL -- Send an e-mail message	253
SET -- Assign or concatenate string variable(s)	254
SETEXTRACT -- Extract delimited substring from a string	256

SETLEFT -- Extract left substring	257
SETLEN -- Assign length of specified string to a variable	258
SETMID -- Extract mid substring	259
SETNUM -- Assign or evaluate numeric variable(s)	260
SETRIGHT -- Extract right substring	262
SETSUBSTR -- Find number of substrings in string	263
SNAPSHOT -- Take a "snapshot" of the local PC file system	264
SPEAKER -- Control modem speaker mode	265
SRVNAME -- Define service name and control interaction with SrvMonitor	266
STOP -- Stops script processing	267
TERMINAL -- Activate Terminal applet	268
TRACELOG -- Control the trace log file	269
TRACEWIN -- Activate/deactivate trace window	271
UNZIP - Extract file(s) from a zip archive	273
USEPORT -- Specify COM port and/or port settings	275
WORKINGDIR -- Specify default working folder	277
WRITEFILE -- Write character string or string variable value to text file	278
ZIP -- Create or add to a zip archive	279
Sample Script Files	280
Simple Async Dial-Up Connection	281
Simple Async Dial-Up Connection With Error Recovery	282
Dial-Up Connection Performing a Logon	283
Dial-In Connection With Authorization	284
Installing FileLink as an NT Service	285
Shutting Down a Running FileLink Service	287
Monitoring a FileLink Service	288
Using the CronMaker Utility	290
CronMaker Event Creation Example	292
CronMaker Event Creation Example P2	293
CronMaker Event Creation Example P3	294
CronMaker Event Creation Example P4	295
Cron Event File Format	296
Using COM/OLE to Control FileLink	298
COM/OLE Operational Overview	299
Sample C++ and Visual Basic Project Files	300
FileLink COM/OLE Interface Description - A Programmer's View	302
FLStartSession -- Method to initiate a FileLink session	303
FLEndSession -- Method to terminate a FileLink session	304
FLSendCommand -- Method to send a script command to FileLink	305
FLStopCommand -- Method to stop a running FileLink script command	307
FLGetVariable -- Method to get the value of a FileLink script variable	308
FLGetVBSVariable -- VBS method to get the value of FileLink script variable	309
FLCommandProgress -- Event fired to update SENDFILE/RCVFILE progress	310
FLCommandResult -- Event fired at the conclusion of a non-blocking command	311
FLLogMsgs -- Event fired to provide script log information	312
COM/OLE Return Codes	313
Sample VBScript Program	314
Using Script File Result Codes	314

Index

320

Notice to Evaluation Users



Thank you for your interest in FileLink®.

We are pleased that you have taken the time to obtain this evaluation version of FileLink. You have 30 days from the time you install the software to put FileLink through its paces. If you decide to continue using FileLink past 30 days, simply contact us to purchase the product, and we'll authorize the version you have for unlimited use.

Important

All authorization of FileLink is performed through the program's **Help** menu. There are instructions provided for each step of the license authorization process if you elect to purchase FileLink.

We'll keep you informed about changes to FileLink so you can always be up to date. We also encourage you to give us feedback (both positive and negative) about your experience with FileLink.

Thanks again!

FileLink User's Guide

Notational Conventions

Various fonts, type styles, and notations are used throughout this help file to make it more understandable. Below is a list of type examples with explanations of their purpose in this file.

<code>OPCODE</code>	Upper case Courier font is used for FileLink script commands in examples throughout this help document; script command opcodes may be upper or lower case in actual use
(Esc)	Bolded and inside (..) are references to named keys on the keyboard
variable	Bolded text within paragraph bodies refer to FileLink command variables or is also used to indicate a file name
/option	Bolded text within paragraph bodies preceded by a / refer to FileLink command options
[text]	Text inside [..] refer to script command arguments or variables in syntax descriptions
<i>=nn, or nn</i>	Italicized n's refer to numeric values
<i>= "xxx"</i>	Italicized x's in quotation marks refer to string values
[..] [..]	The vertical bar reads as <i>or</i> meaning either one or the other or both of the [..] items may be present

Introducing FileLink

We call FileLink® ***The Automated File Transfer Solution.***

FileLink brings the power of mainframe oriented batch, unattended file transfer operation to the everyday world of asynchronous modem protocol (Xmodem, Zmodem, etc.).

FileLink's design is centered around the execution of script files but the product also includes simple but very capable TTY-style terminal emulator capable of ANSI, DEC, Televideo, Wyse and other "glass teletype" terminal emulation.

By providing automated, unattended operation, FileLink can be set up to send and receive files at any time of the day or night. For example, FileLink's powerful [script language](#) can place a call through your modem, recognize and send character strings (such as logon and password sequences), send and receive text and binary files using a variety of file transfer protocols, and detect and recover from errors.

Advanced scripting that takes advantage of FileLink's integrated SQL database and PGP encryption/decryption capabilities is also possible.

FileLink maintains a detailed log file to record the events of a file transfer session. Each log file entry is stamped with the system date and time and records the step-by-step activity of a session for later review. Each log entry is immediately written to the log file so it is always up to date - even if your system loses power or crashes.

In summary, FileLink has been designed from the ground up as an unattended, file transfer package. Many other popular asynchronous communications packages designed for attended operation are excellent if you are there to drive, but often come up short if your needs call for unattended operation. For those times, there is FileLink.

See also: [FileLink Features](#)

FileLink Features Overview

Serengeti Systems has been making mainframe file transfer products for almost 20 years. We now bring all that knowledge to world of async modem with the introduction of FileLink.

You can find a comprehensive list of FileLink features by following the links below.

[General Features](#)

[File Transfer Features](#)

[Script Language Features](#)

General Features

- Multi-threaded 32-bit Windows application
- Powerful script language for completely unattended operation
- Support for popular asynchronous file transfer protocols including Xmodem, Kermit, Ymodem, and Zmodem
- FileLink Script File Editor (or any other editor) access for script file editing
- CronMaker utility to easily manage scheduling of script events associated with the CRON script command
- Auto-dial & auto-answer with Hayes compatible modems
- Direct connection for modem-less connections
- Up to 56K bps over dial-up lines; up to 115.2K bps for direct connections
- Send and receive e-mail messages with SMTP and POP servers under script control
- "Hot send" feature for automatic file transmission
- Automatic naming of received files
- TTY Terminal applet for built-in, interactive TTY sessions
- Emulation of popular TTY terminals like VT100, Wyse, and Televideo
- Built-in PGP encryption, decryption, and keyring management support
- Complete session log with date and time stamps
- Log file management including automatic naming, and maximum size limitation
- Multiple simultaneous sessions from a single system
- Configurable for COM1 up to COM9 (for FileLink) or simultaneous use of COM1 up through COM48 (for FileLink XE)
- COM/OLE automation built-in providing a script command interface for user-written applications (sample C++ and Visual Basic projects provided)

File Transfer Features

- ASCII mode for simple, unblocked transmission and reception of text files
- Kermit protocol for blocked, error-correcting transmission and reception of text and binary files
- Xmodem and Xmodem1K protocol for blocked, error-correcting transmission and reception of text and binary files
- Ymodem protocol for blocked, error-correcting transmission and reception of text and binary files
- Zmodem protocol for blocked, error-correcting transmission and reception of text and binary files
- Override control on remotely named files sent to FileLink
- Configurable Kermit compression control
- Configurable Kermit 7- or 8-bit data path
- Both 128 and 1K block size (Xmodem and Xmodem1K)
- Configurable CRC or LRC block checking
- Configurable ACK control (referred to as the G-option for the Xmodem1K and Ymodem protocols)
- Configurable Zmodem sliding window
- Optional Zmodem recovery from interrupted transmissions (crash recovery)
- Configurable Zmodem *over-write* file control

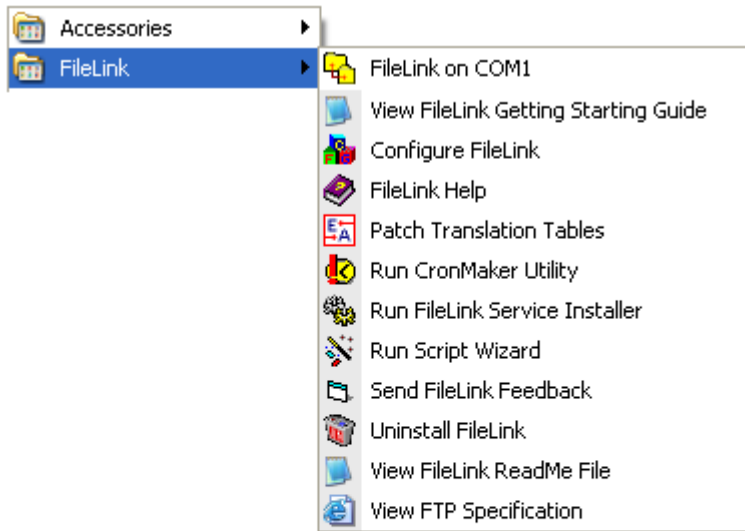
Script Language Features

- Integrated SQL database support
- Integrated PGP encryption and decryption support
- Execute scripts from the command line, batch files, parent process, or menu
- Execute external processes and evaluate results
- Command line parameter substitution
- Call scripts from within scripts
- Time activated commands
- Branching and looping commands
- Extensive result codes and error testing
- String variables
- String and sub-string manipulation
- Commands for interactive prompting and messaging
- Accept commands from a remote system via a COM port
- Accept commands from a client process
- Remote user name and password authorization
- Write messages directly to NT event log
- Complete session log with date and time stamps
- Log file management including automatic naming, maximum size limitation
- Create or add to zip archive files
- Extract file(s) from zip archive files

Running FileLink

The focal point of FileLink's design centers around the running of script files. FileLink can be used interactively but the user interface is oriented to the development and testing of script files. Typically you will use the interactive capabilities of FileLink as necessary to create the script file you need, and then create one or more shortcuts to launch the program running one of these previously created script files.

If you allow the [configurator](#) to create a shortcut (in the **Start** menu or on the desktop) for you, the shortcuts look like the following:



Start menu shortcut



Desktop shortcut

[Shortcut Properties Command line switches](#)

For information on how to run FileLink interactively, follow the following links.

[Running FileLink in a Window](#)

[Running FileLink With Prompting](#)

[Running FileLink as an Icon](#)

[Using the FileLink TTY Terminal Applet](#)

Configuration for Current User vs. All Users

When you configure FileLink, the settings you select are saved in the Windows registry in one of two locations depending on whether FileLink was installed to be used by the current user only or any user that may log onto a given PC. These types of installation are mutually exclusive.

When installed for the current user, settings are saved in the HKEY_CURRENT_USER (HKCU) registry location. Whenever this user logs onto Windows, FileLink finds the settings saved for this user in **HKCU\Software\FileLink**. If another user logs on and runs FileLink, there will be no saved settings associated with this user, and FileLink will have to be configured anew. This permits each user to have unique configuration settings for FileLink.

In most cases this is fine. In some environments, however, you may wish to have a single configuration for all users on a given machine. In this case, settings are saved in the HKEY_LOCAL_MACHINE (HKLM) registry location and all FileLink users share a common configuration. This method must also be used if FileLink is to be operated as an NT service.

To accommodate both possibilities, FileLink first looks in HKEY_LOCAL_MACHINE for configuration settings. If the settings are not found, FileLink next looks in HKEY_CURRENT_USER.

If neither location contain FileLink configuration settings, it is likely that FileLink is being run without having been completely installed using the InstallShield setup wizard. In this case, FileLink automatically creates a default configuration for the current user. If the FileLink Configurator is run first in this situation, it prompts you to create a new configuration for the current user or for all users.

Command Line Switches

To load FileLink, double-click on a shortcut on your Windows desktop created by the FileLink configurator or one of the menu items that appears when clicking the **Start** button and navigating to **All Programs | FileLink**. One or more of the following command line switches may have been associated with FileLink during installation, or may be specified when defining the properties of FileLink within the Target control of the Shortcut Properties dialog.

The general command line syntax is

```
filelink [-d] [-gfile] [-m] [-n] [-pn] [-sfile] [-v] [&var1& | %var2%] [-x]
```

The following paragraphs describe these and other command line switches in detail:

-d **Debug mode**

The **-d** switch activates the debug mode. The debug mode writes internal debug information to the log file during script processing. This switch is only useful for diagnostics to be interpreted by a Serengeti System technical support engineer.

-ft **Accept FileLink default settings**

The **-ft** switch may be used with the **-px** switch to suppress a pop-up message reminding the user to run the FileLink configurator for the detected COM port if the port has not been previously configured. This switch is useful in unattended environments when the default settings are acceptable and/or when there is no user present to respond to the pop-up message.

-g file **Get import settings file**

The **-g** switch specifies the optional settings file to be used to configure FileLink. Settings files have an **.set** extension and are created by a previous run of FileLink. The settings are read from the file and transferred to the Registry overwriting all previously existing settings. Typically settings would only need to be imported once. Also, refer to the [EXPORT](#) and [IMPORT](#) script commands.

-h **Run hidden**

The **-h** switch results in FileLink running completely invisible from the desktop.

-l **Lock a minimized window**

The **-l** switch may be specified with the **-m** switch to lock the minimized state of FileLink and prevent a user from maximizing the window. This may be advantageous to prevent users from having access to the console window and its associated controls. This switch is ignored if the **-m** switch is not present.

-m **Run minimized**

The **-m** switch results in FileLink starting with its main window minimized.

-n **No splash screen**

The **-n** switch suppresses the FileLink splash screen and other informative windows (i.e., information about auto-detected COM port(s), see **-p** switch) that may otherwise be displayed when the program first loads.

-p *n* **Specify COM port**

The **-p** switch selects the COM port to be used by FileLink. FileLink may be used on a single COM port in a given system. For example, **-p2** specifies COM2. Default settings are assumed for each COM port. If necessary, FileLink may be configured specifically for each COM port by running FileLink configurator.

If you do not know which COM port to use ahead of time, specify **-px**. This option results in FileLink scanning the PC for all available COM ports and using the first COM where a modem is detected. If no modem is detected, then the first COM port detected is used. The default configuration settings are used if the port found has not been previously configured. (Also see the **-ft** switch.)

-s *file* **Execute script file**

The **-s** switch specifies the script file to be loaded and executed. Pressing the **(Esc)** key or clicking the Stop button cancels script file execution.

-t [*id string*] **Specify SrvMonitor Service Name**

The **-t** switch specifies a service name used to identify this particular instance of FileLink to **SrvMonitor** (a Windows tray applet) provided with FileLink that may be used to monitor the operation of FileLink when it is running as an NT service or when it is running minimized. For more information see [Monitoring a FileLink Service](#) and the [SRVNAME](#) script command.

-tl **Enable TTY Terminal applet logging**

The **-tl** switch may be helpful to Serengeti System technical support in troubleshooting problems with terminal emulation in the TTY Terminal applet. You may be asked to add with switch to the FileLink shortcut to produce a set of log files to be sent in for analysis.

-ua / -uc **Initialize FileLink for use by All Users / Current User Only**

The **-ua** and **-uc** switches are not commonly used unless FileLink is to be run on a system and it has NOT been installed using the provided InstallShield installation program (i.e., **setup.exe**). Under such circumstances, FileLink will use the specified switch to create the appropriate Windows registry hive (where its settings are saved) according which user(s) may use FileLink.

-v **Run as Windows service**

The **-v** switch should be specified whenever FileLink is started as a Windows service.

-x **Run TTY Terminal applet only**

The **-x** switch should be specified whenever the FileLink console window (and script processor) is to be suppressed and only the TTY Terminal applet is to be visible.

&sub-parm&

- Or -

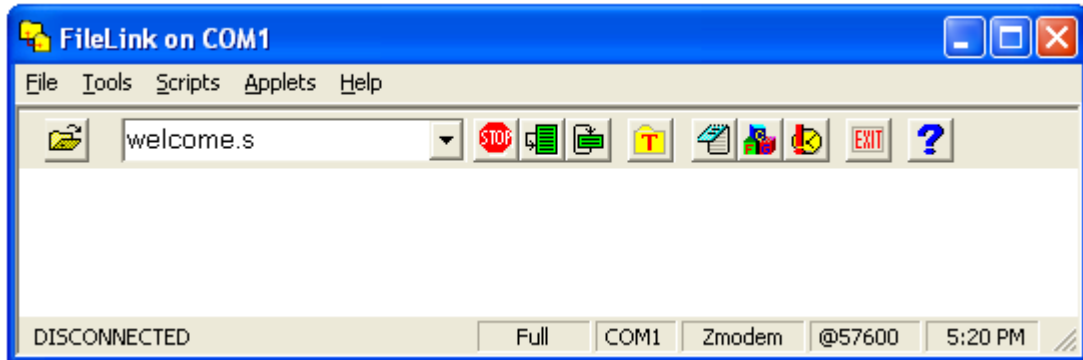
%sub-parm% **Define Shortcut Target argument**

Arguments delimited by & or % are assigned to internally defined script file variables %1 through %2. There can be up to nine script file variables assigned in this way. Script file variable arguments must follow the last switch passed into FileLink as shown below.

```
C:\program files\fileLink.exe -n -sscript.s %var1% %var2%
```

FileLink Console Window

The main FileLink window is shown below. There is the main menu and numerous toolbar buttons to control the execution of your script files. Click on a menu item to explore the FileLink menu system or click on a button to read a short description of its function.



To run an unattended session using FileLink, create a script file specific to your needs with the FileLink Script File Editor (which can be launched directly from within FileLink). If you prefer a different text editor, you can configure FileLink accordingly.

You may load FileLink and click the open file button to specify the script file to run. Otherwise create a shortcut to FileLink and add the **-s<file name>** switch to the command line to specify the name of your script file. Double click on the FileLink shortcut icon to run FileLink with your script file.

[Script Language](#)

[TTY Terminal Applet](#)

[CronMaker Utility](#)

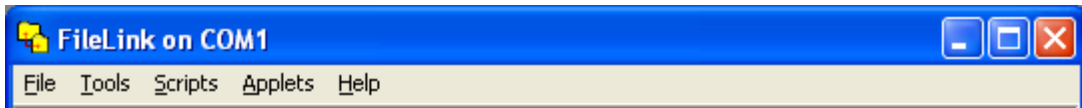
FileLink Console Window Details

By default FileLink runs in a small window (called the console window) that allows you to type script commands for immediate execution and control script file execution via a simple user interface consisting of a menu bar System_Menu and series of toolbar buttons. For information on entering commands into the console window, see [Entering Commands in the Console Window](#).

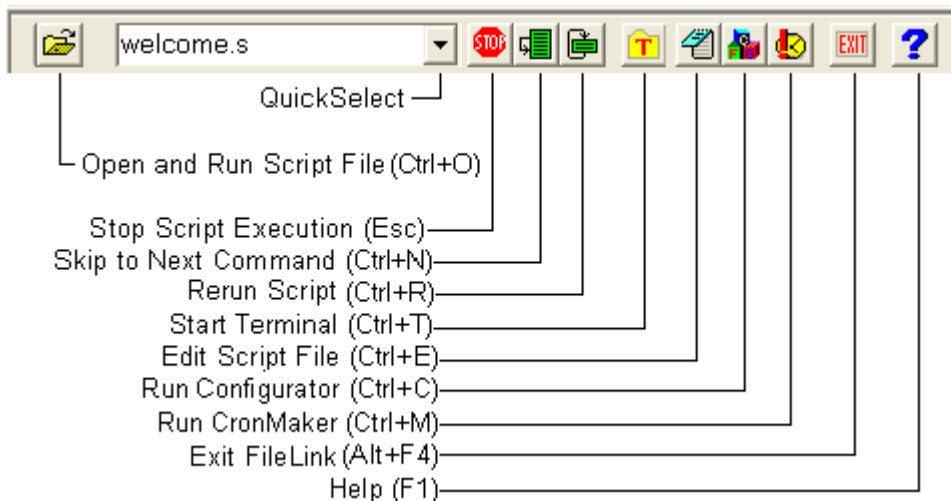
Text displayed in the console window may be displayed in color. The colors are keyed to the meaning of each line of text.

Black	Command echo and other information
Green	Command result messages
Red	Error messages and BREAK indicator

The console window menu bar is shown below.



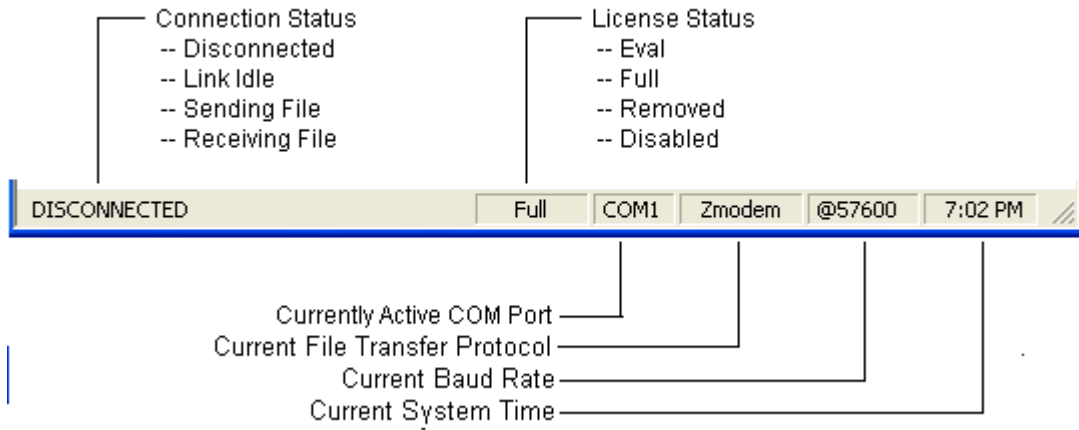
However, FileLink is primarily operated via the toolbar buttons. These buttons provide the controls shown below.



When FileLink is running in a window, the current COM port and script file name is displayed as part of the program title.



The bottom window pane contains the information shown below.

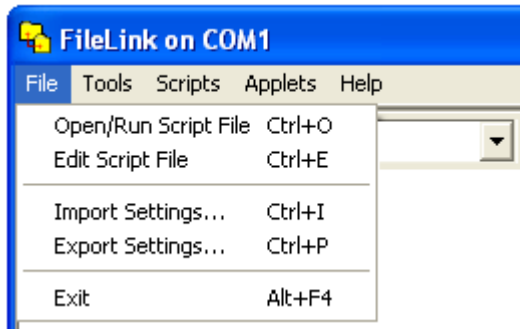


When FileLink is running in a window, the window remains open when script execution is complete, unless the [EXIT](#) script command is executed. This allows you to scroll back and view the output from the FileLink session, or to restart the script file.

See also: [Using FileLink](#), [System Menu](#)

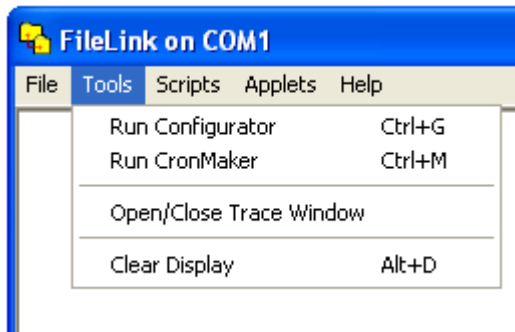
File Menu

The FileLink **File** menu is shown below. The actions off this menu support opening and editing script files, and saving and restoring configuration settings. Click on a menu item for information on its function.



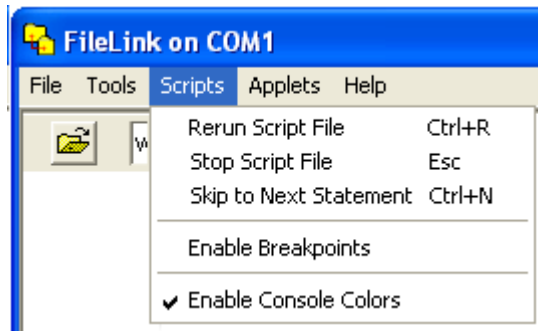
Tools Menu

The FileLink **Tools** menu is shown below. The actions off this menu support running the configurator, managing the trace window, and clearing the FileLink main window. Click on a menu item for information on its function.



Scripts Menu

The FileLink Scripts menu is shown below. The actions off this menu manage the running of script files. Click on a menu item for information on its function.



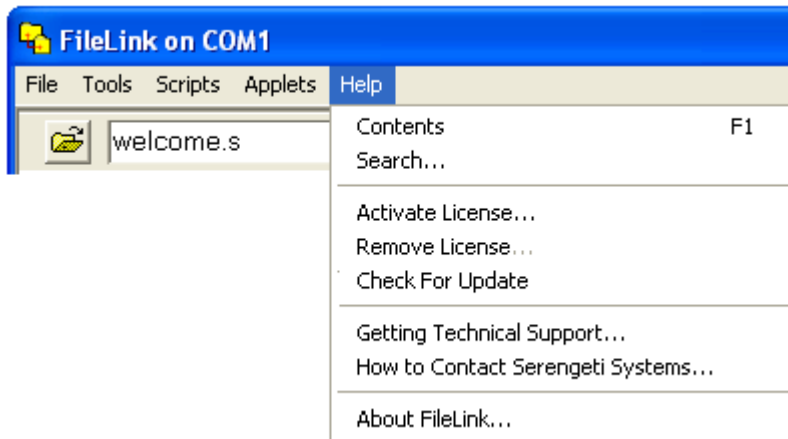
Applets Menu

The FileLink **Applets** menu is shown below. Click on a menu item for information on its function.



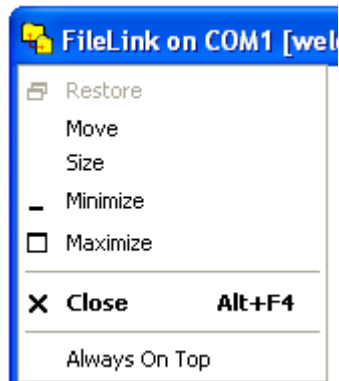
Help Menu

The FileLink **Help** menu is shown below. The actions off this menu support getting online help with the operation of FileLink and script file programming, authorization and product licensing, and contacting Serengeti Systems. Click on a menu item for information on its function.



System Menu

One FileLink control is present in the **System** menu that drops down when you click on the icon in the upper left corner of the FileLink window. The **System** menu is shown below.



The FileLink control of interest is described below.

Always On Top

Click [here](#) if you wish the FileLink window to always be visible on your screen.

Using FileLink

Getting Online Help

Online help is available within the various components of FileLink.

FileLink Configurator

Within the FileLink Configurator, click on the **Help** button to view the Table of Contents of the FileLink help file. For help on a specific file or prompt, click on the ? (question mark) in the upper left corner of the FileLink window. The cursor changes to include a question mark. Now to display a pop-up help window on a particular subject, position the cursor over the item and click again.

FileLink

Within FileLink itself, click on the ? (question mark) button to view the Table of Contents of the FileLink help file. For more specific help, highlight text (such as a script command) in the FileLink window, and then click on the ? (question mark) button. If there is an exact match in the help file, that information is displayed. If the exact information is not matched, the help file index is displayed matching your highlighted text as closely as possible.

FileLink TTY Terminal Applet

Within the FileLink Terminal applet, click on the **Help** menu item to view the portion of the help file related to use of this applet.

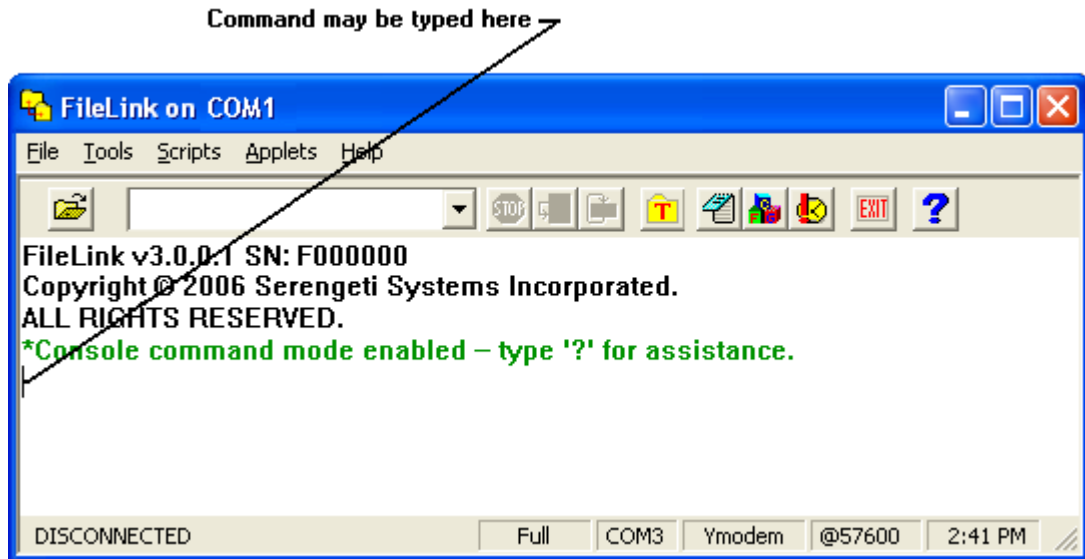
Note

If the background color of some the images in this help file are not white, then you should change the **Colors** control in Control Panel **Display Properties | Settings** to *High Color (16 bit)* or higher.

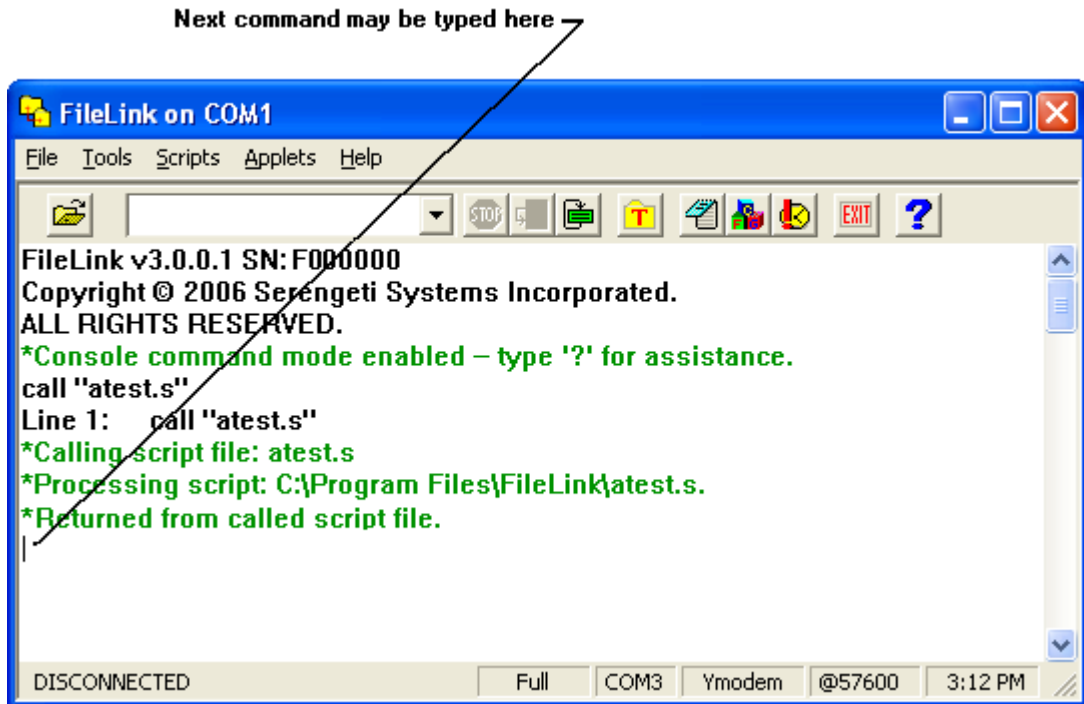
Entering Commands in the Console Window

Beginning with v2.1.0, FileLink's console window may be used interactively to enter, edit, and execute individual script commands. The interaction of FileLink within the console window is much like the behavior of the Windows Command Prompt window.

The console window is not intended as FileLink's primary user interface. That remains the FileLink script language. However, the console window is useful for typing the occasional command and particularly for script debugging.

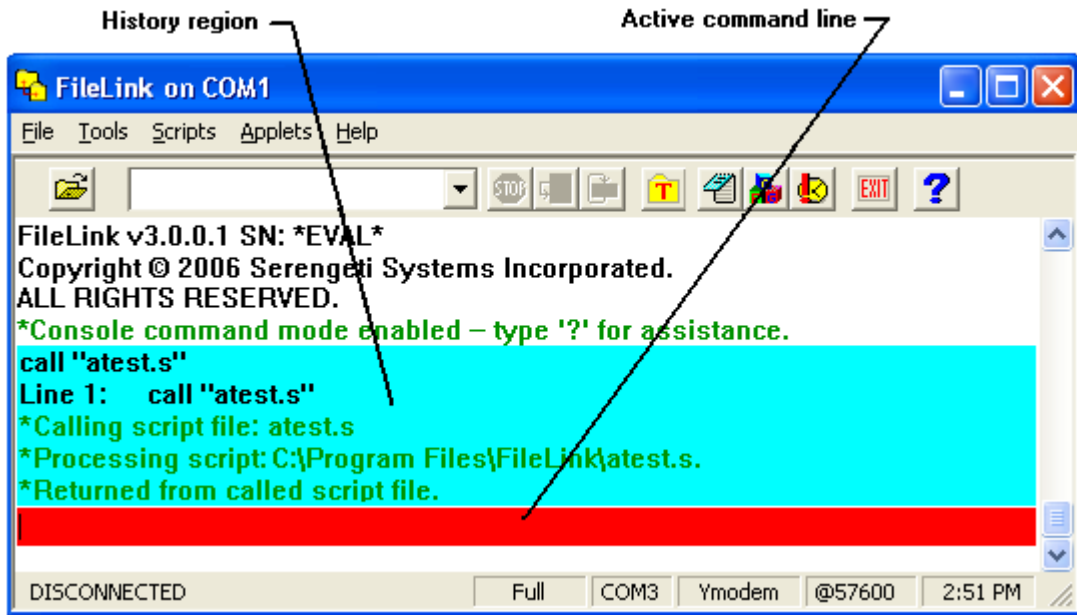


After entering a command and it is executed by FileLink, control returns to the console window and you may enter another command.



The following control keys are recognized in the FileLink console window:

F1	Display general help or help about a specific command if pressed with the cursor over the command.
Up Arrow	Move cursor off the active command line into the preceding history region (perhaps to position the cursor over a previously executed command so F1 can be pressed).
Down Arrow	Cycle through the previous 10 commands entered if in the active command line; move the cursor down one line if in the history region.
Enter	Execute the command if the cursor is within the active command line; if the cursor is in the history region, reposition the cursor to the end of command line.
Home	Reposition the cursor to the beginning of the command line.
End	Reposition the cursor to the end of the command line.
Backspace	Erase previous character within the command line.
Left/Right Arrow	Reposition the cursor left or right within the command line or the history region.
Alt + D	Clear the window.



Scheduling File Transfers With FileLink

Part of FileLink's usefulness is derived from its ability to perform file transfers unattended at scheduled times when running a script.

The Scheduling Script Commands

Scheduling operations within a script file are controlled by using either the PAUSE or CRON script commands.

The [PAUSE](#) command is the simplest method and is best suited for scheduling a one-time event. This command results in script execution being suspended for fixed period of time (e.g., 60 minutes) or until a fixed time of day (e.g., 11PM). At the designated time, the script "wakes up" and performs a series of tasks. If the tasks are to be repeated, the script would loop back to the PAUSE command and wait for the next designated time to arrive.

The [CRON](#) command provides much more complex scheduling. This command allows events like "every Tuesday and Thursday at 11PM", "once a month at midnight of the 5th day", "every 15 minutes on March 30", etc., and any combination thereof to be scheduled. To make scheduling with the CRON command more simple, FileLink includes the [CronMaker scheduling utility](#). With CronMaker, you build the "[crontab.txt](#)" input file to the CRON command with a few mouse clicks. When the next CRON event triggers, the script "wakes up" and performs a series of tasks. If the tasks are to be repeated, the script would loop back to the CRON command and wait for the next CRON event to trigger.

Launch FileLink running a script with one or both of these commands and the scripted tasks will occur when you want them to.

Using the CRON Command - A Brief Overview

When scheduling task(s) with CRON, there are usually three discrete components that need to be created:

1. One or more scripts that perform the tasks you want done.
2. The event file used by the CRON command.
3. A simple "master" script which loops to execute the CRON command and to launch the script (s) that do the work.

To schedule tasks with CRON, the following step-by-step approach is suggested.

First, create and test the script(s) needed to perform the file transfer tasks at hand. It is important that you test that each script does what you want it to BEFORE subjecting them to a final scheduled environment.

Next use the CronMaker utility to create or edit the event schedule. This process creates an event file (normally named "crontab.txt" that is saved in FileLink's program files folder). This file is direct input for the CRON command.

To make it all work, launch FileLink with the master script. The master script executes the CRON command which will in turn launch task specific scripts at the scheduled times and then

loop back to await the occurrence of the next event.

The following is a sample master CRON script:

```
:loop
CRON                ;; wait for scheduled event
PERFORM %nextcmd  ;; launch the task script
GOTO loop          ;; loop to wait for next event
```

There is a similar file named "CronMaster.s" provided with FileLink in the Sample Scripts folder.

Scheduling Dependability When Running as an NT Service

This scheduling capability is further enhanced by FileLink's ability to be installed and run as an NT service on Windows NT, 2000, and XP machines. (NT services are not supported in Windows 98 or ME.) When running as a service, FileLink starts when the system boots up and remains active all the time - you don't need to remember to start it up every day.

Installing FileLink as an NT service is easy using the provided [SrvInstaller utility](#). To perform scheduled operations, install FileLink as a service using a suitable script that incorporates either or both of the PAUSE and CRON script commands.

Summary

The combination of FileLink's scheduling and NT service capability is a powerful combination to make sure that your file transfers take place dependability when you want them to.

Modems and FileLink

FileLink is designed for use with AT command set modems. These are also referred to as Hayes compatible modems.

FileLink expects modems to be in their factory default settings. The first thing to do if you are experiencing communications problems is to make sure your modem is set for factory defaults. See the [MODEMDEFAULTS](#) script command for more information.

If the factory settings are in place and communications problems persists, then the modem initialization string preconfigured by the FileLink Configurator may not be correct for your particular modem. For example, we have found that the Courier V.Everything modem requires a different modem initialization string.

Default modem initialization string: **ATW2&D2**

Courier modem initialization string: **AT&F1X4**

Refer to your modem documentation for initialization commands or contact Serengeti Systems technical support if you are unable to make your modem work with FileLink.

Detecting Modems

FileLink can detect a modem attached to one of the COM ports in your PC. This allows FileLink to configure itself if it is run on a PC and the location of the modem is not known ahead of time. This detection can take place when FileLink is initially loaded by adding the **-px** switch to the program shortcut or by running the [MODEMDETECT](#) script command.

Related Command(s): [MODEMCMD](#), [MODEMRESET](#), [DIAL](#), [ANSWER](#)

Detecting Modems at Startup

Each instance of FileLink must be associated with an available COM port in your PC.

FileLink provides the **-p** shortcut switch to the purpose of specifying the associated connection when the program starts.

In some cases, you may not know which COM port(s) are available on a given PC. FileLink has the ability to scan the PC for COM ports and then to attempt to detect a modem on each of the ports. This is done by specifying the **-px** switch.

When auto-detecting COM ports, FileLink will utilize the COM port where the first modem is detected; if no modem is detected, FileLink will utilize the first COM port found. In all cases, if predefined settings for the found COM port exist (previously created using the FileLink configurator), these settings will be used, otherwise the default settings are used.

In addition, the [MODEMDETECT](#) script command is available to perform this same operating at script execution time.

See also: [Shortcut Properties Command Line Switches](#)

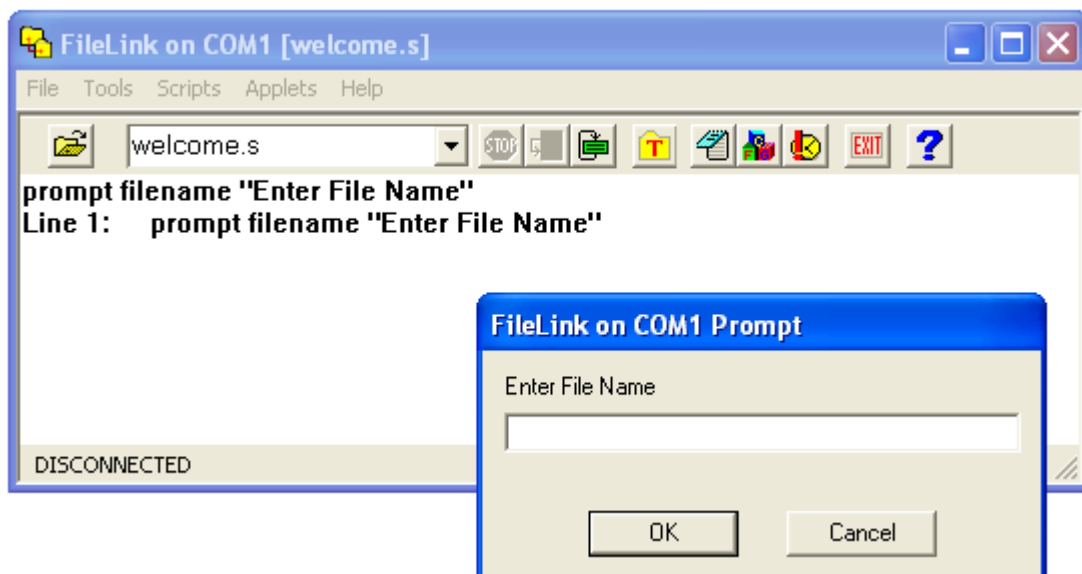
Running FileLink With Prompt.s

An alternate method to typing in commands in for FileLink to execute (rather than into the console window) is to use the provided **prompt.s** script file. This script file is focused around the **PROMPT** script command and effectively demonstrates its use. Using this command in a simple script file demonstrates a scripted, prompt-driven FileLink environment.

The **prompt.s** script file provided with FileLink for this purpose is shown below. The following script illustrates how FileLink can be scripted to prompt for a script command (which is saved in the variable `var`) which is then provided to the **PERFORM** command for execution.

```
:top
  PROMPT var /history=on
IFERROR= $ERROR_PROMPT_CANCELLED goto done
  PERFORM var
  GOTO top
:done
  ASK "Exit FileLink?"
  IFNO goto top
  MESSAGEBOX "Exiting FileLink"
:exit
```

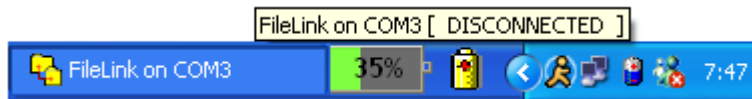
The prompt dialog box looks like this.



Running FileLink as an Icon

FileLink runs as an icon if Run Minimized is specified in the Shortcut Properties for FileLink, if the MINIMIZE script command is executed, or if the FileLink window is minimized manually.

When FileLink is running as an icon, the link state of the connection is included in the program title.



The link states are:

- DISCONNECTED
- LINK IDLE
- SENDING FILE
- RECEIVING FILE

When running as an icon, FileLink terminates without any user intervention when script file execution completes. The FileLink log file contains a history of the session if you need to review it later.

FileLink and PGP Cryptography

FileLink incorporates the ability to encrypt and decrypt files using PGP (an acronym for Pretty Good Privacy) cryptography and for managing PGP keyrings.

PGP is a commonly recognized method of what is referred to as *strong cryptography*. Cryptography is the science of using mathematics to encrypt and decrypt data. PGP cryptography enables you to protect sensitive information when it is stored locally or transmitted to a remote location, so that it cannot be read by anyone except the intended recipient.

Of course, PGP cryptography is useless if only one party uses it. To use PGP in conjunction with FileLink, you and the remote system must be in agreement to use PGP cryptography..

In general, cryptography can be *strong* or *weak*. Cryptographic strength is measured in the time and resources it would require to recover the original information. For all practical purposes, using the technology available, it is impossible to an undesired recipient to decipher the result of strong cryptography. No one can say if strong encryption will hold up under tomorrow's computing power, but the cryptography employed by PGP is among the best available today.

PGP Public and Private Keys

PGP utilizes *public key cryptography*. Public key cryptography is a scheme that uses a pair of keys for encryption: a *public key*, which encrypts data, and a corresponding *private*, or *secret key* for decryption. You provide your public key to anyone you want to share information with while keeping your private key secret.

Anyone with a copy of your public key can then encrypt information that only you can read. Conversely, if you have someone else's public key, you can encrypt information that only they can read - in other words, only the person who has the corresponding private key can decrypt the information.

PGP, as implemented by FileLink, permits you to create keys of three different sizes (measured in bits): 768, 1024, and 2048. Larger keys will be cryptographically secure for a longer period of time. If what you want to encrypt needs to be hidden for many years, you might want to use the largest key.

Keys are stored in encrypted form in files called *keyrings*. As you use PGP, you will typically add the public keys of your recipients to your keyring. If you lose your keyring, you will be unable to decrypt any information encrypted by keys on that keyring. By default, FileLink expects keyring files to be in its current working folder.

Keys created by FileLink consist of the following user-supplied elements:

- A user name (required and at least five characters in length)
- A comment (optional and of any desired length)
- An e-mail address (optional and of any desired length)
- A [passphrase](#) (required and at least eight characters in length)

When encrypting a file, the public key of the recipient must be specified. Keys are identified by any combination of user name, comment, and/or e-mail address. This combination is often referred to as a **key ID**. These elements may be specified in part or in full. You often see a key ID specified in the following format:

user name (comment) <e-mail address>

All that is required within a key ID is enough unique information to locate the desired key. For example, if a key is created using a user name of **Dick Tracy** and there are no other keys on a keyring with a user name containing **Dick**, then only the first name is required as part of the key ID.

Important

The required key creation elements vary between PGP implementations. For example, not all PGP keys contain a comment element. Some PGP implementations may permit shorter user names and some may not require a passphrase.

PGP Passphrases

A passphrase is a collection of words and characters used by PGP cryptography when you create a public/private key file set, whenever encrypted files are signed, and when files are decrypted.

Passphrases differ from passwords only in length. Passwords are usually short -- six to ten characters. Short passwords are acceptable for logging on to a computer system, but they are not safe for use with encryption systems. Passphrases are usually much longer -- up to 100 characters or more. Their greater length makes passphrases more secure.

Picking a good passphrase is one of the most important things you can do to preserve the privacy of the files you encrypt using PGP. A passphrase should be:

- Known only to you
- Long enough to be secure
- Hard to guess -- even by someone who knows you well
- Easy for you to remember and type accurately if necessary
- Use a combination of upper and lower case characters and digits (for example: TesT03PhrasE)

Important

FileLink secures your passphrase by saving it in an encoded format in the Windows registry along with its other settings. The passphrase is also never displayed in the FileLink console window nor written to any log file. But be aware that *it does appear in clear-text* in a script file. Therefore, the method of specifying your passphrase during configuration is the most secure.

PGP ASCII Armoring

PGP key files and encrypted files may be saved in a format referred to as *ASCII armored*. This format is an encrypted representation of a file consisting entirely of printable ASCII (or text-mode only) characters.

Files in this format contain no binary values, and therefore may be easily sent as part of e-mail messages and visually examined using programs like Notepad.

Files saved in this format are approximately 30% larger than their non-armored counterparts. When decrypted, both armored and non-armored files reproduce an identical original.

PGP Digital Signatures

PGP encrypted files may contain *digital signatures*. Digital signatures enable the recipient of a file to verify the authenticity of the information's origin, and also verify that the information has not been tampered with. A digital signature also prevents the sender from claiming that he or she did not actually send the information. Therefore, A digital signature serves the same purpose as a handwritten signature in that it attests to the contents of the information as well as to the identity of the signer.

FileLink's Implementation of PGP

First and foremost, FileLink is a scriptable file transfer package and not a comprehensive PGP encryption tool, but FileLink does provide the basic functionality required to utilize PGP.

FileLink is integrated with the open-source package named GnuPG from which FileLink obtains its PGP encryption/decryption engine.

The following functionality is provided by the FileLink Configurator:

- The creation of private and public key(s) and the creation of associated "keyrings"
- The ability to export public keys to ASCII armored and non-ASCII armored files from a keyring to be shared with others (ASCII armored key files are plain-text files that are easily shared via e-mail or other means; non-armored files are in a binary format)
- The ability to import key(s) to a keyring
- The ability to delete key(s) from a keyring
- The ability to use existing FileLink or GPG keyrings

The following functionality is provided by script commands within FileLink itself:

- The ability to encrypt and optionally digitally sign files (the [PGPENCRYPT](#) command)
- The ability to decrypt files for which you have a corresponding public key (the [PGPDECRYPT](#) command)
- The ability to import public keys to a keyring (the [PGPIIMPORT](#) command)

PGP has the added advantage of compression. Much like a zip file, files that are encoded using PGP are also compressed. Of course, file(s) are expanded and restored to their original state when they are decrypted by their intended recipient.

PGP is a complex encryption technology and the preceding paragraphs barely scratch the surface as an introduction. If you are new to PGP specifically and public key cryptography in general, we strongly recommend doing some independent study on the subject to make sure that you understand the advantages and dangers associated with the encryption and decryption of files.

Extending FileLink's PGP Functionality

In some cases, you may be required to encrypt or decrypt a file using a GnuPG option that is not directly implemented by FileLink. In order to do so, both the PGPENCRYPT and PGPDECRYPT commands support the **/gpgopt** option which allows you to specify any necessary GnuPG option(s). The **/gpgopt** option should be used by advanced users only and FileLink interoperability with all GnuPG options is not guaranteed.

The following example encrypts a file and specifies a supported option not directly supported by FileLink.

```
PGPENCRYPT "in" "out" /user="Dick" /gpgopt="--force-v3-sigs"
```

Multiple GnuPG options may be passed using **/gpgopt**. When doing so, separate each complete option with a semi-colon as shown below.

```
PGPENCRIPT ... /gpgopt="--force-v3-sigs;--no-verbose"
```

Be sure to always precede each GnuPG option with two dashes.

Troubleshooting FileLink's PGP Functionality

For advanced troubleshooting, the PGPENCRIPT and PGPDECRYPT commands support the **/gpglog** option which results in commands and responses to and from GnuPG (gpg.exe) being written to a log file.

The following example encrypts a file and writes to a log file named "encrypt.log".

```
PGPENCRIPT "in" "out" /user="Dick" /gpglog="encrypt.log"
```

If a fully qualified file name is not specified, the log file will be created in the current FileLink working folder. If the file exists, it will be appended to. Delete the file before each PGPENCRIPT or PGPDECRYPT command if you want only a single command to be logged.

The interpretation of the resulting log file is left to the user or the file may be requested by FileLink technical support to assist you with a particular problem.

Backing Up Your Keyring File Set

It is very important that you backup your PGP keyring. If a keyring is lost or destroyed, it will no longer be possible for you to decrypt files sent to you using any public key that you may have distributed. And recipients of encrypted files from you will not be able to decrypt your files until you provide them with a new public key.

A PGP keyring used by FileLink actually is a set of three files. They will reside in FileLink's working folder, or in a folder that you designate. The files are:

- pubring.gpg
- secring.gpg
- trustdb.gpg

To protect against loss of your encrypted data, it is vital that you make backup copies of these files as often as you add, import, or delete keys from your keyring.

Using PGP With FileLink Step-By-Step Guide

PGP setup is done using the FileLink Configurator under the [Configure PGP](#) tab.

In order to use PGP with FileLink, you must have your own PGP public/private key-pair. You can either create a new one or you can import one.

In order to encrypt a file with FileLink, you must have imported the recipient's public key *before* encryption.

In order to decrypt a file with FileLink, you must have exported your public key and sent it to the person who will be encrypting the file *before* they encrypt it.

Creating a New Key-Pair

You may [create a key-pair](#) using the FileLink Configurator by following these steps:

- Click the 'Create Key' button.
- Make sure the 'Create Empty Keyring' box is not selected.
- You must enter a user name; an email address and/or comment field are optional.
- Enter the size. (default is 1024)
- Enter an expiration date. (default is never)
- Enter a Passphrase and verify it.
- Save the passphrase to use it without specifying it in your script (for added security).
- Select the folder where the keyring will be created.
- Click 'OK' to create the keyring and add the key-pair to this keyring.

Importing a Key-Pair

You may [import a key-pair](#) using the FileLink Configurator by following these steps:

- Click the 'Create Key' button.
- Make sure the 'Create Empty Keyring' box is selected.
- Select the folder where the keyring will be created.
- Click 'OK' to create the empty keyring.
- Click the 'Manage Keys' button.
- Click the 'Import...' button.
- Browse to the folder where your exported PGP keyring is located.
- Select the file and click 'OK'.
- Your key should now be imported and ready to use.

Any key from PGP Desktop (a product from PGP Corporation) must already have been created and exported, please see the documentation for PGP Desktop for instructions on how to create/export a key-pair using this product.

Importing a Public Key

You may [import a public key](#) using the FileLink Configurator by following these steps:

- Click the 'Manage Keys' button.
- Click the 'Import...' button.
- Browse to the folder where the public key file is located.
- Select the file and click 'OK'.
- The key should now be imported and ready to use.

A public key must have been exported and received in order to import the public key.

Encrypting a File

This is done using [PGPENCRIPT](#) script command. For example:

```
PGPENCRIPT "file_to_encrypt" "destination_encrypted_file" /  
user="Recipient1"
```

You must have already imported the recipient's public key.

Decrypting a File

This is done using [PGPDECRYPT](#) script command. For example:

```
PGPDECRYPT "encrypted_file" "c:\temp\output"
```

You must have already exported your public key and sent it to the recipient.

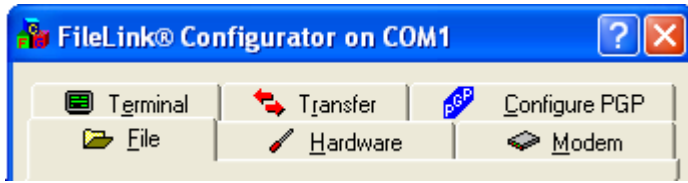
Using the FileLink Configurator

The FileLink® Configurator is used to select which COM port(s) to use. You configure the COM port for async modem connections and otherwise setup how you want to use FileLink.

The configurator is run once during installation and is accessible from a button on the FileLink toolbar. You must run the configurator at least once before attempting to use FileLink on a particular connection. The first time a connection is configured you are given the option to create a **Start** menu shortcut and a desktop shortcut to run FileLink on this connection.

When the configurator is run from the FileLink toolbar button, any settings you change do not take effect until a script file is executed (or unless FileLink is restarted.) For example, if you change the baud rate and protocol while running the configurator, the FileLink status bar does not reflect the newly defined values until you run a script file.

The FileLink Configurator is organized around six tabs. Click on the tabs for a brief description of tab. Follow the hyperlinks below for more detailed information.



[File Configuration](#)

[Hardware Configuration](#)

[Modem Configuration](#)

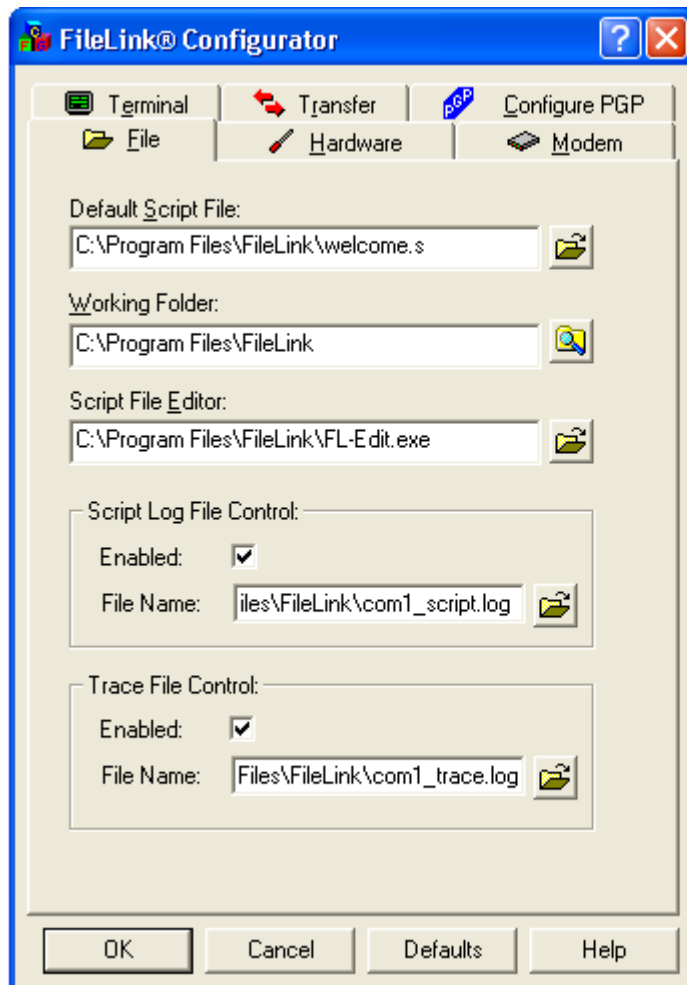
[Terminal Configuration](#)

[Transfer Configuration](#)

[PGP Configuration Tab](#)

File Configuration

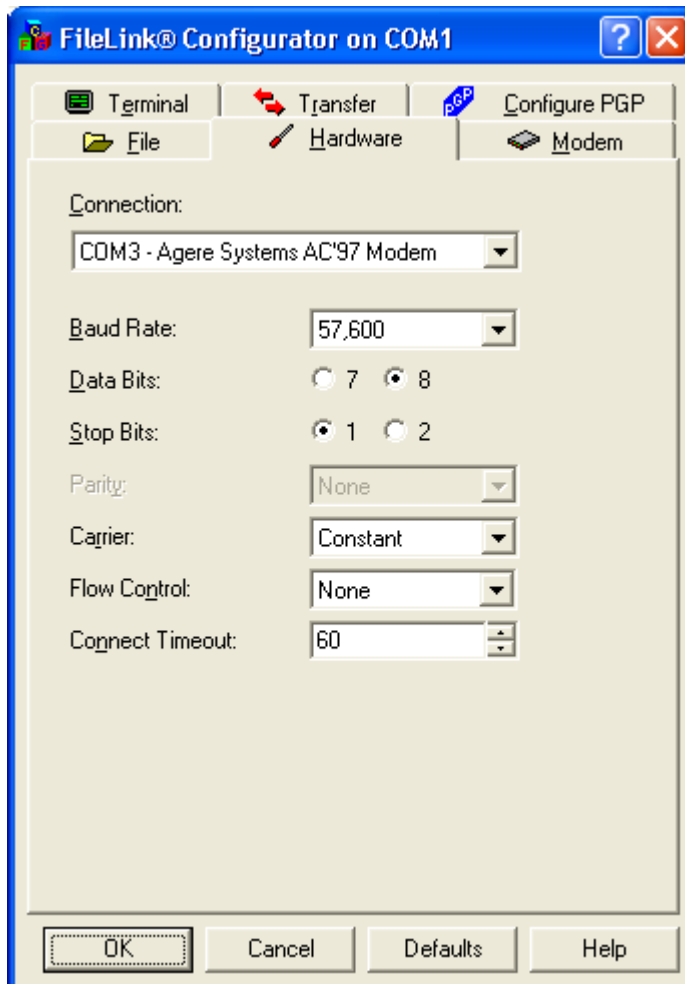
When you click on the **File** tab the following dialog is displayed. Click on any of the controls in the body of the dialog for more information on each configuration item contained therein.



Hardware Configuration

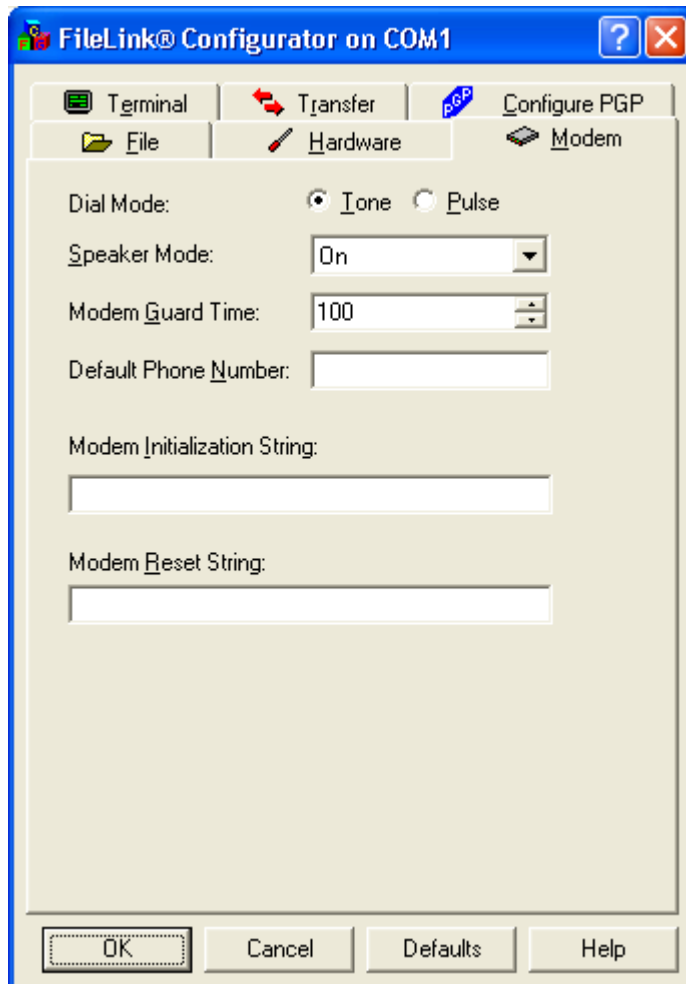
When you click on the **Hardware** tab one of the following dialog is displayed corresponding to a selection of a COM port for async file transfers. Click on any control within the body of the dialog for more information on a particular configuration item.

Sample dialog for **COM port** selection:



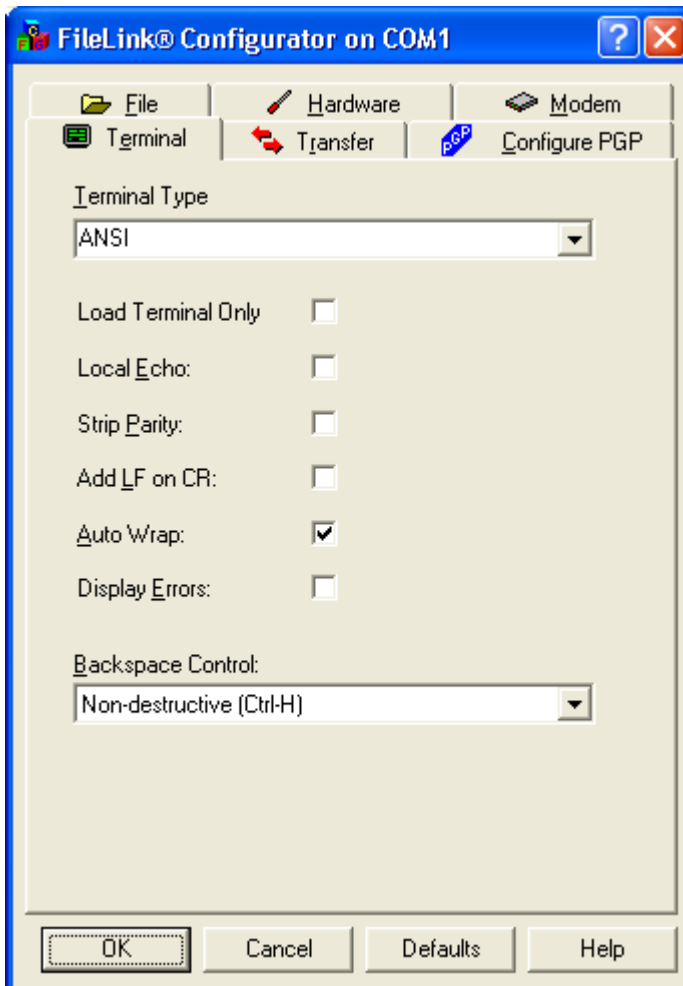
Modem Configuration

When you click on the **Modem** tab the following dialog is displayed. Click on any of the controls in the body of the dialog for more information on each configuration item contained therein.



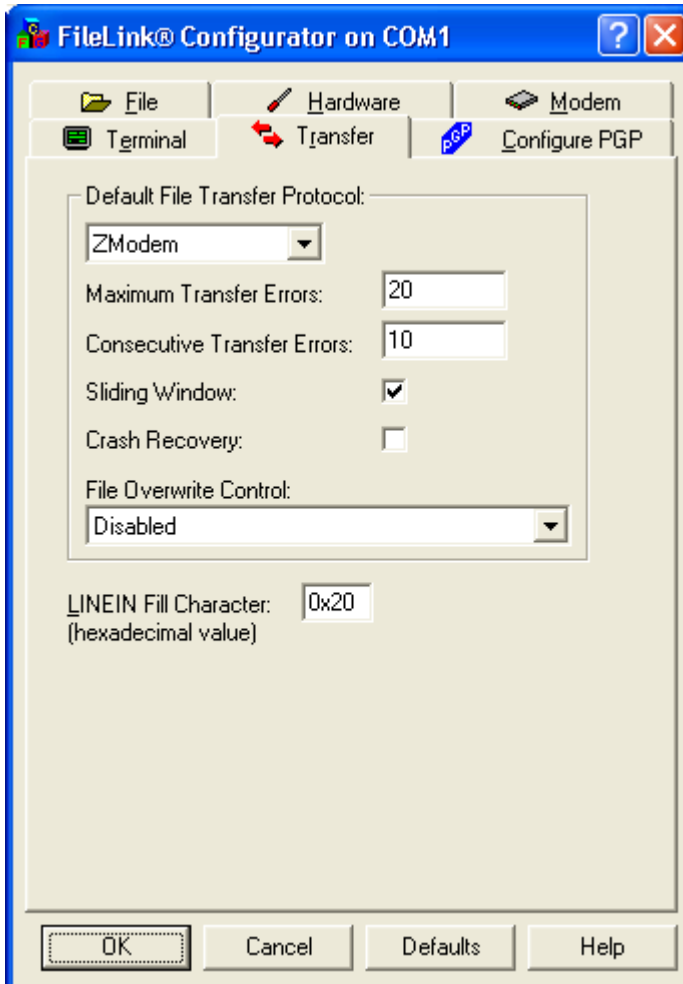
Terminal Configuration

When you click on the **Terminal** tab the following dialog is displayed. Click on any of the controls in the body of the dialog for more information on each configuration item contained therein.



Transfer Configuration

When you click on the **Transfer** tab the following dialog is displayed. Click on any of the controls in the body of the dialog for more information on each configuration item contained therein.



File Transfer Protocol Configuration

FileLink supports six different modem file transfer methods or protocols. These six protocols are configured by clicking the Transfers tab of the FileLink Configurator.

The configuration details for each method or protocol is presented below.

[ASCII Transfers](#)

[Kermit Transfers](#)

[Xmodem Transfers](#)

[Xmodem1K Transfers](#)

[Ymodem Transfers](#)

[Zmodem Transfers](#)

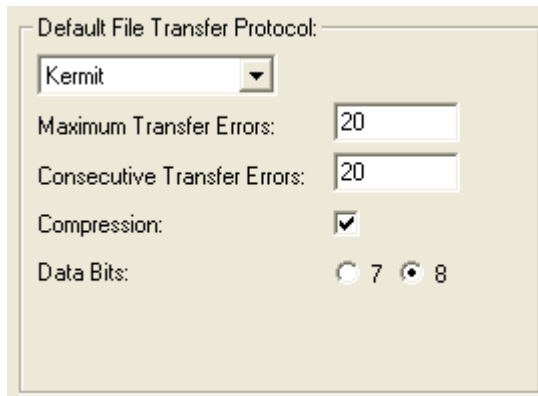
ASCII File Transfers

There are no special configuration settings for ASCII file transfers. This type of transfer is not block oriented, does not support binary files, and there is no error recovery. When receiving, there is no protocol to tell when the remote system finished sending a file, so FileLink waits approximately 10 seconds before terminating a command after at least one character has been received.

Kermit File Transfers

Kermit is an error-correcting, block-oriented protocol. Kermit transfers start with a header block containing file name and date-time stamp for the file, followed by the contents of the file. This continues for as many files as the sender transmits. By default, Kermit transfers performs Run-Length-Encoding (RLE) data compression that may reduce transmission time.

The configuration settings for this protocol in the FileLink Configurator appear below. Click on a particular setting control for more information.



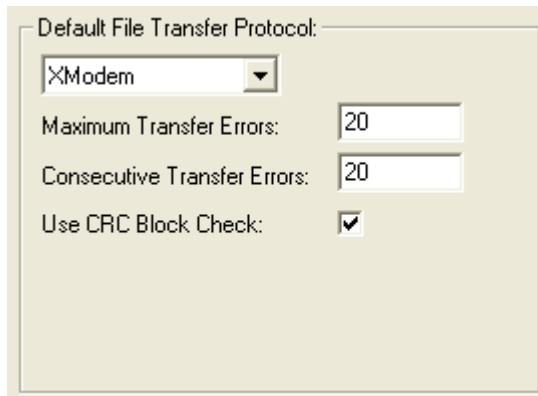
The image shows a configuration window titled "Default File Transfer Protocol:". The settings are as follows:

- Default File Transfer Protocol: Kermit (selected in a dropdown menu)
- Maximum Transfer Errors: 20 (text input field)
- Consecutive Transfer Errors: 20 (text input field)
- Compression: (checked)
- Data Bits: 7 8 (radio buttons)

Xmodem File Transfers

Xmodem is an error-correcting, block-oriented protocol. The block size is 128 bytes. To use this protocol, you must configure the COM port to use 8-data bits. By default, Xmodem uses the 16-bit CRC block check method.

The configuration settings for this protocol in the FileLink Configurator appear below. Click on a particular setting control for more information.



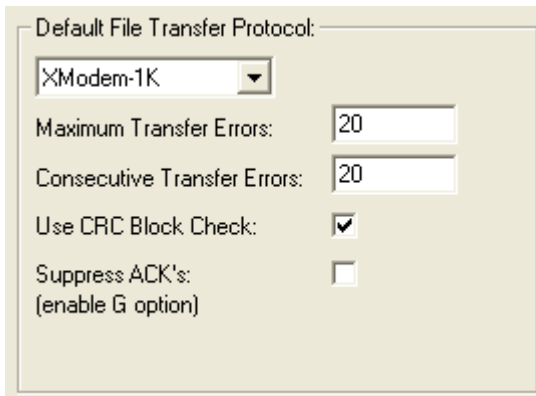
The screenshot shows a configuration window titled "Default File Transfer Protocol:". It contains the following settings:

- Default File Transfer Protocol: XModem (selected in a dropdown menu)
- Maximum Transfer Errors: 20 (text input field)
- Consecutive Transfer Errors: 20 (text input field)
- Use CRC Block Check: (checkbox)

Xmodem1K File Transfers

Xmodem1K is an error-correcting, block-oriented protocol. The block size is 1024 bytes. To use this protocol, you must configure the COM port to use 8-data bits. By default, Xmodem1K uses the 16-bit CRC block check method.

The configuration settings for this protocol in the FileLink Configurator appear below. Click on a particular setting control for more information.



Default File Transfer Protocol:

XModem-1K

Maximum Transfer Errors: 20

Consecutive Transfer Errors: 20

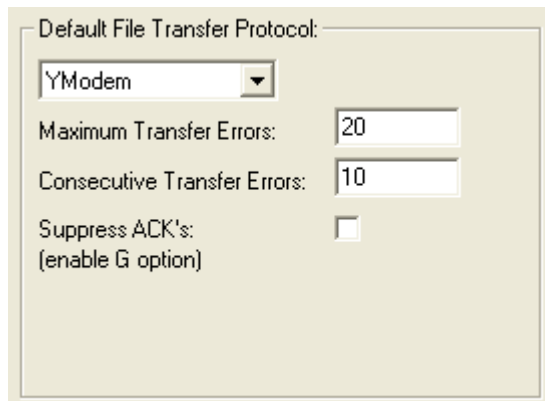
Use CRC Block Check:

Suppress ACK's:
(enable G option)

Ymodem File Transfers

Ymodem is an error-correcting, block-oriented protocol. Ymodem transfers start with a header block containing file name and date-time stamp for the file, followed by the contents of the file. This continues for as many files as the sender transmits. To use this protocol, you must configure the COM port to use 8-data bits. By default, Ymodem uses the 16-bit CRC block check method.

The configuration settings for this protocol in the FileLink Configurator appear below. Click on a particular setting control for more information.



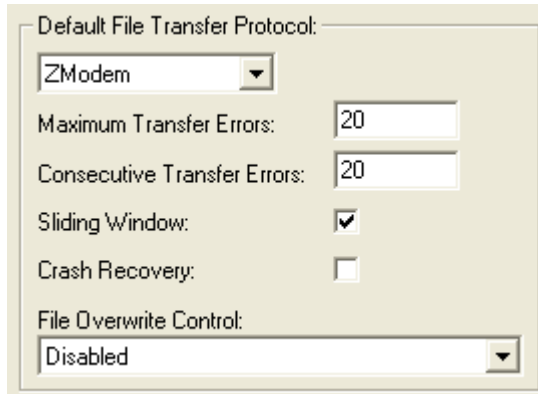
The image shows a configuration window titled "Default File Transfer Protocol:". It contains the following settings:

- A dropdown menu set to "YModem".
- A text input field for "Maximum Transfer Errors" with the value "20".
- A text input field for "Consecutive Transfer Errors" with the value "10".
- A checkbox for "Suppress ACK's: (enable G option)" which is currently unchecked.

Zmodem File Transfers

Zmodem is an error-correcting, block-oriented protocol. Zmodem transfers start with a header block containing file name and date-time stamp for the file, followed by the contents of the file. This continues for as many files as the sender transmits. To use this protocol, you must configure the COM port to use 8-data bits. Zmodem uses the 32-bit CRC block check method with block sizes of up to 1024 bytes.

The configuration settings for this protocol in the FileLink Configurator appear below. Click on a particular setting control for more information.

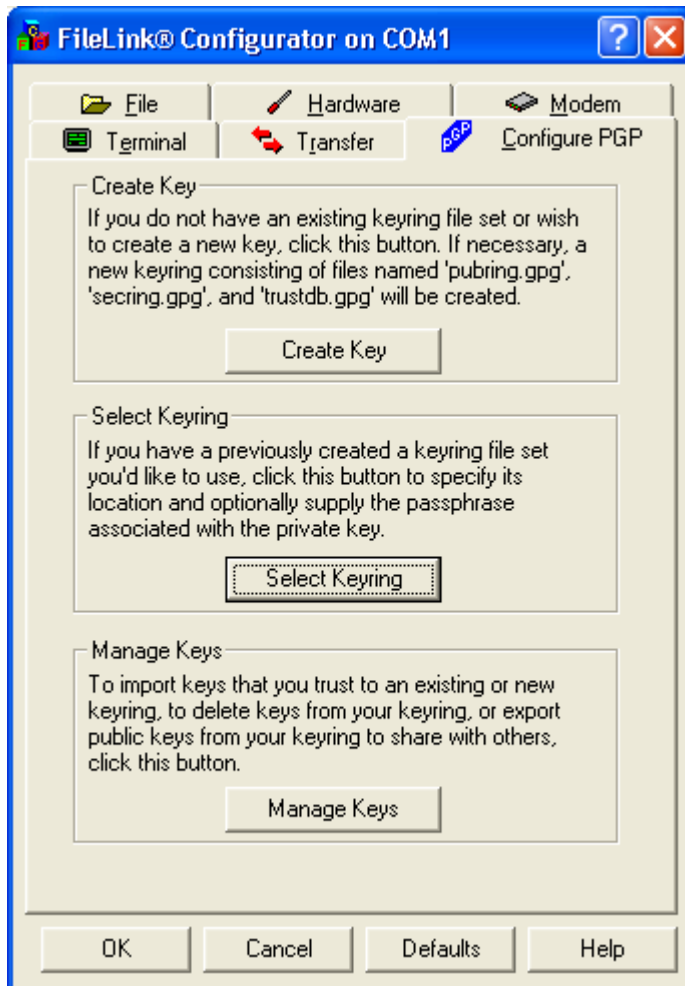


The image shows a configuration dialog box titled "Default File Transfer Protocol:". It contains the following settings:

- Default File Transfer Protocol: ZModem (selected in a dropdown menu)
- Maximum Transfer Errors: 20 (text input field)
- Consecutive Transfer Errors: 20 (text input field)
- Sliding Window: (checkbox)
- Crash Recovery: (checkbox)
- File Overwrite Control: Disabled (selected in a dropdown menu)

PGP Configuration

When you click on the **Configure PGP** tab the following dialog is displayed. Click on any of the controls in the body of the dialog for more information on each configuration item contained therein.



PGP Configuration: Create Key

When you click on the **Create Key** button the following dialog is displayed. This dialog has controls that allow you to create a new PGP key in an existing FileLink or GPG keyring, or to create a key in a new keyring. Keyring files have an extension of **.gpg**. (GPG is a commonly used Open Source implementation of PGP cryptography which is utilized by FileLink.)

Click on any of the controls in the body of the dialog for more information on each configuration item contained therein. For more information, see [Using PGP With FileLink Step-By-Step Guide](#).

Create Key ? X

Create Empty Keyring

Key User Name:

Key Comment:

Key E-mail Address:

Key Size (Bits): 2048 1024 768

Key Expires: Never 3/13/2004 ▼

Key Passphrase: Show Typing

Key Passphrase Verification:

Save Passphrase

Keyring Folder:

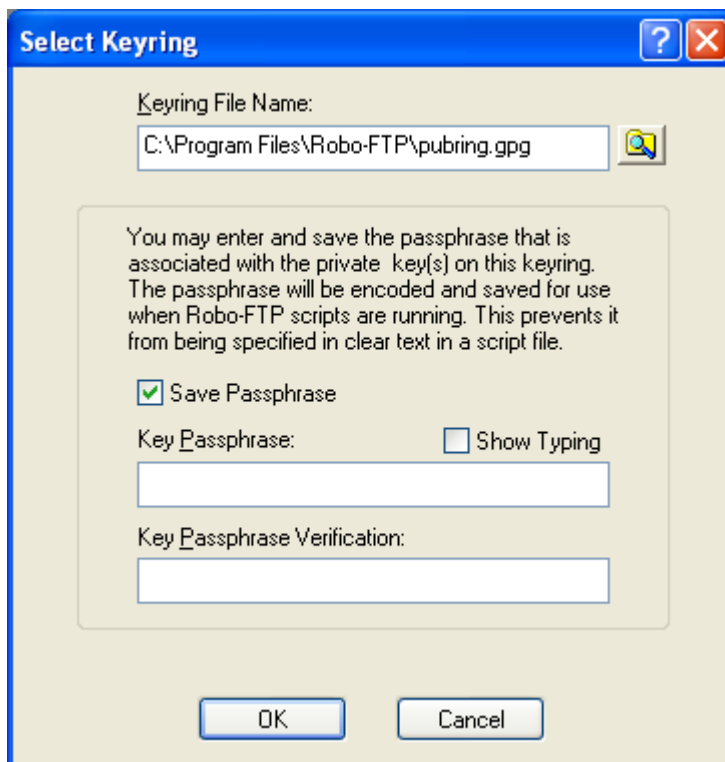
OK Cancel

PGP Configuration: Select Key

When you click on the **Select Key** button the following dialog is displayed. This dialog has controls that allow you to select an existing FileLink or GPG keyring. Files with an extension of **.gpg** are displayed in the file selection control. (GPG is a commonly used Open Source implementation of PGP cryptography which is utilized by FileLink.)

PGP keys created by other PGP packages may be used, but FileLink requires you to [create a keyring](#) and then import keys exported from these other products.

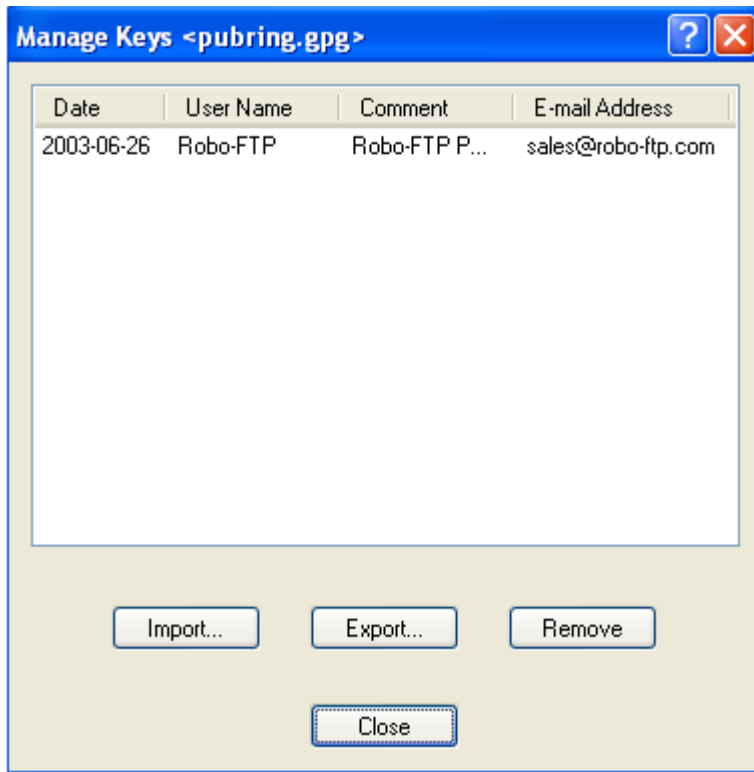
Click on any of the controls in the body of the dialog for more information on each configuration item contained therein.



PGP Configuration: Manage Keys

When you click on the **Manage Keys** button the following dialog is displayed. This dialog has controls that allow you import, export, and delete keys from the current keyring. The current keyring is either the [last keyring created](#) or the [last keyring selected](#).

Click on any of the controls in the body of the dialog for more information on each configuration item contained therein. For more information, [Using PGP With FileLink Step-By-Step Guide](#).



User vs. Machine Configuration

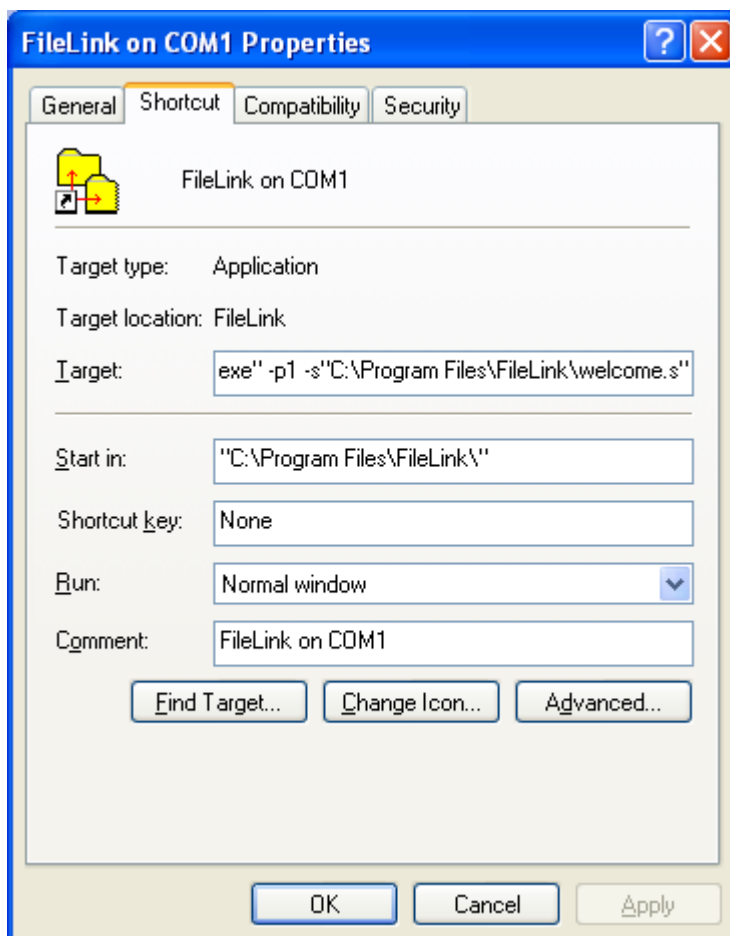
When running the FileLink Configurator, the configuration settings are saved in the Registry in association with the current user logged into Windows. (Technical reference: the HKEY_CURRENT_USER Registry location is used.) This permits each user to have unique configuration setting(s) for FileLink.

In most cases this is fine. In some environments, however, you may wish to have a single configuration for all users on a given machine. (Technical reference: the HKEY_LOCAL_MACHINE Registry location is used.)

To accommodate both possibilities, FileLink first looks in HKEY_CURRENT_USER for configuration settings. If the settings are not found, FileLink next looks in HKEY_LOCAL_MACHINE.

By default the FileLink Configurator writes configuration settings to HKEY_CURRENT_USER. To have it use HKEY_LOCAL_MACHINE, you must create a shortcut for the configurator and add the **-I** switch within the **Target** field in the short properties dialog. When you run the configurator by way of this shortcut, you are creating a single configuration for FileLink for all users.

See the example below on how to create the shortcut.

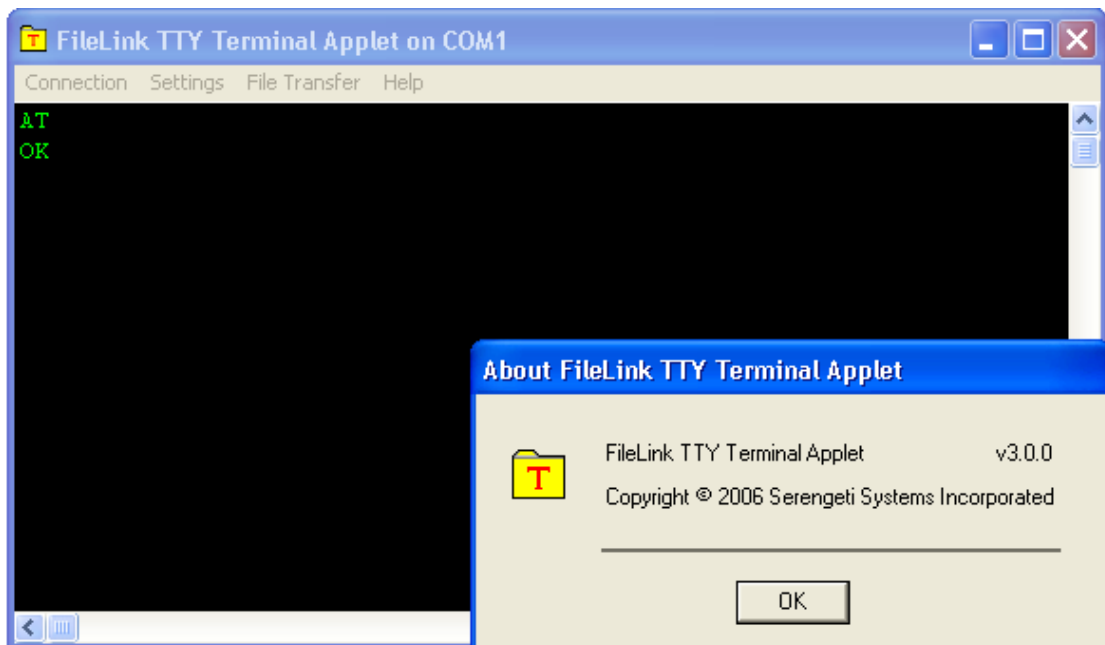


Using the FileLink TTY Terminal Applet

The FileLink TTY Terminal applet is a simple TTY style terminal emulator. The applet is started by clicking on the **Run TTY Terminal** menu item or toolbar button, or by executing the **TERMINAL** script command. You may switch freely between the TTY Terminal applet and the FileLink script environment.

Each environment shares the same communications session, so you may connect to or disconnect from the remote system, for example, from either a script file or from within the TTY Terminal applet. The same set of configuration settings apply to both environments.

The TTY Terminal applet is useful to understand how a communications session with a particular remote system takes place so that it can be automated with a script file.



The TTY Terminal applet is menu-driven. The menus are organized as follows:

[TTY Terminal Connection Menu](#)

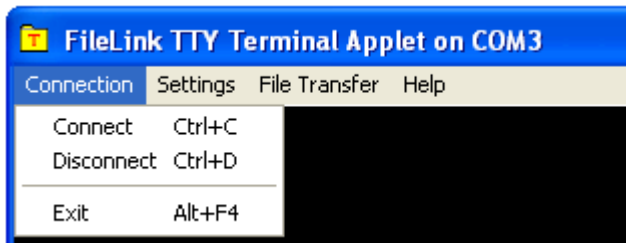
[TTY Terminal Settings Menu](#)

[TTY Terminal File Transfer Menu](#)

[TTY Terminal Help Menu](#)

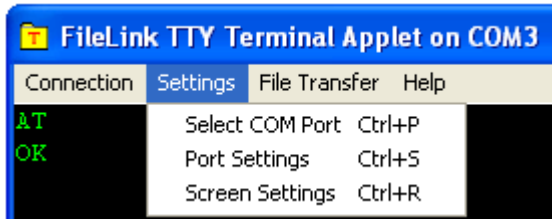
Terminal Connection Menu

The Terminal **Connection** menu is shown below. The actions off this menu support opening and closing the configured COM port, and permit exiting back to the FileLink main window. Click on a menu item for information on its function.



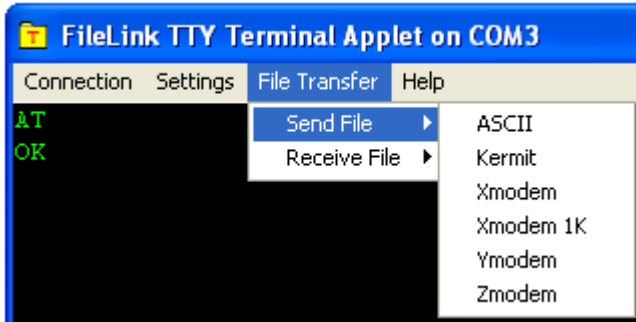
Terminal Settings Menu

The Terminal **Settings** menu is shown below. The actions off this menu support changing the current COM port, changing some of the settings of the COM port and the Terminal environment (baud rate, parity, line wrapping, etc.), and changing the Terminal type (e.g., ANSI, DEC, Wyse, etc.), screen size, fonts, and colors. Click on a menu item for information on its function.



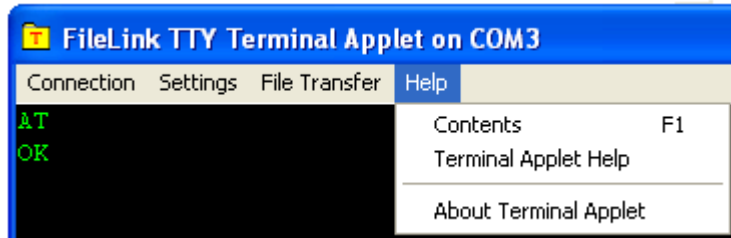
Terminal File Transfer Menu

The Terminal **File Transfer** menu is shown below. The actions off this menu support the selection of a file transfer protocol and then subsequently the sending or receiving of one or more files. Click on a menu item for information on its function.



Terminal Help Menu

The Terminal **Help** menu is shown below.



The FileLink Script File Editor

The FileLink script file editor is a specially designed text editor for editing FileLink script files. The editor provides for the easiest possible creation and maintenance of script files. Major features include:

- Context sensitive help for all FileLink script commands
- Color syntax highlighting of FileLink script commands
- Automatically capitalize FileLink script commands as you type
- Edit multiple script files and save (and reload) all at once as a "session"
- Find in files function that automatically searches script files (files with **.s** extension)
- Optional tabbed interface when editing multiple files
- Optional line numbers

This is the FileLink editor main screen:


```

11  IFNFILE "first_welcome.txt" GOTO already_run
12  MESSAGEBOX "The Automated File Transfer Solution." "Welcome to Fi
13  SET greeting = "Thank you, "
14  PROMPT your_name "What is your first name?" "Welcome to FileLink"
15  IFERROR= $ERROR_PROMPT_CANCELLED GOTO finish
16  IFSTRCMP your_name "" GOTO no_name
17  SET greeting &= your_name
18  SET greeting &= "."
19  DISPLAY greeting
20  MESSAGEBOX greeting
21  :no_name
22  MESSAGEBOX "Please read the FileLink help file for information on
23  MESSAGEBOX "You may use the Terminal or FTP applets to connect im
24  ASK "Show this welcome next time you run FileLink?" "Welcome to F
25  IFYES GOTO finish

```

FileLink script command syntax is highlighted in color as shown below:

Blue	Script commands
Violet	Script command options (e.g., <i>/append</i>); command operators (e.g., &)
Black	Variables and strings
Green	Comments
Red	Predefined error variables (e.g., <i>\$ERROR_SUCCESS</i>); numeric constants (e.g., <i>NATO 30</i>)
Bold Black	Script labels

The editor is launched from a FileLink menu or the toolbar and automatically opens the currently active script file.

Script Programming

To accommodate automated and unattended operations it is designed for, FileLink provides a powerful script language. See [Script Language Features](#).

Script files can be executed directly from a FileLink shortcut on the desktop or when selected interactively while FileLink is running .

Script files are normally produced with the FileLink Script File Editor . FileLink has toolbar buttons to launch the editor (or other editor of choice) for this purpose. Unless otherwise specified, script files are assumed to have an extension of **.s**.

See also: [Script File Commands](#), [Sample Script Files](#)

Script File Command Arguments

Script commands consist of an opcode followed by one or more arguments. An argument falls in one of the following types.

- an [alpha-numeric](#) string (enclosed in single or double quotes)
- a [variable](#) (names starting with alphabetic character, %, or \$ symbol)
- a [command option](#) (starting with / symbol)
- a numeric value

See also: [Script Language Features](#), [Script Commands](#)

Script File Alphanumeric Constants

Alphanumeric constants define file names and other character strings that may be used in script commands directly or assigned to script variables. Alphanumeric constants are always enclosed by single or double quotation marks. Examples of alphanumeric constants are:

```
SENDFILE 'c:\My Data\Update Inventory.dbf'  
COPY "file2" "file1"  
SET filename = "filepath"
```

Whenever you are using a file name in a constant or variable, FileLink always assumes the file is located in the working folder. If you want to reference a file elsewhere, you should always use the file's full path name.

Script File Numeric Constants

Numeric constants may be used in appropriate script commands or assigned to script variables. Numeric constants differ from alphanumeric constants in that the enclosing single or double quotation marks are optional. Examples of numeric constants are:

```
SETNUM x = 1  
SETNUM y = "2"      ; ; same as y = 2  
NATO 60
```

Script File Variables

Variable arguments are internally or user defined string or numeric values that may be used in script commands where an alphanumeric or numeric value is expected. Variables may be created and assigned a value by using the [SET](#) or [SETNUM](#) script commands, or are automatically created and assigned a value by the FileLink script processor when used in certain script commands.

Script variables begin with at least one alphabetic character and may be up to 255 characters in length. Variables are not case sensitive. For example, **\$abc** and **\$ABC** are the same variable. Up to 4096 variables may be assigned at one time.

Once a variable is assigned, it remains defined for the duration of an active script file or until it is unassigned with the SET command. The values assigned to script variables may be up to 1020 characters in length. Examples of alphanumeric variables are shown below:

```
SETSET phone_number = "555-1212"  
DIAL phone_number  
SETSET file_name = "c:\anyfile"  
DELETE filename
```

FileLink script files may perform substring manipulation on variables. The [SETEXTRACT](#), [SETLEFT](#), [SETMID](#), [SETRIGHT](#), and [SETSUBSTR](#) commands allow extraction of a delimited substring, substrings from the left to right from mid string, right to left, and to find the occurrences of a substring respectively.

FileLink script files may also perform basic arithmetic manipulation on numeric variables. The [INC](#) and [DEC](#) commands allow incrementing and decrementing of numeric strings while the SETNUM command permits adding, subtracting, multiplication, and division. Examples of numeric variables are shown below:

```
SETNUM x = 1  
SETNUM e = m x c x c ; (hint: e = mc2)  
INC x
```

There are a number of internal script variables assigned by FileLink to make your script development easier and more powerful. These are described their own [Internal Script Variables](#) section.

See also: [Using Shortcut Target Arguments in Script Files](#)

Script File Command Options

Command options may be present on certain script file commands following the opcode and any arguments that may be required. Options always begin with the / symbol and must not contain any embedded spaces. For instance, in an option such as **/baudrate=9600**, you must not put spaces on either side of the equal sign. Many commands support multiple options. Examples of command options are below:

```
RCVFILE /flush /timeout=0  
PROTOCOL "xmodem" /use_checksum  
USEPORT "COM1" /baudrate=9600 /parity=none /stopbits=1
```

Labels in Script Files

To facilitate conditional and unconditional branching in FileLink script files, a label is used to define the destination of a branch. Labels always begin with a colon, must not exceed 32 characters in length, and may appear anywhere within a line. FileLink supports up to a total of 320 unique labels at once.

Consider the following example of a valid label.

```
:top
```

Refer to [LOOPTO](#), [GOTO](#), [LOOPIF](#), and [IFERROR](#) for more details.

Comments in Script Files

Either a semi-colon or an asterisk may be used to denote the beginning of comment in a FileLink script file. A comment may be a separate line unto itself or be on the same line as a command.

Consider the following examples of valid script file comments.

```
; Connect with the remote system
CONNECT
* this is a comment line *
LINEOUT "hello world"
DIAL "555-1212" ;; connect with directory assistance?
```

Debugging Script Files

FileLink provides several commands to specifically facilitate the debugging of script files. These commands are:

BREAK	- Set a script file breakpoint
GO	- (Re)run the currently selected script file from beginning
RESUME	- Resume execution of a script file stopped at a breakpoint
STOP	- Exit from the break state

There is also a control named Enable Breakpoints under the **Scripts** menu which controls whether or not BREAK command(s) are recognized when a script is running.

When debugging, place the BREAK command at strategic places in the flow of execution of a script file to be debugged. When the FileLink script processor encounters a BREAK, script execution will be suspended.

FileLink is now in the *break state* and control is returned to the console window. Here you can view variable values, for example, by typing [DISPLAY](#) commands into the console command line, reassign variable values by typing [SET](#) or [SETNUM](#), or perform other tasks that may assist you in finding problems in your script logic.

Script execution may be resumed by typing the [RESUME](#) command into the console command line or by clicking the Skip To Next Command (**Ctrl + N**) toolbar button. The script may be stopped by typing the [STOP](#) command or by clicking the **Stop** toolbar button. To restart the script (perhaps after editing it to make corrections), type the [GO](#) command or click the Rerun Script File (**Ctrl + R**) toolbar button.

When your script has been thoroughly debugged, either edit your script and remove all the BREAK commands or clear the Enable Breakpoints control in the **Scripts** menu and BREAK commands will be ignored on subsequent runs of the script.

Using Variables in Command Options

FileLink script command options may not be specified using variables directly. The following is an example of a script command with an option:

```
USEPORT /parity=none
```

In this case `none` is a literal string (without quotation marks) and it cannot be replaced with a variable. However, there is an indirect way that this can be done. This involves the building of a complete command in a variable and using the [PERFORM](#) script command to run it.

For example, let's assume that you wish to set some COM port parameters that are not known until the script is run. The command might look like the following:

```
USEPORT /baudrate=9600 /parity=none
```

Say that the script prompts the user for the baudrate and parity settings and these are assigned to script variables `baud` and `par`. You might be inclined to try something like:

```
USEPORT /baudrate=baud /parity=par ;Wrong!
```

This will result in wrong baud rate and parity values (the variable names, not their values) being provided to the `USEPORT` command. The supported method would be:

```
SET cmd = "USEPORT /baudrate=" & baud & " /parity=" & par  
PERFORM cmd
```

In another example, let's assume that you wish to control the reading of text file using the `READFILE` command with `/record` option, but the record number varies and you wish to use a variable to specify which record to read. Normally the command would look something like the following:

```
READFILE "datafile" datarecord /record=1
```

The `/record` option accepts a numeric constant only. Using a variable, you would like do something like:

```
SETNUM rec = 10  
READFILE "datafile" datarecord /record=rec ;WRONG!!
```

The supported method is:

```
SET file = "datafile"  
SETNUM rec = 10  
SET cmd = "READFILE " & file & " /record=" & rec  
PERFORM cmd
```

This method may be used on any FileLink script command that uses options.

Using Functions

Functions are a handy way to re-use a sequence of commands that may be otherwise be repeated in a script file. Like functions or subroutines in other programming environments, FileLink script file functions make script development easier and results in more easily read and maintained scripts.

A script may define up to 256 unique functions. Functions may call other functions but the nesting depth of embedded functions is limited to 32 calls. Functions may not be called recursively (i.e., a function cannot call itself).

Functions must be declared before they can be used. FileLink provides two script directives to create what is referred to as a *function declaration section* within a script file. This section must be at the top of the script file. It must also be in the main script which is to say a *function declaration section* cannot be in a script that is invoked by way of a [CALL](#) script command.

```
;; an example function declaration section
BEGINFUNCTIONS
;; function(s) are defined here
ENDFUNCTIONS
```

Functions are defined within the *function declaration section* in the following manner.

```
;; an example function
FUNCTION MyFunction
;; function body is here
ENDFUNCTION
```

Putting it all together.

```
;; an example function declaration section
BEGINFUNCTIONS
;; function(s) are defined here
;; an example function
FUNCTION MyFunction
;; function body is here
ENDFUNCTION
ENDFUNCTIONS
```

There can be multiple functions.

```
;; an example function declaration section
BEGINFUNCTIONS
;; function(s) are defined here
;; example function #1
FUNCTION MyFunction1
;; function body is here
```

```

ENDFUNCTION
;; example function #2
FUNCTION MyFunction2
;; function body is here
ENDFUNCTION
ENDFUNCTIONS

```

Calling a function is accomplished by simply using the function name as a command in the script file. The following calls the two functions declared above.

```

;; call my functions
MyFunction1
MyFunction2

```

Up to nine arguments may be passed to a function. The function must be declared showing the arguments and then called with the corresponding number of arguments.

```

;; an example function with two arguments
FUNCTION MyFunction arg1 arg2
;; function body is here
ENDFUNCTION

;; how to call the function with two arguments
SET var = "I am an argument"
MyFunction var "b"

```

Because functions are global, they are accessible from the main script and any script that are invoked with CALL or CHAIN script commands. They are also persistent which is to say they remain defined after a script terminates. It is possible to define functions in one script file and then call those functions when running other script file(s).

All variables in the FileLink script environment are global. FileLink functions support arguments. The variables that are created in association with function arguments are also global. This is demonstrated with the following script where both DISPLAY commands show the same value "a".

```

;; declare our function
BEGINFUNCTIONS
FUNCTION MyFunction arg1
DISPLAY arg1
ENDFUNCTION
ENDFUNCTIONS
;; script execution begin here
MyFunction "a"
DISPLAY arg1
STOP

```

Pay close attention to variables used in a script and make certain that uniquely named

variables are used whenever appropriate.

All of the previous examples show a single return point from a function. Specifically, all of the preceding functions return when there are no more command(s) in the function to perform. Multiple return points are possible in more complex scripts by using the RETURN script command.

```
;; declare our function
BEGINFUNCTIONS
FUNCTION MyFunction arg1
;; complicated operations
GOTO more_todo
RETURN
: more_todo
;; more complicated operations
RETURN
ENDFUNCTION
ENDFUNCTIONS
```

The RETURN statement is not always required. In the preceding function, the second RETURN statement is redundant. The script could be written as shown below.

```
;; declare our function
BEGINFUNCTIONS
FUNCTION MyFunction arg1
;; complicated operations
GOTO more_todo
RETURN
: more_todo
;; more complicated operations
ENDFUNCTION
ENDFUNCTIONS
```

In this case, the ENDFUNCTION directive is recognized as the end of the MyFunction function so the RETURN command is implied.

Finally, the RETURN statement allows for a numeric return code to be passed back. Upon return to the calling script, the return code may be tested using the [IFERROR](#) script commands and is saved in the [%lasterror](#) variable. The following example shows how different return points from a function may be indicated to the calling script.

```
;; declare our function
BEGINFUNCTIONS
FUNCTION MyFunction arg1
;; complicated operations
GOTO more_todo
RETURN 1
: more_todo
```

```
;; more complicated operations  
RETURN 2  
ENDFUNCTION  
ENDFUNCTIONS
```

Testing a return code would look something like this.

```
;; call my functions  
MyFunction1  
IFERROR= 1 GOTO from_Return1  
IFERROR= 2 GOTO from_Return2
```


Performing Variable Arithmetic and Numeric Comparisons

FileLink variables are always represented internally as string values. However, numeric operations can be performed on variables and strings as long as they contain only numeric (e. g., digits 0 - 9) values, or on numeric constants.

Numeric variables may be created using the SET or SETNUM command as shown below. When initializing variables, the only difference between SET and SETNUM is that SETNUM verifies that the variable is being set to a numeric value. Any previous value in the variable, regardless if it numeric or non-numeric, is discarded.

```
SET x = "100"  
SETNUM x = 100
```

Numeric variables may be incremented and decremented using the INC and DEC commands as shown below.

```
INC x  
DEC x
```

Basic integer arithmetic may also be performed on numeric variables using the SETNUM command. This command allows for addition, subtraction, multiplication, and division to be performed on two variables and/or numeric string constants as shown in the various statements below.

```
SETNUMSETNUM a = 100  
SETNUMSETNUM x = a + a      ;; addition  
SETNUMSETNUM x = a - 100   ;; subtraction  
SETNUMSETNUM x = 100 x a   ;; multiplication  
SETNUMSETNUM x = a / 100   ;; division
```

Numeric variables may be compared using one of the IFNUM commands as shown below.

```
IFNUM x 100 goto vars_equal  
IFNUM< x 101 goto var_less_than  
IFNUM> x 99 goto var_greater_than  
IFNUM< x 101 goto var_less_than_or_equal  
IFNUM> x 99 goto var_greater_than_or_equal  
IFNUM! x 99 goto vars_not_equal
```

Performing Date Arithmetic

Various dates are saved in internal FileLink script variables. For example, the current date is always saved in the **%date** variable.

FileLink provides two script commands, [DATEADD](#) and [DATESUB](#), that allow variables containing a date (i.e., any string in the format **mm-dd-yy** or **mm/dd/yy**) to be manipulated by adding or subtracting a specified number of days.

See also: [Using the %date, %datetime, and %time Variables](#)

Controlling Script Command Logging

Two different options are supported to suppress the echoing of script commands to both the console window and the log file. One option suppresses the command and any result message (s) while another suppresses the command echo but permits result message(s) to appear.

Begin a script command with an "at" symbol to completely suppress the command and its result message(s).

```
; Dial the remote system but suppress all output  
@DIAL remote_number
```

Begin a command with an exclamation point to suppress the command from being echoed but permit any result message(s) to be logged.

```
; Issue command but do not echo  
!PERFORM cmd
```

Scheduling Script Operation

FileLink provides two build-in methods to schedule when certain operations are performed. Simple daily scheduling is provided by the [PAUSE](#) command. More complex scheduling is provided by the [CRON](#) command. In some cases, the commands may be used together.

Simple scheduling amounts to delaying the execution of script commands for a specified period of time or until a specified time of day. For example, the following script command delays script execution until 2:00AM.

```
PAUSE /until=2:00
```

More advanced scheduling enables scripts to be run at on certain days of the week, days of the month, and/or at various times of day. The CRON command is used to this purpose. The CRON command utilizes predefined scheduling conditions (e.g., @daily) or a file (named "crontab.txt") that allows for multiple scheduling conditions. For example, the following script command runs a specified script file at midnight each night.

```
CRON "@daily" 'CALL "nightly.s"'
```

It is also possible to use the PAUSE and CRON commands together to achieve some of the same results. For example, since the @daily condition becomes active at midnight, you might want to have the script become active at 1:30AM instead of at midnight. To accomplish this, you could use the following script commands.

```
CRON "@daily"  
PAUSE /until=1:30  
CALL "nightly.s"
```

More complex scheduling is possible using the CRON script command in conjunction with the "crontab.txt" file. The format of this file is complex and not something the casual user needs to be concerned with. The [CronMaker](#) utility is provided with FileLink for the direct creation and modification of "crontab.txt" files.

Refer to [CRON](#) and [PAUSE](#) for more details.

Sending and Receiving E-mail in Script Files

FileLink is e-mail enabled. E-mail messages are composed and sent within a script using the [CREATEMAIL](#) and [SENDMAIL](#) script commands. E-mail messages are received using the [GETMAIL](#) command.

For FileLink e-mail to work, the PC must have an established network connection. To send e-mail, the PC must have access to an SMTP server. To receive e-mail, the PC must have access to a POP3 server. In both cases, the server name or IP address and any appropriate log on ID and password must be known when the script runs.

These commands work independently of any e-mail client that you may have installed on your PC. When receiving messages, FileLink can simply get the get next message available on the server, or search the subject line of all pending messages for specific value and only get this one message.

Messages received may be viewed and/or saved to a file. Optionally, messages may be left on the server. The subject line of the message is saved in a script variable for additional processing during script execution if necessary.

Listed below are some examples of how e-mail functionality may be utilized in FileLink scripts:

- Send a message to acknowledge a successful file download (or upload)
- Wait for an e-mail message to be received before initiating a file transfer
- E-mail a file as soon as it appears in a local folder (or in a server directory)
- Send an e-mail from anywhere to initiate a file transfer
- Send an e-mail from anywhere to instruct FileLink to execute a specific script file

Refer to [CREATEMAIL](#), [SENDMAIL](#), and [GETMAIL](#) for more details.

Using Shortcut Target Arguments in Script Files

FileLink permits arguments defined on a Shortcut Target line to be passed in at load time to be used as script file variables. Up to nine arguments are supported. There are nine internally defined variables, %1 through %9, dedicated for this purpose

On the Target line an argument is delimited by either & or %. The following are example arguments:

```
&argument&  
%different argument%
```

The first argument, reading left to right, is assigned to variable %1, the second argument is assigned to %2, and so on. For example, consider the following Shortcut Target command line.

```
"filelink.exe" &"1-512-555-1212"& &SENDFILE "newdata"&
```

With such a Target line, the following script file:

```
DIAL %1  
PERFORM %2  
DISCONNECT  
EXIT
```

would be executed as if it was originally written as:

```
DIAL "1-512-555-1212"  
SENDFILE "newdata"  
DISCONNECT  
EXIT
```

Authorizing Remote Users in TTY Mode

FileLink provides script commands to authorize remote users by way of a user name and an optional password. Up to two additional pieces of information about each user is also available within the FileLink script environment.

Authorization is provided using the [AUTHUSER](#), [AUTHPW](#), and [AUTHDATA](#) script commands. A specially prepared text file, called an **authorization** file, contains all information necessary to support these three script commands. For more information on how to create an **authorization** file, see [Authorization File Format](#).

Authorization would be used when FileLink is acting as a host for remote users in an asynchronous modem protocol file transfer environment. For example, a script file would be prompting the remote user (using the LINEOUT command) for a user name (receiving the response with the LINEIN command), and verifying the response with the AUTHUSER command.

The same sequence could be repeated for a password but the AUTHPW command would be used to match the password with a specific user name.

The AUTHDATA command could be used to obtain information specific to a user such as the user's home directory, a dial back telephone number, a greeting, or any other string data that might be necessary to support a remote user. The **authorization** file provides for two separate pieces of information.

This method is not a particularly secure way to perform user authorization since the user names and passwords are saved in clear text in a simple text file -- if a more secure method is required then running an external user-written program by way of the [EXEC](#) script command is recommended.

See the sample script in [Dial-In Connection With Authorization](#).

Authorization File Format

An **authorization** file contains a list of user names and passwords plus up to two optional data fields which are accessed via the [AUTHUSER](#), [AUTHPW](#), and [AUTHDATA](#) script commands.

Each line of the file contains up to four comma-delimited fields. The format of the file is:

```
<user name>,<password>,<data>,<data>
```

Blank lines, leading spaces, and tabs are ignored. Lines whose first non-space character is a pound-sign (#) are comments and are ignored. Commas may not appear within any of the fields -- they are delimiters only.

This file is simple text and may be created with the FileLink Script File Editor or any text editor. The default **authorization** file name used by FileLink is "authorization.txt" but you may specify a different file name in any of the authorization commands.

Consider the following example where three user name and passwords have been allowed for. In this example, the first data field is the home directory of the particular user and the second data field is a unique greeting to be sent to the user. Note that the data fields are optional and the last entry does not contain a greeting.

```
# daily dial in authorization file
# <user name>,<password>,<home directory>,<greeting>
robtjones,lillypad,\users\robtjones,Good morning Robert
stanman,museum,\users\stanford ind,Thanks for your business
cindyc,ad67d338,\users\cindyc
```

Note that the passwords are stored in clear text in the **authorization** file. Obviously, this is not a completely secure method of user authorization -- if a more secure method of authorization is desired then an external user-written program run with the [EXEC](#) script command is recommended.

Internal Script Variables

FileLink maintains a set of internally defined variables. These variables always begin with the % or \$ symbol. You may use these variables just as you would any user variable, but assigning values to these variables using the SETSET or SETNUM commands is not recommended.

%currentlocaldir	Contains the current local working folder (directory)
%datetime	Contains a formatted date/time string (i.e., Sat Feb 17 11.00.22 2006)
%dbrawqueryresult	Contains the raw SQL query result if the DBGETRESULTS command does not recognize or cannot parse the result into individual %db_xxxx variables.
%dbqueryrows	Contains the number of rows returned on the most recent DBQUERY command SQL database query

%dbqueryvariables	Contains the number of variables created during the most recent DBGETRESULTS command
%difffileid	Contains a numeric representation of the type of difference found on the most recent run of the GETDIFF command
%difffilename	Contains the name of a file where a difference has been found on the most recent run of the GETDIFF command
%difffilepath	Contains the full path a file where a difference has been found on the most recent run of the GETDIFF command
%difffiles	Contains the number of file differences found on the most recent run of the DIFF command
%difffiletext	Contains a text description of the type of difference found on the most recent run of the GETDIFF command
%diffnum	Contains the sequence number of the file difference returned on the most recent run of the GETDIFF command
%lasterror	Error or result code returned by the last script command run
%lasterrormsg	Descriptive text of error returned by the last script command run
%newport	New port number set by MODEMDETECT script command
%nextcmd	Character string obtained by CRON command
%nextfile	File name obtained by GETNEXTFILE command
%nextfiledate	File date obtained by GETNEXTFILE command
%nextfiledatetime	File date/time stamp obtained by GETNEXTFILE command
%nextfilesize	File size (in bytes) obtained by GETNEXTFILE command
%nextfiletime	File time stamp obtained by GETNEXTFILE command
%nextfolder	Optional folder name obtained by GETNEXTFILE command
%nextpath	Full path name obtained by GETNEXTFILE command
%port	Contains the current COM port
%rcvfilecount	Contains the number of files received by last RCVFILE command
%sendfilecount	Contains the number of files sent by last SENDFILE command
%snapshotfiles	Contains the number of files examined during the most recent run of the SNAPSHOT command
%time	Contains the current system time (i.e., 11.00.22)
%timedate	Same as %datetime
%unzipcount	Contains the number of files unzipped during the previous UNZIP command
%zipcount	Contains the number of files zipped during the previous ZIP command
<code>\$ERROR_??</code>	Predefined \$ERROR constants for use with the IFERROR command
<code>%1 - %9</code>	Variables assigned from Shortcut Target arguments

Consider the following examples in which the use of an internal variable is shown.

```
GETNEXTFILE  
MESSAGEBOX %nextfile "Last file name received was:"  
;; create a directory using current date and time  
MAKEDIR %datetime
```

Details on internal script variable usage follows in additional sections.

Using the %cr, %crlf, and %lf Variables

When concatenating strings using the [SET](#) script command, the %cr, %crlf, and %lf variables may be used to add carriage control to a string.

For example, the following command builds a two line string.

```
SET my_var "line #1" & %crlf & "line #2"
```

Using the %currentlocaldir Variable

The current local working folder (or directory) is maintained in an internal variable named **%currentlocaldir**.

```
WORKINGDIR "c:\temp"  
DISPLAY %currentlocaldir
```

Whenever changing the working folder in script functions or called scripts, it is recommended that the original folder always be restored prior to returning as shown in the following sample code.

```
FUNCTION things_todo  
;; save current working folder  
SET savdir = %currentlocaldir  
;; do whatever...  
WORKINGDIR "c:\temp"  
;; ...  
;; restore working folder  
WORKINGDIR savdir  
RETURN
```

Using the %date, and %datetime, and %time Variables

These variables contain the current system date and time.

The **%date** variable contains the current system date in the form mm-dd-yy (i.e., **02-16-01**). Note: the more common mm/dd/yy (i.e. **02/16/01**) form is not used so that the **%date** variable may be used to name files.

The **%datetime** variable contains a formatted date and time string (i.e., **Sat Feb 17 11.00.22 2001**). Alternate variable **%timedate** is also accepted.

The **%time** variable contains the current system time in the form hh.mm.ss (i.e., **11.00.22**). Note: the more common hh:mm:ss (i.e. **11:00:22**) form is not used so that the **%time** variable may be used to name files.

Consider the following example where a file name is created using the current time in order to make it unique. The resulting file name would be something like: **c:\data\file.11.00.22**.

```
SET $rcvfile = "C:\data\file."  
SET $rcvfile &= %time
```

Consider the following example where a unique directory is created beneath the FileLink current working folder.

```
MAKEDIR %datetime
```

Using the %dbqueryrawresult, %dbqueryrows and %dbqueryvariables Variables

These three variables are set by the [DBGETRESULTS](#), [DBQUERY](#), and [DBGETRESULTS](#) script commands respectively.

The **%dbqueryrawresult** variable contains a raw text string resulting from a SQL query submitted by the DBQUERY that cannot be parsed into individual **%db_XXXX** variables by a subsequent DBGETRESULTS command. Normally this variable is reserved for troubleshooting purposes and is only assigned in the event of an error during the execution of DBGETRESULTS.

The **%dbqueryrows** variable contains the number of rows resulting from a SQL query submitted by the DBQUERY script command. A maximum of 1000 rows is allowed on any single query before an error occurs.

The **%dbqueryvariables** variable contains the number of individual **%db_XXXX** variables created by the DBGETRESULTS command when parsing the result of a single SQL query submitted by the DBQUERY script command.

Using the %difffileid, %difffilename, and %difffiletext Variables

These variables are set as a result of running the [GETDIFF](#) script command when a difference is detected with a file in the local PC file system. GETDIFF is used after running the [DIFF](#) script command which compares baseline information obtained about the file system (using the [SNAPSHOT](#) script command) with the current state of files within the file system.

The **%difffilepath** variable contains the full path name of a changed file and **%difffilename** just the file name itself.

The **%difffileid** and **%difffiletext** variables contain information about how the file has changed in numeric and text formats, respectively.

Possible values for **%difffileid** are:

```
5001 = $DIFF_FILE_NOT_FOUND
5002 = $DIFF_FILE_IS_NEW
5003 = $DIFF_FILE_SIZE
5004 = $DIFF_FILE_DATETIME
```

Possible values for **%difffiletext** are:

```
** File not found
** File is new
** File size has changed
** File date/time stamp has changed
```

Consider the following example where only files that have a different size are of interest.

```
DIFF
:loop
GETDIFF
IFERROR $ERROR_READ_EOF GOTO done
IFNUM! %SETdifffileid $DIFF_FILE_SIZE GOTO loop
MESSAGEBOX %difffilename "Size of this file has changed."
GOTO loop
:done
```

Using the %difffiles and %diffnum Variables

These variables are set by the [DIFF](#) and [GETDIFF](#) script commands respectively.

The **%difffiles** variable contains the total number of different files that the DIFF command detects when scanning the local PC file system within the folder and subfolders (if any) specified.

The **%diffnum** variable contains the sequential number of a given file returned by the GETDIFF command.

Used together, it is possible to use these variables to know how many more differences there may be when performing a script loop calling GETDIFF to identify multiple differences.

More specifically, **%diffnum = %difffiles** after GETDIFF has been issued a sufficient number of times to find all the currently identified differences detected by the most recent DIFF command.

Using the %lasterror Variable

The **%lasterror** variable is initialized to the last error or completion code of the most recently completed script command.

The following console window excerpt demonstrates one use of this variable.

```
Line 5:  RCVFILE "does_not_exist"  
*550 Requested action not taken. File unavailable  
*File receive operation failed. [1059]  
Line 6:  DISPLAY %lasterror  
%lasterror = 1059
```

Using the %lasterrormsg Variable

The **%lasterrormsg** variable is initialized to a text description of the last error of the most recently completed script command. This message is the same as what appears in the script log.

The following console window excerpt demonstrates one use of this variable.

```
Line 5:  RCVFILE "does_not_exist"  
*550 Requested action not taken. File unavailable  
*File receive operation failed. [1059]  
Line 6:  DISPLAY %lasterrormsg  
%lasterrormsg = *File receive operation failed
```

Using the %lastfile and %lastpath Variables

Several of the file transfer protocols supported by FileLink, i.e., Zmodem, permit the remote system to supply the name of a file when it is received by FileLink. FileLink does provide a mechanism to override this remotely defined name, but when this is not used, you may use the **%lastfile** or **%lastpath** variables if you need to perform some operation on the file within a script file.

These internal script variables are replaced the file name or by the path and file name of the last file received by FileLink. If no file has been received, **%lastfile** and **%lastpath** are empty strings.

For example, consider the following script example where FileLink is waiting for the remote system to send a file named **June Inventory.rpt**.

```
PROTOCOL "zmodem"  
:again  
RCVFILE /timeout=0  
IFSTRCMP %lastfile "June Inventory.rpt" goto got_it  
MESSAGEBOX "Did not receive correct file"  
GOTO again  
:got_it  
MESSAGEBOX "Inventory report received"
```

Using the %nextcmd Variable

The **%nextcmd** variable is used in conjunction with the [CRON](#) script command. The operation to be performed, if any, whenever a scheduling condition is matched, is saved in this variable.

Consider the following example in which the CRON command is used to awaken FileLink once an hour to execute the **hourly.s** script file.

```
:loop
;; note the use of both single and double quotation marks
CRON "@hourly" 'CALL "hourly.s"'
PERFORM %nextcmd
GOTO loop
```

See the description of the [CRON](#) command for information on scheduling FileLink operations.

Using the %newport Variable

The **%newport** variable is initialized when the MODEMDETECT script command has executed successfully.

The variable is set to either the first COM (e.g., COM1 to COM48) where a modem was detected; the first COM port detected in the system if no modem is detected and the **/firstportok** option is specified; or a null string if no COM ports are detected or if no modem is detected and the **/firstportok** option is not specified.

The following example shows how to set FileLink to use a COM port where a modem has been detected.

```
MODEMDETECT  
IFERROR $ERROR_NO_MODEMS_DETECTED goto no_modems  
USEPORT %newport
```

Using the %nextfile, %nextpath, and %nextfolder Variables

Under some circumstances you may need to have FileLink send one or more files where you do not know the file name in advance. There may also be times when a local directory structure is unknown and you wish to obtain names of subfolders.

FileLink provides [GETNEXTFILE](#) command expressly to support the “hot send” feature. The **%nextfile** and **%nextpath** variables, and optionally the **%nextfolder** variable, are used in conjunction with this command. The first two internal script variables are replaced by the file name and the path and file name, respectively, of the file obtained by the GETNEXTFILE command. If no file has been found, **%nextfile** and **%nextpath** are empty strings.

In addition, there are options to this command to get the newest or oldest file present.

Consider the following example in which FileLink monitors a specific directory named **c:\File Uploads** for the presence any file with an extension of **.upload**. When a file is found, FileLink dials the remote system and sends it.

```
WORKINGDIR "c:\File Uploads"  
:loop  
GETNEXTFILE "*.upload" /timeout=0  
DIAL "555-1212"  
SENDFILE %nextfile  
DISCONNECT  
GOTO loop
```

Optionally, the GETNEXTFILE command can be instructed to return local subfolder names along with any other files that may be present. In this case, when a folder is found its name is returned in the **%nextfolder** variable and the **%nextfile** variable is set to an empty string. In this case, the **%nextpath** variable contains the complete path name of the folder.

Refer to [GETNEXTFILE](#) for more details.

Using the %nextfiledate, %nextfiledatetime, and %nextfiletime Variables

The **%nextfiledate**, **%nextfiledatetime**, **%nextfilesize**, and **%nextfiletime** variables are initialized to the file date, file time, and/or file size of the last local file or folder returned by the [GETNEXTFILE](#) script command. (**%nextfilesize** is not meaningful if a folder is returned.)

The format of the date and time are the same as the **%date**, **%datetime**, and **%time** variables, and the **%nextsitedate**, **%nextsitedatetime**, and **%nextsitetime** variables.

The **%nextfiledate** variable is in the form mm-dd-yy (i.e., **02-16-01**). Note: the more common mm/dd/yy (i.e. **02/16/01**) form is not used so that the **%nextfiledate** variable may be used to name files.

The **%nextfiledatetime** variable contains a formatted date and time string (i.e., **Sat Feb 17 11.00.22 2001**).

The **%nextfiletime** variable is in the form hh.mm.ss (i.e., **11.00.22**). Note: the more common hh:mm:ss (i.e. **11:00:22**) form is not used so that the **%nextfiletime** variable may be used to name files.

The **%nextfilesize** variable contains an integer string value corresponding to the size of the file in bytes (i.e., **66001**).

Refer to [GETNEXTFILE](#) for more details.

The [DATEADD](#) and [DATESUB](#) script commands may be used to manipulate **%nextfiledate** or other date variable.

See also: [Using the %date, %datetime, and %time Variables](#),

Using the %port Variable

The **%port** variable is initialized automatically to the COM port and may be used by a script to easily identify the port in use.

You might, for example, want to have the same script to be used by multiple instances of FileLink and still be able to create dialog messages, files names, etc. that include the active port number.

The following example creates a variable containing a file name that includes the port number. If FileLink is operating on COM3, the resulting variable would be **out3**.

```
SET output_file = "out" + %port
```


Using the %rcvfilecount and %sendfilecount Variables

The **%rcvfilecount** and **%sendfilecount** variables record the number of files transferred with the most recent [RCVFILE](#) and [SENDFILE](#) script commands, respectively.

Consider the following example in which the total number of files downloaded is used in a message that is displayed to the user.

```
RCVFILE "*"*.*"
SET msg = %rcvfilecount + " files received"
MESSAGEBOX msg
```

Consider the following example in which the total number of files uploaded is used in a message that is displayed to the user.

```
SENDFILE "*"*.*"
SET msg = %sendfilecount + " files sent"
MESSAGEBOX msg
```

Related Command(s): [RCVFILE](#), [SENDFILE](#)

Using the %snapshotfiles Variable

This variable is set by the [SNAPSHOT](#) script command.

The **%snapshotfiles** variable contains the total number of files examined by the most recent SNAPSHOT command when scanning the local PC file system within the folder and subfolders (if any) specified.

These variable(s) may be used with the **%difffiles** variable for record keeping or reporting purposes depending on the requirements of your application.

Using the %zipcount and %upzipcount Variables

The **%zipcount** and **%upzipcount** variables record the number of files zipped and unzipped with the most recent [ZIP](#) and [UNZIP](#) script commands, respectively.

Consider the following example in which the total number of files zipped is used in a message that is displayed to the user.

```
ZIP "zipfile" "*.xml"
SET msg = %zipcount + " XML files zipped"
MESSAGEBOX msg
```

Script File Command Overview

This section describes all the commands available for script processing. Commands are shown in uppercase for legibility but may be upper or lower case when used.

The general syntax of a FileLink script command is shown below.

```
opcode [arg1] ... [argn]
```

The opcodes for the various commands are listed below:

<code>;</code> or <code>*</code>	-Comment line
<code>:</code>	-Label marker
<code>@</code>	-Suppress command and results from logging
<code>!</code>	-Suppress command only from logging
ANSWER	-Wait for incoming telephone call
APPEND	-APPEND one file to another
ASK	-Display question in a Yes/No dialog box
AUTHDATA	-Obtain user data from authorization file
AUTHPW	-Verify remote user password
AUTHUSER	-Verify remote user name
BEGINFUNCTIONS	-Begin function declaration section in script file
BREAK	-Set a script file breakpoint
BROWSE	-Display a pop-up open file dialog box
CALL	-CALL another script file
CHAIN	-Transfer to another script file
CHGDIR	-An alias for the
CONNECT	-open direct connection
CONSOLE	-Control output to CONSOLE window
COPY	-COPY one file to another location
CREATEMAIL	-Create An e-mail message
CRON	-script scheduling using crontab file

<u>DATEADD</u>	-Add days to a date variable
<u>DATESUB</u>	-Subtract days from a date variable
<u>DBCLOSE</u>	-Close and optionally delete a database file
<u>DBGETRESULTS</u>	-Get results from a database query
<u>DBQUERY</u>	-Issue a database query
<u>DBREWIND</u>	-Reset results search to beginning
<u>DBUSE</u>	-Create and/or open a database file
<u>DEC</u>	-Decrement a variable by one
<u>DELDIR</u>	-Delete an empty local folder
<u>DELETE</u>	-DELETE a file
<u>DIAL</u>	-Initiate modem auto-dialer
<u>DIFF</u>	-Find differences between current state of the local PC file system and the last snapshot or DIFF operation
<u>DIFFREWIND</u>	-Reset file pointer for GETDIFF command
<u>DISCONNECT</u>	-DISCONNECT the line
<u>DISPLAY</u>	-DISPLAY all or a specified variable
<u>DOSCMD</u>	-Execute An internal MS-DOS command
<u>ENDFUNCTION</u>	-End function declaration
<u>ENDFUNCTIONS</u>	-End function declaration section in script file
<u>EXEC</u>	-Execute An external program
<u>EXIT</u>	-Quit FileLink
<u>EXPORT</u>	-EXPORT configuration settings to a file
<u>FLUSH</u>	-FLUSH characters from receive buffer
<u>FUNCTION</u>	-Begin FUNCTION declaration
<u>GETDIFF</u>	-Sequentially report the different files found during the last DIFF operation
<u>GETFILE</u>	-Get next file in local folder/subfolder tree
<u>GETMAIL</u>	-Get An e-mail message
<u>GETNEXTFILE</u>	-Get next file in a local directory
<u>GETREWIND</u>	-Reset GETFILE to first matching file
<u>GO</u>	-(Re)run the currently selected script file from beginning
<u>GOTO</u>	-Direct flow to Label
<u>IFDATE</u>	-Conditional branch upon file date comparison
<u>IFERROR</u>	-Conditional branch after testing result code
<u>IFFILE</u>	-Conditional branch on file existence
<u>IFNFILE</u>	-Conditional branch on file non-existence
<u>IFNO</u>	-Conditional branch if 'No' is clicked in ASK dialog box
<u>IFNSTRCMP</u>	-Conditional branch when two string variables are not equal
<u>IFNSUBSTR</u>	-Conditional branch if sub-string is not found in string variable
<u>IFNUM</u>	-Conditional branch upon numeric variable comparison
<u>IFSIZE</u>	-Conditional branch upon file size comparison
<u>IFSTRCMP</u>	-Conditional branch when two string variables are equal
<u>IFSUBSTR</u>	-Conditional branch if sub-string is found in string variable
<u>IFTIME</u>	-Conditional branch upon file time comparison
<u>IFYES</u>	-Conditional branch if 'Yes' is clicked in ASK dialog box
<u>IMPORT</u>	-IMPORT configuration settings from a file
<u>INC</u>	-Increment a variable by one
<u>LINEIN</u>	-Read one or more characters from COM port

LINEOUT	-Write one or more characters to COM port
LISTDIR	-Write local directory listing to a file
LOG	-Specify the script LOG file name
LOGMSG	-Write a message to the script LOG file
LOGNTEVENT	-Write a message to the NT application event LOG
LOOPCOUNT	-Set maximum loop repetition
LOOPIF	-Conditional branch used in conjunction with LOOPCOUNT
LOOPTO	-Unconditional branch used in conjunction with LOOPCOUNT
MAILTO	-Send An e-mail message (manually) via default e-mail client
MAKEFILENAME	-Create a unique, non-existent file name
MESSAGEBOX	-DISPLAY text in message box
MINIMIZE	-MINIMIZE FileLink window
MODEMCMD	-Send AT command to modem
MODEMDEFAULTS	-Restore modem factory default settings
MODEMDETECT	-Detect first available COM port and/or modem in system
MODEMRESET	-Send Reset command to modem
MODEMRESP	-Read response to command Sent to modem
MOVE	-MOVE one file to another location
NATO	-Specify a No activity time-outNATO
MAKEDIR	-Create a new local folder
PAUSE	-PAUSE for specified length of time or until specified hour:minute
PERFORM	-Execute script command contained in character string or variable
PGPCOMMAND	-Send a "raw" GnuPG command
PGPDECRYPT	-Decrypt a PGP encrypted file
PGPENCRYPT	-Encrypt a file using PGP
PGPIMPORT	-IMPORT a PGP key
PLAYSOUND	-Play a sound (.wav) file
PRESSANYKEY	-Suspend script execution pending a key press
PRINT	-PRINT a file
PROMPT	-DISPLAY message box and accept user input
PROTOCOL	-Specify default file Transfer PROTOCOL
RCVFILE	-Receive one or more files
READFILE	-Read string variable value from text file
RENAME	-RENAME a file
REMOTECMD	-PERFORM a script command received via a COM port
RESTORE	-RESTORE minimized FileLink window to original size
RESUME	-RESUME execution of a script file stopped AT a breakpoint
RETURN	-Force RETURN from a FUNCTION
SENDCMD	-Send script command to remote FileLink (same as LINEOUT)
SENDFILE	-Send one or more files
SENDMAIL	-Send An e-mail message
SET	-Assign or concatenate string variable(s)
SETEXTRACT	-Extract delimited substring from a string
SETLEFT	-Extract left substring
SETLEN	-Assign length of specified string to a variable

SETMID	-Extract mid substring
SETNUM	-Assign or evaluate numeric variable(s)
SETSUBSTR	-Find number of substrings in string
SETRIGHT	-Extract right substring
SPEAKER	-Control modem SPEAKER mode
SNAPSHOT	-Take SNAPSHOT of current state of local PC file system
SRVNAME	-Define service name and Control interaction with SrvMonitor
STOP	-Stops script processing
TERMINAL	-Activate the TTY TERMINAL applet
TRACELOG	-Specify trace/diagnostic LOG file name
TRACEWIN	-Open/Close trace logging window
UNZIP	-Extract file(s) from a zip archive
USEPORT	-Specify COM port and parameters
WORKINGDIR	-Specify default working folder
WRITEFILE	-Write character string or string variable value to text file
ZIP	-Create or Add to a ZIP archive

See also: [Script File Command Arguments](#), [Sample Script Files](#)

Script Commands Grouped by Function

Configuration Commands

- [PROTOCOL](#) - Specify default file transfer protocol
- [USEPORT](#) - Specify COM port and parameters

Debugging Commands

- [BREAK](#) - Set a script file breakpoint
- [GO](#) - (Re)run the currently selected script file from beginning
- [RESUME](#) - Resume execution of a script file stopped at a breakpoint

E-mail Commands

- [CREATEMAIL](#) - Create an e-mail message
- [GETMAIL](#) - Get an e-mail message
- [SENDMAIL](#) - Send an e-mail message

FileLink Host Mode Authorization Commands

- [AUTHDATA](#) - Obtain user data from **authorization** file
- [AUTHPW](#) - Verify remote user password
- [AUTHUSER](#) - Verify remote user name

Function Directives and Commands

- [BEGINFUNCTIONS](#) - Begin function declaration section in script file
- [ENDFUNCTION](#) - End function declaration
- [ENDFUNCTIONS](#) - End function declaration section in script file
- [FUNCTION](#) - Begin function declaration
- [RETURN](#) - Force return from a function

Link Connection/Establishment and Disconnect Commands

- [ANSWER](#) - Wait for incoming telephone call
- [CONNECT](#) - Open direct connection
- [DIAL](#) - Initiate modem auto-dialer
- [DISCONNECT](#) - Disconnect the line

Local File Commands

- [APPEND](#) - Append one file to another
- [APPEND](#) - Append one file to another
- [ARCHIVEDIR](#) - Define FileLink's archive folder
- [CHGDIR](#) - An alias for the [WORKINGDIR](#) command
- [COPY](#) - Copy one file to another location
- [DELDIR](#) - Delete an empty local folder
- [DELETE](#) - Delete a file
- [EXPORT](#) - Export configuration settings to a file

<u>GETFILE</u>	- Get next file in local folder/subfolder tree
<u>GETNEXTFILE</u>	- Get next file in a local folder
<u>IMPORT</u>	- Import configuration settings from a file
<u>LISTDIR</u>	- Write local directory listing to a file
<u>MAKEDIR</u>	- Create a new local folder
<u>MAKEFILENAME</u>	- Create a unique, non-existent file name
<u>MAKEDIR</u>	- Create a new local folder
<u>MOVE</u>	- Move one file to another location
<u>PRINT</u>	- Print a file
<u>READFILE</u>	- Read string variable value from text file
<u>RENAME</u>	- Rename a file
<u>UNZIP</u>	- Extract file(s) from a zip archive
<u>WRITEFILE</u>	- Write character string or string variable value to text file
<u>ZIP</u>	- Create or add to a zip archive

Log File / Console Control Commands

<u>CONSOLE</u>	- Control output to console window
<u>LOG</u>	- Specify the script log file name
<u>LOGMSG</u>	- Write a message to the script log file
<u>LOGNTEVENT</u>	- Write a message to the NT application event log
<u>SRVNAME</u>	- Define service name and control interaction with SrvMonitor
<u>TRACELOG</u>	- Specify trace/diagnostic log file name

Modem Control Commands

<u>MODEMCMD</u>	- Send AT command to modem
<u>MODEMDEFAULTS</u>	- Restore modem factory default settings
<u>MODEMDETECT</u>	- Detect first available COM port and/or modem in system
<u>MODEMRESET</u>	- Send reset command to modem
<u>MODEMRESP</u>	- Read response to command sent to modem
<u>SPEAKER</u>	- Control modem speaker mode

Script File Branching Commands

<u>GOTO</u>	- Direct flow to label
<u>IFDATE</u>	- Conditional branch upon file date comparison
<u>IFERROR</u>	- Conditional branch after testing result code
<u>IFFILE</u>	- Conditional branch on file existence
<u>IFNFILE</u>	- Conditional branch on file non-existence
<u>IFNO</u>	- Conditional branch if 'No' is clicked in ASK dialog box
<u>IFNSTRCMP</u>	- Conditional branch when two string variables are not equal
<u>IFNSUBSTR</u>	- Conditional branch if sub-string is not found in string variable
<u>IFNUM</u>	- Conditional branch upon numeric variable comparison
<u>IFSIZE</u>	- Conditional branch upon file size comparison
<u>IFSTRCMP</u>	- Conditional branch when two string variables are equal

IFSUBSTR	- Conditional branch if sub-string is found in string variable
IFTIME	- Conditional branch upon file time comparison
IFYES	- Conditional branch if 'Yes' is clicked in ASK dialog box
LOOPCOUNT	- Set maximum loop repetition
LOOPIF	- Conditional branch used in conjunction with LOOPCOUNT
LOOPTO	- Unconditional branch used in conjunction with LOOPCOUNT

Script File Control Commands

CALL	- Call another script file
CHAIN	- Transfer to another script file
CRON	- Script scheduling using crontab file
DISPLAY	- Display all or a specified variable
DOSCMD	- Execute an internal MS-DOS command
EXEC	- Execute an external program
EXIT	- Quit FileLink
NATO	- Specify a no activity time-out
PAUSE	- Pause for specified length of time or until specified hour: minute
PERFORM	- Execute script command contained in character string or variable
RETURN	- Return from a called script file
STOP	- Stops script processing

Serial COM Port I/O Commands

FLUSH	- Flush characters from receive buffer
LINEIN	- Read one or more characters from COM port
LINEOUT	- Write one or more characters to COM port
RCVFILE	- Receive one or more files
REMOTECMD	- Perform a script command received via a COM port
SENDCMD	- Send script command to remote FileLink (same as LINEOUT)
SENDFILE	- Send one or more files

SQL Database Commands

DBCLOSE	- Close and optionally delete a database file
DBGETRESULTS	- Get results from a database query
DBQUERY	- Issue a database query
DBREWIND	- Reset results search to beginning
DBUSE	- Create and/or open a database file

User Interface Commands

ASK	- Display question in a Yes/No dialog box
BROWSE	- Display a pop-up open file dialog box
MESSAGEBOX	- Display text in message box
PLAYSOUND	- Play a sound (.wav) file

- [PROMPT](#) - Display message box with title and prompt, and accept user input
- [TERMINAL](#) - Activate the TTY Terminal applet

Variable Commands

- [DATEADD](#) - Add days to a date variable
- [DATESUB](#) - Subtract days from a date variable
- [DEC](#) - Decrement a variable by one
- [INC](#) - Increment a variable by one
- [SET](#) - Assign or concatenate string variable(s)
- [SETEXTRACT](#) - Extract delimited substring from a string
- [SETLEN](#) - Assign length of specified string to a variable
- [SETLEFT](#) - Extract left substring
- [SETMID](#) - Extract mid substring
- [SETNUM](#) - Assign or evaluate numeric variable(s)
- [SETRIGHT](#) - Extract right substring
- [SETSUBSTR](#) - Find number of substrings in string

Window Control Commands

- [MINIMIZE](#) - Minimize FileLink window
- [RESTORE](#) - Restore minimized FileLink window to original size
- [TRACEWIN](#) - Open/close trace logging window

ANSWER -- Wait for incoming telephone call

Syntax: ANSWER [/options]

Arguments: none

Options: /timeout=*nn* Answer time-out in seconds; if 0, wait indefinitely

This script command places the modem into auto-answer mode and waits for an incoming call.

Related Command(s): [DIAL](#), [CONNECT](#), [MODEMRESET](#)

APPEND -- Append one local file to another

Syntax: APPEND

Arguments: [src name] [Variable](#) or [string](#) defining a file or path name; if no path is defined FileLink's working folder is used
[dest name] Variable or string defining a file or path name; if no path is defined FileLink's working folder is used

Options: none

This script command to append the source file to the destination file.

Related Command(s): [COPY](#), [DELETE](#), [RENAME](#), [WORKINGDIR](#)

ARCHIVEDIR -- Define FileLink's archive folder

Syntax: ARCHIVEDIR [path name]
Arguments: [path name] A [variable](#) or [string](#) to specify the path name of FileLink's archive folder (directory)
Options: None

This script command defines the folder to be used when the **/archive** option is used on the SENDFILE command.

Consider the following example.

```
;; archive "c:\Program Files\FileLink\example.txt"  
ARCHIVEDIR "c:\Program Files\FileLink\archive"  
SENDFILE "example.txt" /archive
```

Related Command(s): [MAKEDIR](#), [SENDFILE](#), [WORKINGDIR](#)

ASK -- Display dialog box with yes/no question

Syntax:	ASK	[message] [title]
Arguments:	[message]	Variable or string defining a text message to display within a pop-up dialog box (up to 1000 characters).
	[title]	Variable or string defining the window title displayed in the dialog box.
Options:	/large	Select this option to display the dialog box in a larger 12 point font rather than the default 8 point.
	/local	By default, FileLink displays a message box in the center of the screen. Use this option if you wish the box to be centered relative to the FileLink window instead.
	/nocrlf	Ignore embedded <code>\n</code> and/or <code>\r</code> carriage control.

This command not allowed when running as an NT Service or in a locked minimized window.

This script command displays a dialog box with **Yes** and **No** buttons. The window title and text within the dialog are specified in the command. Control returns to next script command when you close the dialog by clicking on one of the buttons. This command is useful to ask yes or no questions of an operator during the course of a file transfer session.

If FileLink is running a script in a unlocked minimized window then FileLink's window will be restored when this command is performed.

Consider the following example of a script file that prompts if a file should be sent to the remote system.

```
ASK "Send file now?" "Question"
  IFNO goto do_not_send
  SENDFILE file_name
:do notsend
```

The dialog looks like the following.



Two embedded formatting or carriage control character sequences are recognized. A `\n` sequence is interpreted as a carriage return and a `\r` sequence is interpreted as a line feed. Use of this carriage control sequences permit you to display multiple lines inside a dialog box. For example:

```
ASK " Line 1 \n\r Line 2 \n\r Question?"
```

Use of the option **/nocrlf** suppresses the recognition of the **\n** and **\r** sequences. This is useful if you are displaying file names in the message box that may include either of these two sequences. For example:

```
ASK "Send c:\newfile\reports.dat now?" /nocrlf
```

The [message] can be quite large -- up to 1000 characters. When using extremely long messages, we suggest that you precede the command with an [@ modifier](#) to suppress the echoing of the command to the console window and log file to preserve readability. Use care also not to overflow the possible space in the Windows dialog box that this script command will display by including too many embedded carriage control sequences.

The **/large** and **/local** options may not be used together.

Related Commands: [MESSAGEBOX](#), [PROMPT](#), [IFYES](#), [IFNO](#)

See also: [Running FileLink With Prompting](#)

AUTHDATA -- Obtain user data from authorization file

Syntax:	AUTHDATA	[user] [data] [field] [auth file]
Arguments:	[user]	Variable or string defining a user name; this is typically received from a remote user by way of the LINEIN command
	[data]	Variable where the user data is to be stored
	[field]	Variable or string specifying which of the two available user data fields is to be obtained; this value must be either "1" or "2"
	[auth file]	Optional variable or string defining the name of the authorization file; if this is omitted, the default file "authorization.txt" is used
Options:	None	

This script command would be used when FileLink is acting as a host to remote systems in an async modem protocol file transfer environment.

This command may be used to obtain pre-defined information about a particular remote user. The process involves prompting for a user name, searching for a match within a previously created **authorization** file, and if found, returning one of two different string values associated with that user. See [Authorization File Format](#) for more information on how these values are stored.

Consider the following example where the remote user is prompted for a user name and password, and if the user is verified, the working folder for that user is changed to a directory name saved in the first data field for that user in the default **authorization** file.

```

LINEOUT "Enter your user name:"
LINEIN username /timeout=60
AUTHUSER username
IFERROR= $ERROR_AUTHORIZATION_FAILED goto UnrecognizedUser
LINEOUT "Enter your password:"
LINEIN password /timeout=60
AUTHPW username password
IFERROR= $ERROR_AUTHORIZATION_FAILED goto InvalidPassword
AUTHDATA username new_dir "1"
IFERROR= $ERROR_AUTHORIZATION_FAILED goto InvalidUser
WORKINGDIR new_dir

```

A matching entry in the **authorization** file might look like the following.

```
username,password,\user\username
```

Related Command(s): [AUTHPW](#), [AUTHUSER](#)

See also: [Authorizing Remote Users](#), [Dial-In Connection With Authorization](#)

AUTHPW -- Verify remote user password

Syntax:	AUTHPW	[user] [password] [auth file]
Arguments:	[user]	Variable or string defining a user name; this is typically received from a remote user by way of the LINEIN command
	[password]	Variable or string defining a password; this is typically received from a remote user by way of the LINEIN command
	[auth file]	Optional variable or string defining the name of the authorization file; if this is omitted, the default file "authorization.txt" is used
Options:	None	

This script command would be used when FileLink is acting as a host to remote systems in an async modem protocol file transfer environment.

This command to may be used after a remote user had been identified to verify that they have a valid password. The verification process involves prompting for a user name, searching for a match within a previously created **authorization** file, and if found, repeating the prompting process to obtain the user's password. A look up in the **authorization** file is then performed to match the password with the user name. See [Authorization File Format](#) for more information.

Consider the following example where the remote user is prompted for a user name and password, and the user is verified using the default **authorization** file.

```
LINEOUT "Enter your user name:"  
LINEIN username /timeout=60  
AUTHUSER username  
IFERROR= $ERROR_AUTHORIZATION_FAILED goto UnrecognizedUser  
LINEOUT "Enter your password:"  
LINEIN password /timeout=60  
AUTHPW username password  
IFERROR= $ERROR_AUTHORIZATION_FAILED goto InvalidPassword
```

Related Command(s): [AUTHDATA](#), [AUTHUSER](#)

See also: [Authorizing Remote Users](#), [Dial-In Connection With Authorization](#)

AUTHUSER -- Verify remote user name

Syntax: AUTHUSER [user] [auth file]

Arguments: [user] [Variable](#) or [string](#) defining a user name; this is typically received from a remote user by way of the [LINEIN](#) command

[auth file] Optional variable or string defining the name of the authorization file; if this is omitted, the default file "authorization.txt" is used

Options: none

This script command would be used when FileLink is acting as a host to remote systems in an async modem protocol file transfer environment.

This command may be used to verify that a remote user is authorized to communicate with FileLink. The verification process involves prompting for a user name and then searching for a match within a previously created **authorization** file. See [Authorization File Format](#) for more information on this file.

Consider the following example where the remote user is prompted for their user name and it is verified using the default **authorization** file.

```
LINEOUT "Enter your user name:"  
LINEIN username /timeout=60  
AUTHUSER username  
IFERROR= $ERROR_AUTHORIZATION_FAILED goto UnrecognizedUser
```

Related Command(s): [AUTHDATA](#), [AUTHPW](#)

See also: [Authorizing Remote Users](#), [Dial-In Connection With Authorization](#)

BEGINFUNCTIONS -- Begin function declaration section

Syntax: BEGINFUNCTIONS
Arguments: None
Options: None

This script directive is used to mark the beginning of a function declaration section of a script file. A function declaration section must be at the beginning of the script file (before any functions are called) and must not be used in any script file invoked with a [CALL](#) script command.

Consider the following examples.

```
BEGINFUNCTIONS
FUNCTION MyFunction
;; body of MyFunction
ENDFUNCTION
ENDFUNCTIONS
```

```
BEGINFUNCTIONS
FUNCTION MyFunction1
;; body of MyFunction1
ENDFUNCTION
FUNCTION MyFunction2
;; body of MyFunction2
ENDFUNCTION
ENDFUNCTIONS
```

Related Command(s): [ENDFUNCTION](#), [ENDFUNCTIONS](#), [FUNCTION](#), [RETURN](#)

BREAK -- Set a breakpoint location

Syntax: BREAK
Arguments: None
Options: None

This command is used to set “breakpoints” in script files for debugging purposes.

When debugging, place the BREAK command at strategic places in the flow of execution of a script file to be debugged and script execution will be suspended when BREAK is encountered. When in debug mode, make sure that the Enable Breakpoints the **Scripts** menu is enabled.

After encountering BREAK, the FileLink script processor suspends the script and FileLink enters the *break state* and control is returned to the console window. Here you can view variable values, for example, by typing [DISPLAY](#) commands into the console command line, reassign variable values by typing [SET](#) or [SETNUM](#), and then resume script execution from the point of interruption.

Script execution may be resumed by typing the RESUME command into the console command line or by clicking the Skip To Next Command (**Ctrl + N**) toolbar button.

When your script has been thoroughly debugged, either edit your script and remove all the BREAK commands or clear the Enable Breakpoints control in the **Scripts** menu and BREAK commands will be ignored.

Related Command(s): [GO](#), [RESUME](#), [STOP](#)

See also: [Debugging Script Files](#)

BROWSE -- Display a pop-up open file dialog box

Syntax:	BROWSE	[variable] [title] [initdir] [filter] [options]
Arguments:	[variable]	A variable to store the file name or file name with full path selected in the open file dialog box; if the variable does not previously exist, it is created.
	[title]	Optional variable or string defining a open file dialog box window title. If not specified, neither [initdir] nor [filter] may be specified.
	[initdir]	Optional variable or string defining the initial folder for the open file dialog box. If not specified, [filter] may not be specified.
	[filter]	Optional variable or string defining a filter to be used when displaying files in the open file dialog box.
Options:	/nopath	When this option is used only the file name is saved in [variable]. By default the full path and file name is returned.

[This command not allowed when running as an NT Service or in a locked minimized window.](#)

This script command displays a open file dialog box on your display that permits browsing for a user-selected file. The window title, starting folder, and a file name filter may be specified in the command. Control returns to next script command when you close the dialog by clicking on the OK or Cancel buttons. This command is useful to allow a user to make a select of a file to be processed by a running script.

If FileLink is running a script in a unlocked minimized window then FileLink window will be restored when this command is performed.

The script file can detect if the Cancel button has been clicked by testing for result code 1013 or the [\\$ERROR variable](#) \$ERROR_PROMPT_CANCELLED.

Related Commands: [MESSAGEBOX](#), [ASK](#), [PROMPT](#)

CALL -- Call another script file

Syntax:	CALL	[file name] [/options]
Arguments:	[file name]	Variable or string defining a file or path name; if no path is defined
Options:	/silent	Do not echo script commands or results to log file or FileLink window.
	&arg&	Up to nine arguments may be passed to the called script in the same manner that they may be passed from the command line into FileLink when it is launched; the first argument is saved in script variable %1, the second in %2, etc. up to %9.

Use of this script command temporarily transfers control to another script file. When the called script file exits, control returns to the original script file at the statement immediately following the CALL statement.

The following is an example of calling a script file and of the script file being called.

```
CALL "called_script.s"
```

The called script file might contain the following.

```
MESSAGEBOX "now running called_script.s"
RETURN
```

Arguments may be passed to a called script using the **&..&** syntax. Any strings found between two **&** delimiters are saved in sequence in internal variables named **%1** through **%9** which are then accessible within the called script. The following example passes two arguments.

```
CALL "called_script.s" &arg1& &arg2&
```

When the called script is running, it will find **%1** = "arg1" and **%2** = "arg2".

Note: any changes made to the FileLink environment in a called script will persist after returning to the calling script. For example, if the working folder is changed the change will remain in effect upon the return. View called scripts as simply extensions to the original script, not as separate environments.

Related Command(s): [CHAIN](#), [EXEC](#), [RETURN](#), [WORKINGDIR](#)

CHAIN -- Transfer to another script file

Syntax:	CHAIN	[file name] [/options]
Arguments:	[file name]	Variable or string defining a file or path name; if no path is defined
Options:	&arg&	Up to nine arguments may be passed to the chained script in the same manner that they may be passed from the command line into FileLink's when it is launched; the first argument is saved in script variable %1, the second in %2, etc. up to %9.

Use of this script command transfers control to another script file. Control does not return to the original script file.

Arguments may be passed to a called script using the **&..&** syntax. Any strings found between two **&** delimiters are saved in sequence in internal variables named **%1** through **%9** which are then accessible within the chained to script. The following example passes two arguments.

```
CHAIN "called_script.s" &arg1& &arg2&
```

When the new script is running, it will find **%1** = "arg1" and **%2** = "arg2".

Consider the following example where a variable has been previously saved to determine which of two script files are to be executed.

```
IFSTRCMP which_one "its2" goto chain_to_2  
CHAIN "script1.s"  
:chain to_2  
CHAIN "script2.s"
```

Related Command(s): [CALL](#), [EXEC](#) , [WORKINGDIR](#)

CHGDIR -- Change local default folder

Syntax: CHGDIR [folder name]
Arguments: [folder name] [Variable](#) or string defining the new default folder name.
Options: None

This is an alias for the [WORKINGDIR](#) script command.

Related Command(s): [DELDIR](#), [MAKEDIR](#), [WORKINGDIR](#)

CONNECT -- Open direct connection

Syntax: CONNECT [/options]
Arguments: None
Options: /timeout=*nn* Connection time-out in seconds; if 0, wait indefinitely

This script command is used to establish a non-modem connection on a COM port. On a COM port, this command raises the DTR modem signal. If constant carrier is configured, this command also raises the RTS modem signal. A connection is established when FileLink detects the DSR modem signal (and the CTS signal if constant carrier is configured.)

Consider the following example.

```
;; direct connect on a COM port  
CONNECT /timeout=60
```

Related Command(s): [DIAL](#), [ANSWER](#)

CONSOLE -- Control output to console window

Syntax: CONSOLE [/options]
Arguments: None
Options: /off Turn off all output to the console window
 /on Turn on output to console window

When running scripts, it may be desirable to suppress the echoing of commands to the console window for all or a part of script file execution. Use this command to turn off and on output to the console window as desired. Even with output to the console window suppressed, output to the log file is not affected.

COPY -- Copy one local file to another location

Syntax:	COPY	[src file] [dest file]
Arguments:	[src name]	Variable or string defining a file or path name; if no path is defined FileLink's working folder is used
	[dest name]	Variable or string defining a file or path name; if no path is defined FileLink's working folder is used
Options:	None	

This script command to copies the source file to the destination location.

Full file or path names are required. For example, the following is a valid command.

```
COPY "c:\test\file" "c:\test2\file"
```

The following is an invalid command in the same environment.

```
COPY "c:\test\file" "c:\test2"
```

Related Command(s): [APPEND](#), [MOVE](#), [DELETE](#), [RENAME](#), [WORKINGDIR](#)

CREATEMAIL -- Create an e-mail message

Syntax:	CREATEMAIL	[from name] [from email] [subj] [body] [attach] [options]
Arguments:	[from name]	Variable or string defining the optional name of the sender
	[from email]	Variable or string defining the e-mail address of the sender.
	[subj]	Variable or string defining an optional subject line for the e-mail message
	[body]	Variable or string defining the body of the message.
	[attach]	Variable or string defining the file name of an optional attachment for the e-mail message
Options:	/nocrlf	Ignore embedded \n and/or \r carriage control

This command builds an e-mail message to be sent using the [SENDMAIL](#) command. The recipient of a message is specified in the SENDMAIL command.

All of the arguments to this command are required; however the [from name], [subj], and [attach] arguments may be empty strings.

The [body] variable provides for a simple message body. No formatting is permitted (i.e., no carriage control). Variables, hence the message body, cannot be longer than 2040 characters. Use an attachment file to send larger messages.

Carriage control within the message body (i.e., a **\n** to insert a line feed in the message and a **\r** to insert a carriage return) is permitted unless the **/nocrlf** option is specified. Use of the option **/nocrlf** suppresses the recognition of the **\n** and **\r** sequences. This is useful if you are e-mailing file names in the message body that may include either of these two sequences.

Consider the following example where an e-mail message is created with an attachment.

```
SET from = "FileLink Sales"
SET email = "sales@FileLink.com"
SET subj = "Thanks for your order!"
SET body = "An invoice is attached"
SET attach = "c:\sales\customer.inv"
CREATEMAIL from email subj body attach
```

The following example results in a message without an attachment.

```
SET from = "FileLink Sales"
SET email = "sales@FileLink.com"
SET subj = "Thanks for your order!"
SET body = "We appreciate your business."
CREATEMAIL from email subj body ""
```

Related Command(s): [SENDMAIL](#), [GETMAIL](#)

CRON -- Schedule script operations

Syntax	CRON	[file name]
Alt Syntax:	CRON	[@ cond] [cmd]
Arguments:	[file name]	Variable or string defining a file or path name; if no path is defined FileLink's working folder is used; this cron event file contains one or more scheduling conditions; if omitted, the default file is "crontab.txt" in FileLink's working folder; see Cron Event File Format.
	[@ cond]	Variable or string specifying one of FileLink's pre-defined scheduling conditions; these conditions must begin with the '@' symbol (e.g., @hourly).
	[cmd]	Optional variable or string to be stored in the %nextcmd variable; this permits a particular command to be performed when a pre-defined scheduling condition is matched; see Using the %nextcmd Variable .
Options:	none	

This script command provides FileLink with extensive scheduling capabilities which surpasses those of the PAUSE command.

Two forms of scheduling are supported: (1) predefined conditions for a single event (e.g., twice daily); and (2) complex user defined conditions (e.g., 4PM Monday through Friday and midnight Sunday).

In either case, a script command (or other character string) may be specified to be saved in the **%nextcmd** variable when there is a match. Often this will be a command to execute a particular script file at the time of the event. (The command would be executed in a subsequent PERFORM command.) This command string is optional.

Single event scheduling is achieved using one of the following designators:

@yearly	midnight Jan 1
@monthly	midnight, first day of each month
@weekly	midnight, each Sunday
@daily	midnight, each day
@hourly	daily, every hour on the hour
@every2hours	daily, every two hours beginning at midnight
@every4hours	daily, every four hours beginning at midnight
@every6hours	daily, every six hours beginning at midnight
@every8hours	daily, every eight hours beginning midnight
@twicedaily	daily, at midnight and noon
@onhalfhour	daily, every hour on the half hour
@everyhalfhour	daily, every half hour
@every15min	daily, every 15 minutes
@every5min	daily, every 5 minutes

Consider the following example where FileLink sends a file every 30 minutes.

```
:loop
;; note the use of both single and double quotation marks
CRON "@onhalfhour" 'SENDFILE "30minutes.dat"'
PERFORM %nextcmd
GOTO loop
```

More complex scheduling conditions are achieved using the CRON command slightly differently and uses a file containing one or more conditions. This file is referred as a **crontab** file. You may use a file of any name, but if you do not specify a file name in the CRON command, FileLink uses a default name of "crontab.txt".

The format of this file is complex (see [Crontab File Format](#)) and not something the casual user needs to be concerned with. The [CronMaker](#) utility is provided with FileLink for the direct creation and modification of "crontab.txt" files.

Technically savvy users may want to consider the following example where FileLink executes a unique script file on each weeknight at 11PM. The **crontab** file located in the FileLink working folder, named "crontab.txt", would contain:

```
# run a script at 11PM every weekday

0 23 * * mon      CALL "Monday.s"
0 23 * * tue      CALL "Tuesday.s"
0 23 * * wed      CALL "Wednesday.s"
0 23 * * thu      CALL "Thursday.s"
0 23 * * fri      CALL "Friday.s"
```

The FileLink script file would look like:

```
:loop
CRON
PERFORM %nextcmd
GOTO loop
```

When the CRON command executes it will display a confirmation message to the console as to when the next CRON event will trigger. Using the previous example, if today is Monday at 5PM, the CRON command would confirm the next event to occur at 11PM later that day with the following message:

```
*CRON will trigger at 11:00PM today and run this command: "CALL "Monday.s" "
```

When no command string is specified, the CRON command is similar to the PAUSE command in that it simply waits until there is a match in schedule time(s) before script execution is allowed to resume.

Important

If there are multiple trigger events in the "crontab.txt" file, be aware that CRON does not queue events that may occur at the same time. For example, if you

have two events that are scheduled to trigger at 3:30PM, CRON will act only upon the first and the second will be ignored. If you need to have multiple scripts run the same time off of a single event trigger, you should make a single corresponding script “smart” enough to use the CALL script command to run the other(s) sequentially.

Related Command(s): [PAUSE](#), [WORKINGDIR](#)

See also: [Crontab File Format](#), [Using the %nextcmd Variable](#)

DATEADD-- Add specified number of days to date variable

Syntax: DATEADD [variable] [const]
Arguments: [variable] A [variable](#) containing a date in the format of **mm-dd-yy**.
 [const] A variable or a numeric constant representing the number
 of days to add to the date contained in [variable].
Options: none

This script command is used to add a specified number of days to date variable.

Consider the example below where 14 days is added to a variable containing current date.

```
SET today = %date  
DATEADD today 14
```

The number of days to add may also be expressed in a variable as shown below.

```
SET today = %date  
SETNUM SETdays = 14  
DATEADD today days
```

The **%date** internal variable may not be used in this command directly.

Related Command(s): [DATESUB](#)

See also: [Using the %date, %datetime, and %time Variables](#)

DATESUB -- Subtract specified number of days from date variable

Syntax: DATESUB [variable] [const]
Arguments: [variable] A [variable](#) containing a date in the format of **mm-dd-yy**.
 [const] A variable or a numeric constant representing the number
 of days to subtract from the date contained in [variable].
Options: none

This script command is used to subtract a specified number of days to date variable.

Consider the example below where 14 days is subtract to a variable containing current date.

```
SET today = %date  
DATESUB today 14
```

The number of days to subtract may also be expressed in a variable as shown below.

```
SET today = %date  
SETNUM SETdays = 14  
DATESUB today days
```

The **%date** internal variable may not be used in this command directly.

Related Command(s): [DATEADD](#)

See also: [Using the %date, %datetime, and %time Variables](#)

DBCLOSE -- Close and optionally delete SQL database file

Syntax: DBCLOSE [/options]
Arguments: None
Options: /delete Delete the file after closing it.

This script command is used to close a SQL database file when it is no longer needed.

Add the **/delete** option if the file is considered temporary and it should be deleted after being closed.

When a database file is closed, FileLink deletes any **%db_XXXX** variable(s) that may have been created by prior execution of the DBGETRESULTS script command.

Related Command(s): [DBGETRESULTS](#), [DBUSE](#), [DBQUERY](#)

DBGETRESULTS -- Get results from a SQL database query

Syntax: DBGETRESULTS
Arguments: None
Options: None

This script command is used to obtain the results of a previous query into a SQL database file submitted by the DBQUERY script command.

DBGETRESULTS returns query results sequentially one row at a time. If multiple rows are expected from a given query, DBGETRESULTS would be called multiple times to fetch the data from all rows. When using the built-in SQLite database engine a maximum of 1000 rows are available. This row limit does not apply to ODBC connections made using a DSN.

Script variable(s) of the form **%db_***** corresponding to each column in the database table are created by this command and are assigned the value found in the particular database row being returned. The **%dbqueryvariables** script variable is assigned to the total number of variables created.

For example, if a table in a given SQL database is created with two columns, one named "fld1" and the other "fld2" then two **%db_***** variables will be created by the DBGETRESULTS command as shown .

```
DBGETRESULTS
DISPLAY %db_fld1
DISPLAY %db_fld2
```

The following example shows how multiple rows may be retrieved when a suitable query has been issued to the database.

```
:loop
DBGETRESULTS
IFERROR $ERROR_DB_ALL_RESULTS_RTND GOTO done
DISPLAY %db_fld1
DISPLAY %db_fld2
DISPLAY %dbqueryvariables
GOTO loop
:done
```

In some cases DBGETRESULTS may not recognize the results of a query and/or is unable to create corresponding **%db_***** variables. In such a case, DBGETRESULTS returns \$ERROR_DB_RAW_QUERY_RESULTS and the raw results string is assigned to the **%dbqueryrawresult** script variable. If you are expecting a differently formatted query result then you can parse this string programmatically or, otherwise, use this variable for troubleshooting purposes.

Related Command(s): [DBCLOSE](#), [DBUSE](#), [DBQUERY](#), [DBREWIND](#)

DBQUERY -- Issue a SQL query

Syntax: DBQUERY [query]
Arguments: [query] [Variable](#) or string defining a SQL query to issue.
Options: None

This script command is used to execute SQL statements on the [currently open database](#) the results of which may be processed using the [DBGETRESULTS](#) script command.

The built-in SQLite database engine will fail and return an [\\$ERROR_DB_QUERY_FAILED](#) error on queries that return more than 1000 rows. This row limit does not apply to ODBC connections made using a DSN. Any query returning columns with more than 4096 characters will fail with an [\\$ERROR_DB_UNSUPPORTED_RESULT](#) error.

Use of the FileLink SQL database commands assumes that you have a working knowledge of SQL databases and queries. It is beyond the scope of FileLink documentation or technical support to offer support or education related to SQL so the syntax and format of any command or query submitted via the DBQUERY script command is left to the script programmer.

The first query submitted to a newly created database (see DBUSE) should create any necessary table(s) that you wish to save data into. An example query to create a table named MyTable is shown below.

```
DBQUERY "create table MyTable (fld1 text primary key, fld2 text);"
```

Data may then be saved in the database as shown in the following example which adds two rows of data to the database.

```
DBQUERY "insert into MyTable values ( 'row1', 'data1' );"  
DBQUERY "insert into MyTable values ( 'row2', 'data2' );"
```

To locate data in the database, a query like the following might be used.

```
DBQUERY "select * from MyTable where fld1='row1';"  
IFERROR $ERROR_DB_QUERY_FAILED GOTO done
```

A whole depth and breadth of SQL commands may be submitted in this manner. Consult SQL documentation for possible queries and syntax. A good resource is <http://www.sqlite.org>. SQLite is the SQL database engine employed by FileLink.

Related Command(s): [DBCLOSE](#), [DBGETRESULTS](#), [DBUSE](#)

DBREWIND -- Reset query results pointer to first row of results

Syntax: DBREWIND
Arguments: None
Options: None

This script command is used to reset the query results pointer back to the first row of the most recent query issued by the DBQUERY script command.

The DBGETRESULTS command is used to sequentially retrieve row(s) resulting from a query. You'd issue the DBREWIND command if you ever wish to have DBGETRESULTS start over again from the first row.

Related Command(s): [DBGETRESULTS](#), [DBQUERY](#)

DBUSE -- Create and/or open a SQL database file

Syntax:	DBUSE	[dbfile] [/options]
Arguments:	[dbfile]	Variable or string defining a database file name; if no path is defined, FileLink's working folder is used.
Options:	/new	Create the file if it does not exist.
	/odbc	Use an ODBC Data Source Name (DSN) instead of a filename.
	/pw=xx	Optional password to use with the /odbc option.
	/user=xx	Optional username to use with the /odbc option.

This script command is used to open a database file or connection. Only one database may be open at any time. Once the database file or connection is open you can use the [DBQUERY](#) script command to execute SQL statements and the [DBGETRESULTS](#) command to process the results of those statements. Execute the [DBCLOSE](#) command to stop using a database file or connection.

Database Files

Call the DBUSE command with no options to open an existing SQLite database file. Add the **/new** option if the database file does not exist and you want a new file to be created. When a new file is created it is just an empty database - you must create one or more tables in the database file via the [DBQUERY](#) script command before data rows may be inserted. If a database file does exist then the **/new** option empties it.

Note: The built-in SQLite database engine is unable to process queries that return more than 1000 rows of results.

ODBC Database Connections

Use the **/odbc** option to specify that the [file name] argument refers to the name of a ODBC User or System Data Source Name (DSN.) File DSNs are not supported. You can create DSN records using the *Data Sources* tool in the *Administrative Tools* folder under the Control Panel. This tool is named *ODBC Data Source Administrator* in some versions of Windows. The exact steps for creating a DSN depends on the features of your database's ODBC driver and is beyond the scope of this help topic. Please contact your local database administrator to resolve DSN configuration issues.

Note: FileLink is a 32 bit application. If you want to use a User or System DSN on a 64 bit version of Windows you must use the 32-bit version of the *ODBC Data Source Administrator* by running %systemroot%**syswow64**\odbcad32.exe. Be sure to use the full path because the 64-bit version of the file is named %systemroot%**system32**\odbcad32.exe so it is quite easy to run the wrong version by mistake. The name of your 32-bit DSN must be unique; your queries may fail if your computer also has a 64-bit DSN defined that shares the same name.

The **/user** and **/pw** options can optionally be combined with the **/odbc** option if you don't save database access credentials in your DSN. Creating a single shared System DSN without stored credentials may be preferable to creating separate User DSN records for each user account on a shared computer.

Related command(s): [DBQUERY](#), [DBGETRESULTS](#), [DBCLOSE](#), [DBREWIND](#),

DEC -- Decrement a variable by one

Syntax: DEC [variable]
Arguments: [variable] A [variable](#) containing from one to six numeric characters.
Options: None

This script command is used to decrement a variable by one. If each character in the variable is not numeric (e.g., digits 0 - 9), then the command fails.

Numeric strings used in the DEC (and [INC](#)) command are assumed to be a fixed length of one to six characters and contain leading zeros. When the value of a variable goes negative, the value wraps (i.e., a two character string is less than **00** after subtracting one then the value set to **99**; **0** wraps to **9**; **000** wraps to **999**; etc.)

Caution

This command is primarily intended to provide a mechanism for sequentially naming files, not as a simple numeric function. So its behavior of wrapping from **00** to **99**, for example, may not be appropriate when doing simple arithmetic. The DEC command may be used for both purposes but you need to remain aware of the command's behavior.

If the variable is not previously assigned, the variable is created and set equal to **000**.

Consider the example below where a variable is used to retrieve sequentially named files (using an decrementing file extension - i.e., file.999, file.998, etc.) from a remote system. Note: numeric values should be assigned enclosed in quotation marks when strings with leading zeroes are desired.

```
SET basename = "file."  
SETNUM counter = "000"  
:loop  
DEC counter  
SET file name = basename & counter  
DISPLAY filename  
RCVFILE file_name  
GOTO loop
```

In another use, consider the example below where a variable is used as a decrementing loop counter.

```
SETNUM counter = 10  
;; loop 10 times  
:loop  
DISPLAY counter  
DEC counter  
IFNUM! counter 0 goto loop
```

Related Command(s): [IFNUM](#), [INC](#), [SET](#), [SETNUM](#)

DELDIR -- Delete an empty local folder

Syntax: DELDIR [folder name]
Arguments: [folder name] [Variable](#) or [string](#) defining the local folder name to delete.
Options: None

This script command deletes an *empty* local folder (also referred to as a directory). Consider the following example in which a folder is deleted on the E: drive.

```
DELDIR "e:\newbie"
```

Related Command(s): [CHGDIR](#), [MAKEDIR](#), [WORKINGDIR](#)

DELETE -- Delete a local file

Syntax: DELETE [file name]
Arguments: [file name] [Variable](#) or [string](#) defining a file or path name; if no path is defined, FileLink's working folder is used.
Options: None

This script command deletes the specified file.

Related Command(s): [COPY](#), [APPEND](#), [RENAME](#), [WORKINGDIR](#)

DIAL -- Initiate modem auto-dialer

Syntax: DIAL [phone #] | [/options]
Arguments: [phone #] [Variable](#) or [string](#) defining the telephone number of the remote modem that FileLink is to dial; if the phone number is omitted, the default phone number is dialed; the [phone #] string may include modem command modifiers.
Options: /timeout=*nn* Dial time-out in seconds; if 0, wait indefinitely

This script command passes the phone number argument to the modem auto-dialer and waits for a connection. Use this command for modem connections to asynchronous hosts for TTY or file transfer .

Related Command(s): [CONNECT](#), [ANSWER](#), [MODEMRESET](#)

DIFF -- Look for differences in the local PC file system

Syntax:	DIFF	[path] [dbfile] [/options]
Arguments:	[path]	Optional Variable or string defining the starting local path from which to begin looking for differences; if omitted, the current working folder is used and [dbfile] defaults to "snapshot_local.sql".
	[dbfile]	Optional Variable or string defining an alternative to the default "snapshot_local.sql" database file where a previous snapshot has been saved.
Options:	/includirs	Look for differences in the current or specified folder and all subfolders thereunder.
	/noupdate	Do not update the snapshot with differences found.

This script command is used in conjunction with the [SNAPSHOT](#) and [GETDIFF](#) script commands to locate individual file differences (i.e., change in size, date/time stamp) within a specified folder (and optional subfolder) tree within the local PC file system.

SNAPSHOT is the first step to establish a baseline (or "snapshot") of the specified folder(s) from which to determine if any file(s) change. DIFF is used subsequently to compare the current state of the file system with what was saved in [dbfile]. Any changes found are saved back into the same database file (to be processed using the GETDIFF command) and the original snapshot is updated to reflect the current state of the file system (unless this is suppressed by the inclusion of the **/noupdate** option).

If the [dbfile] argument is specified it must be the name of a database file previously created by the SNAPSHOT command. If omitted the default "snapshot_local.sql" file is used.

Consider the following example which compares the current working folder and any subfolders with the snapshot saved in the default "snapshot_local.sql" file.

```
DIFF "*" "*" /includirs
```

The total number of differences found in the local PC file system is saved in the **%difffiles** script variable.

Related Command(s): [DIFFREWIND](#), [GETDIFF](#), [SNAPSHOT](#)

DIFFREWIND -- Reset file pointer for GETDIFF command

Syntax: DIFFREWIND
Arguments: None
Options: None

This script command is used in conjunction with the [GETDIFF](#) command to reset the file pointer to the first difference recorded in the database.

You might use this command to “rewind” to the beginning of a series of differences being examined with the GETDIFF command if you need to repeat some process.

Related Commands: [GETDIFF](#)

DISPLAY -- Display all or a specified variable

Syntax: DISPLAY [variable]
Arguments: [variable] A previously defined [variable](#) (optional).
Options: None

This script command may be used to output all variables currently assigned or one specific variable. The variable name(s) and associated value(s) are written to the FileLink window and to the log file. Both internal and user defined variables are included.

Related Command(s): [SET](#)

DISCONNECT -- Disconnect the line

Syntax: DISCONNECT
Arguments: None
Options: None

This script command is used to disconnect FileLink from the remote system. On a COM port, this command drops the DTR and RTS modem signals.

Related Command(s): [DIAL](#), [CONNECT](#), [ANSWER](#).

DOSCMD -- Execute an MS-DOS command

Syntax: DOSCMD [cmd]
Arguments: [cmd] [Variable](#) or [string](#) defining an MS-DOS command such as
 dir, **copy**, **mkdir**, etc. to be executed.
Options: None

This script command executes a specified MS-DOS internal command. FileLink is suspended until the execution of the command is complete.

Consider the following example that lists the contents of the current folder to a file.

```
DOSCMD "dir *.* > tempfile"
```

Consider the following example where a new folder is created and the contents of the current folder is copied there.

```
DOSCMD "mkdir \newfolder"  
DOSCMD "copy *.* \newfolder"
```

Upon return, any exit code from the process launched with DOSCMD is saved in the %**lasterror** script variable and can be tested with any of the [IFERROR](#) commands. For example:

```
DOSCMD "fc file1.txt file2.txt"  
;; 'fc' returns 2 if it cannot compare the specified files  
IFERROR= 2 GOTO fc_error
```

Related Command(s): [CALL](#), [CHAIN](#), [EXEC](#)

See also: [Script File Result Codes](#)

ENDFUNCTION -- End function declaration

Syntax: ENDFUNCTION
Arguments: None
Options: None

This script directive is used to mark the end a function.

Consider the following example.

```
BEGINFUNCTIONS  
FUNCTION MyFunction  
;; body of MyFunction  
ENDFUNCTION  
ENDFUNCTIONS
```

Related Command(s): [BEGINFUNCTIONS](#), [ENDFUNCTIONS](#), [FUNCTION](#), [RETURN](#)

ENDFUNCTIONS -- End function declaration section

Syntax: ENDFUNCTIONS
Arguments: None
Options: None

This script directive is used to end the function declaration section of a script file. A function declaration section must be at the beginning of the script file (before any functions are called) and must not be used in any script file invoked with a [CALL](#) script command.

Consider the following examples.

```
BEGINFUNCTIONS  
FUNCTION MyFunction  
;; body of MyFunction  
ENDFUNCTION  
ENDFUNCTIONS
```

Related Command(s): [BEGINFUNCTIONS](#), [ENDFUNCTION](#), [FUNCTION](#), [RETURN](#)

EXEC -- Execute a external program

Syntax:	EXEC	[exec name] [/options]
Arguments:	[exec name]	Variable or string defining a file or path name of an executable program including any program command line arguments; if no path is defined FileLink's working folder is used; MS-DOS internal commands such as dir are not supported.
Options:	/nowait	Continue script execution after starting external program.
	/passargs	Any subsequent variables or strings following this option are passed as arguments to the program to be run.
	/passerror	Pass the result code from the last script command as an argument to the executed program.

This script command executes a specified external program.

The **/nowait** option determines if the script file execution is suspended until the external program is complete or continues to run. If this option is omitted, script file execution resumes when the program terminates.

The **/passargs** option allows you to pass any number of variable or string values from the script file to the program to be executed as if they specified on the Shortcut Target command line. Any variables or string values that follows the **/passargs** delimiter within the EXEC command line are passed in this way. Consider the following example.

```
SET program_name = "myprog.exe -abc"
SET flag = "-xyz"
EXEC program_name /passargs "-p2" flag
```

The effective command line to the program **myprog.exe** would be:

```
myprog.exe -abc -p2 -xyz
```

The exit value of the executed program is obtained by FileLink when it terminates and it becomes the EXEC command's result code. This result code may be useful in subsequent conditional branching tests in the script file.

Consider the following example where you have a program called **password.exe** which checks for a valid password string. The program returns 0 if the password is valid and 1 otherwise. The following script file would verify a password and then send a file to the caller only if an acceptable password was received.

```
ANSWER /timeout=0
;; wait to receive password text
LINEIN rcvd_password
;; check for valid password in variable 'rcvd_password'
EXEC "password.exe" /passargs rcvd_password
;; don't send if invalid password
IFERROR goto disconnect
```

```
;; send the file
SENDFILE file1
:disconnect
DISCONNECT
EXIT
```

The **/passerror** switch causes the result code from the last script command to be passed an argument to the executed process. The form of the argument is **/flerr=xx**. Consider the following example where the previous script command terminated with a result code of 1013.

```
SET program_name = "myprog.exe -abc"
EXEC "program_name" /passerror
```

The effective command line to the program **myprog.exe** would be:

```
myprog.exe -abc /flerr=1013
```

This option permits an external program to deal specifically with a particular error condition that may be beyond the scope of a FileLink script file.

Related Command(s): [CALL](#), [CHAIN](#), [DOSCMD](#), [WORKINGDIR](#)

See also: [Script File Result Codes](#)

EXIT -- Quit FileLink

Syntax: EXIT [/options]
Arguments: [/options]
Options: /exitcode=*nn* Define exit value for FileLink.
 /forced Exit even if line is connected.

This script command ends script processing and terminates FileLink. The action is the same as clicking the Exit button on the FileLink toolbar or pressing the Alt-F4 key.

The **/exitcode=*nn*** option allows you to define the program exit code passed back to the operating system or calling process

The **/forced** option performs a *forced quit* that terminates any I/O in progress and disconnects the line prior to exiting.

Related Command(s): [STOP](#)

EXPORT -- Export Configuration Settings

Syntax:	EXPORT	[file name] [/options]
Arguments:	[file name]	Variable or string defining a file or path name; if no path is defined FileLink's working folder is used.
Options:	/inclpw	Include this option if you wish to export your encoded PGP private passphrase along with the other FileLink settings. Generally it is NOT recommended to include your passphrase in settings that you are distributing unless you are in control of all locations where these settings may be distributed.

This script command exports the current configuration settings stored in the Windows registry to a file. This file may be used in a subsequent import operation to restore a previous set of configuration settings, to distribute identical settings to multiple sites of FileLink, or may be requested by technical support to ascertain how FileLink is configured.

Settings may also be exported manually using the **Export Settings...** command under the **File** menu.

Settings may be imported using the IMPORT script command, by the [-g command line](#) switch, or **Import Settings...** command under the **File** menu.

Important

Specify the **/inclpw** option **ONLY** if you wish to export your encoded PGP private passphrase along with the other FileLink settings. Generally it is **NOT** recommended to include your passphrase in settings that you are distributing unless you are in control of all locations where these settings may be distributed.

Related command(s): [IMPORT](#)

FLUSH -- Flush characters from receive buffer

Syntax: FLUSH
Arguments: None
Options: None

This script command flushes the FileLink receive buffer. Use this command to discard any characters that might have been received spuriously before issuing a LINEIN or RCVFILE command.

Related Command(s): [LINEIN](#), [RCVFILE](#)

FUNCTION -- Begin a function declaration

Syntax:	FUNCTION	[func name] [arg1 ... arg9]
Arguments:	[func name]	Variable or string specifying a previously declared function name.
	[arg1 ... arg9]	Up to nine variables that are assigned when the function is called.
Options:	None	

This script directive is used within the function declaration section of a script file to define a function named [func name] and enable it to be called during execution of the script file.

Function names may be whatever the script developer chooses to use as long as they do not conflict with script command identifiers.

Up to nine arguments may be passed to a function. When a function declaration is performed, variables with names matching the arguments are created. Since all variables in a FileLink script are global, function arguments should be unique from variables that may appear elsewhere in the script.

Consider the following example where two strings are passed as arguments to a function, assigned to variables within the function, and displayed.

```
;; declare our function
BEGINFUNCTIONS
FUNCTION MyFunction arg1 arg2
DISPLAY arg1
DISPLAY arg2
RETURN
ENDFUNCTION
ENDFUNCTIONS

;; script execution begin here
MyFunction "a" "b"
STOP
```

Related Command(s): [BEGINFUNCTIONS](#), [ENDFUNCTION](#), [ENDFUNCTIONS](#), [RETURN](#)

GETDIFF -- Get specific changes within local PC file system

Syntax:	GETDIFF	[path] [dbfile]
Arguments:	[path]	Optional Variable or string defining the starting local path from which to begin looking for differences; if omitted, the current working folder is used and [dbfile] defaults to "snapshot_local.sql".
	[dbfile]	Optional Variable or string defining an alternative to the default "snapshot_local.sql" database file where a previous snapshot and diff comparison has been saved.
Options:	None	

This script command is used in conjunction with the [SNAPSHOT](#) and [DIFF](#) script commands to locate individual file differences (i.e., change in size, date/time stamp) within a specified folder (and optional subfolder) tree within the local PC file system.

SNAPSHOT is the first step to establish a baseline (or "snapshot") of the specified folder(s) from which to determine if any file(s) change and DIFF is used subsequently to compare the current state of the file system with what was saved in [dbfile]. Any changes found are saved back into the same database file which are then processed by this script command.

If the [dbfile] argument is specified it must be the name of a database file previously created by a previous DIFF command. If omitted the default "snapshot_local.sql" file is used.

When GETDIFF finds a changed file, the full path name of the changed file is returned in the **%difffilepath** script variable, just the file name itself in **%difffilename**, a description of the difference found is returned in the **%difffiletext** script variable, and a corresponding numeric representation is saved in the **%difffileid** script variable. The sequential file number for this difference is returned in the **%diffnum** script variable.

The possible differences are:

\$DIFF_FILE_NOT_FOUND	5001	**File not found
\$DIFF_FILE_IS_NEW	5002	** File is new
\$DIFF_FILE_SIZE	5003	** File size has changed
\$DIFF_FILE_DATETIME	5004	** File date/time stamp has changed

GETDIFF must be performed once for each difference that was detected by the DIFF command and correspondingly saved in [dbfile]. What follows is a simple example that shows how GETDIFF might be used inside a loop to write all the differences found to a text file.

```

DIFF "*" "*"
:loop
GETDIFF
IFERROR $ERROR_READ_EOF GOTO done
SET Diff = "File: " + %difffilepath + " " + %difffiletext
WRITEFILE "differences.txt" Diff /append
GOTO loop

```



```
:done
```

The following example shows how all new files found are uploaded to the remote system.

```
DIFF "*" /includirs
:loop
GETDIFF
IFERRORIFERROR $ERROR_READ_EOF GOTO done
IFNUM! = %difffileid $DIFF_FILE_IS_NEW GOTO loop
SENDFILE %difffilepath
GOTO loop
:done
```

Related Commands: [DIFF](#), [DIFFREWIND](#), [SNAPSHOT](#)

GETFILE -- Get file from folder structure on local PC

Syntax:	GETFILE	[file name] [/options]
Arguments:	[file name]	Variable or string defining a file or path name to look for; wildcard characters allowed; if no path is defined FileLink's working folder is used.
Options:	/includirs	Descend into local folder(s) as they are found.
	/newest	Get the newest file in the folder
	/oldest	Get the oldest file in the folder.
	/timeout= <i>nn</i>	Time-out in seconds to wait for presence of the file; if this option is omitted, FileLink looks for the file and if nothing is found, \$ERROR_NO_FILE_FOUND is returned; otherwise \$ERROR_WAIT_TIMED_OUT is returned.

This script command checks for (and optionally waits for) the existence of a file defined by the [file name] argument. If a matching file is detected, its file name is saved in the **%nextfile** variable and its full path name is saved in the **%nextpath** variables. The date and time of the file are also saved in the **%nextfiledate**, **%nextfiledatetime**, and **%nextfiletime** variables. The size of the file, in bytes, is saved in the **%nextfilesize** variable.

This command is functionally equivalent to the [GETNEXTFILE](#) script command except for two important differences.

GETFILE has the capability to descend into and out of subfolders that it finds in a given folder tree (assuming the **/includirs** option is specified) while GETNEXTFILE does not. (GETNEXTFILE returns folder names it finds in the **%nextfolder** script variable instead.)

Secondly, GETFILE does away with the **/next** option. GETFILE will always return the next file found when called repeatedly on the same folder structure when there are wildcard character (s) in [file name]. In the event that you wish to start over with the first matching file in the folder structure, issue the GETREWIND command and then resume issuing GETFILE.

Otherwise refer to the description of GETNEXTFILE for more details on this command.

Related Commands: [GETREWIND](#)

GETMAIL -- Get an e-mail message

Syntax:	GETMAIL	[server] [subj] [mail file] [/options]
Arguments:	[server]	Variable or string defining the server URL or IP address (e.g., pop3.mail.server or 209.198.128.17) of the POP mail server; the server port is always set to 110.
	[subj]	Variable or string for the message subject line; [subj] must be a variable initialized to an empty string to get the first available message (the subject line of the message is returned by way of this variable); or [subj] must be a variable or string initialized to a non-empty value specifying the subject line of a specific message to look for on the server (searches are all inclusive and case-sensitive).
	[mail file]	Variable or string defining the optional file name to which the received message is written; if [mail file] is an empty string then the message is discarded.
Options:	/nodelete	Do not delete the message from the server (leave on server).
	/pw=xx	Define the password to use when logging on to the incoming mail server.
	/timeout=nn	Time-out, in seconds, to wait for message to be received (if omitted the time-out is set to 30 seconds).
	/user=xx	Define the user name to use when logging on to the mail server.
	/view	View the message in a pop-up window.

This command either gets the first e-mail message available on the specified POP3 server or searches all messages for a one with a matching subject line. Received message may optionally be viewed and/or saved to a file for processing by another e-mail client or application. The file contains the full e-mail message, including headers, in **.eml** format.

When the [subj] variable is set to an empty string when calling GETMAIL, FileLink gets the next available message on the server. When using GETMAIL in this way, it is recommended to use a dedicated mailbox to prevent FileLink from possibly obtaining non-relevant messages. Upon return, the [subj] variable will contain the subject line of the message downloaded. Use of the **/nodelete** option is not advised since GETMAIL will return the same message over and over again unless it is deleted from the server by another user or process.

When searching for a specific message, the [subj] variable must be initialized to the desired subject line prior to calling GETMAIL. Messages not matching the subject line comparison are left undisturbed on the server. If you only wish to check for the presence of a specific message (and leave it on the server too), use the **/nodelete** option.

Consider the following example in which the first available e-mail message is received but not saved to a file, and the message is deleted from the server after it is received.

```
SET server = "pop3.mail.server"
SET subj = ""
;; subj argument MUST be a variable since subject line
;; found in the message is returned
```

```
GETMAIL server subj "" /user=pop3id /pw=pop3pw
```

In the following example, the first available e-mail message is obtained and written to a uniquely named file.

```
SET server = "pop3.mail.server"
SET subj = ""
MAKEFILENAME file "eml" "c:\email messages" "mail"
GETMAIL server subj file /user=pop3id /pw=pop3pw
```

In the following example, e-mail messages on the server are search for a specific subject. If found, the message is downloaded, saved to a file, but is left on the server.

```
SET server = "pop3.mail.server"
MAKEFILENAME file "eml" "c:\email messages" "mail"
;; when subj (variable or string) is not empty, search for this
;; subject line in available messages (nothing is returned)
GETMAIL server "this one" file /user=pop3id /pw=pop3pw /nodelete
```

In the preceding example, the same message would be returned if GETMAIL was executed again since **/nodelete** was specified.

Related Command(s): [CREATEMAIL](#), [SENDMAIL](#)

GETNEXTFILE -- Get file or folder names on local PC

Syntax:	GETNEXTFILE	[file name] [/options]
Arguments:	[file name]	Variable or string defining a file or path name to look for; wildcard characters allowed; if no path is defined FileLink's working folder is used.
Options:	/includirs	Return local folder name(s) as they are found. (Formerly this was the /subdirs option.)
	/newest	Get the newest file in the folder.
	/next	Get the next file or folder name in a local folder; omit this option to obtain the first file from a given local folder; if you wish to obtain subsequent files from this same folder, use this option until you find the desired file or no more files are found (see Important below).
	/oldest	Get the oldest file in the folder.
	/timeout= <i>nn</i>	Time-out in seconds to wait for presence of the file; if this option is omitted, FileLink looks for the file and if nothing is found, \$ERROR_NO_FILE_FOUND is returned; otherwise \$ERROR_WAIT_TIMED_OUT is returned.

This script command checks for (and optionally waits for) the existence of a file defined by the [file name] argument. If a matching file is detected, its file name is saved in the **%nextfile** variable and its full path name is saved in the **%nextpath** and **%nextfolder** variables. The date and time of the file are also saved in the **%nextfiledate**, **%nextfiledatetime**, and **%nextfiletime** variables. The size of the file (excluding a folder), in bytes, is saved in the **%nextfilesize** variable.

The use of this command creates what is referred to as the "hot send" function whereby FileLink automatically sends files when they are placed in a known location.

If a file is open by another application, it will not be 'seen' by this command. Furthermore, the same file will be returned on subsequent iterations of this command unless it is deleted or renamed.

The **/next** option allows the entire contents, both files and folder names, of local folders to be traversed and saved in variables for use in script processing. Without this option, the first file found in the specified folder will always be returned.

Important

For most reliable operation, use the **/next** option only on static local folders (i. e., where no new files are being added and none are being deleted) since each successive **/next** skips one file as FileLink scans through a given folder. When **/next** is omitted, this command always returns the first file that appears in the folder (files are returned in unsorted folder order).

Consider the following example where FileLink waits indefinitely for any file with a **.txt** extension to be created in its working folder.

```
GETNEXTFILE "*.txt" /timeout=0
```

Once such a file exists, its name is saved in the **%nextfile** variable, its path and name is saved

in the **%nextpath** variable, and script execution resumes.

Consider the following example where all the files in the specified folder are sent to the remote system.

```
:label
GETNEXTFILE "\upload_dir\*.*" /timeout=2
IFERROR= $ERROR_WAIT_TIMED_OUT goto sent_last_file
SENDFILE %nextfile /type=BIN
IFERROR goto xmt_error
DELETE %nextfile
GOTO label
:sent_last_file
```

Consider the following example in which the file and folder names in the current local folder are identified and displayed by using the **/next** option.

```
GETNEXTFILE "*.*" /includirs
:loop
IFSTRCMP %nextfile "" goto dir
!DISPLAY %nextfile
!DISPLAY %nextfolder
!DISPLAY %nextpath
PAUSE /for=1
GOTO next
:dir
IFSTRCMP %nextfolder "" goto error
!DISPLAY %nextfile
!DISPLAY %nextfolder
!DISPLAY %nextpath
PAUSE /for=1
:next
GETNEXTFILE "*.*" /next /includirs
IFERROR= $ERROR_NO_FILE_FOUND goto finish
GOTO loop
:error
MESSAGEBOX "This shouldn't ever appear..."
STOP
:finish
MESSAGEBOX "Shown all files!"
```

The **/newest** and **/oldest** options are normally used with a wildcard [file name] to obtain the name of the newest or oldest file present. Use of these options with the **/next** option is not supported. Consider the following example where all files are sent to the remote system beginning with the newest.

```
:next_file
```

```
GETNEXTFILE *.*" /newest
IFERROR= $ERROR_NO_FILE_FOUND goto label
SENDFILE %nextfile
DELETE %nextfile
GOTO next_file
:label
```

The **/includirs** option may be used if you wish local folder names to be returned along with regular file names as they are found. When a folder is found, it is saved in the **%nextfolder** variable and the **%nextfile** variable is set to an empty string. The **/oldest** and **/newest** options have no effect on the folder names returned by the GETNEXTFILE command. Consider the following example that shows how folder files are distinguished from other files.

```
:look_for_folder
GETNEXTFILE *.*" /includirs /next
IFERROR= $ERROR_NO_FILE_FOUND goto error
IFNSTRCMP %nextfile "" goto look_for_folder
; both %nextfile and %nextfolder will not be empty at same time
DISPLAY %nextfolder
GOTO look_for_folder
:error
```

The date and time of the file obtained with the command is saved to three internal variables named **%nextfiledate**, **%nextfiledatetime**, and **%nextfiletime**. This permits you to directly compare the file's date and time to values of your choosing. Consider the following example that shows how a file newer than a specified date may be found.

```
:not_new
GETNEXTFILE *.*" /next
IFERROR= $ERROR_NO_FILE_FOUND goto error
SET filedate = "file date is " & %nextfiledate
DISPLAY filedate
IFDATE< "06-30-02" goto not_new
MESSAGEBOX "found file created after June 30, 2002"
STOP
:error
```

Related Commands: [WORKINGDIR](#)

See also: [Using The %nextfile, %nextpath, and %nextfolder Variables,](#)

[Using The %nextfiledate, %nextfiledatetime, %nextfilesize, and %nextfiletime Variables,](#)

GETREWIND -- Reset file pointer for GETFILE command

Syntax: GETREWIND
Arguments: None
Options: None

This script command is used in conjunction with the [GETFILE](#) command to reset the file pointer to the first wildcard matching file.

You might use this command to “rewind” to the beginning of a series of files found with the GETFILE command if you need to repeat some process.

Related Commands: [GETFILE](#)

GO -- Rerun the currently defined script file

Syntax: GO
Arguments: None
Options: None

This command is intended to be used from the console command line during debugging to (re) run the currently selected script file. This command has the same effect as clicking the Rerun Script File (**Ctrl + R**) button on the toolbar.

The GO command is not supported as a command within a script file itself - it is only for use from the console command line during script debugging.

Related Command(s): [BREAK](#), [RESUME](#), [STOP](#)

GOTO -- Direct flow to label

Syntax: GOTO [label]
Arguments: [label] A valid [label](#) within the current script file.
Options: None

This command directs the flow of execution within a script file to [label]. Even though script file labels begin with a colon, the colon is omitted in the GOTO (and other branching) commands. For example:

```
:loop  
...  
GOTO loop
```

See also: [Labels In Script Files](#)

IFDATE -- Conditional branch upon file date comparison

Form 1

Syntax: IFDATExx [date1] [date2] goto [label]

Forms: IFDATE= [date1] [date2] goto [label]
 IFDATE< [date1] [date2] goto [label]
 IFDATE> [date1] [date2] goto [label]
 IFDATE<= [date1] [date2] goto [label]
 IFDATE>= [date1] [date2] goto [label]
 IFDATE! = [date1] [date2] goto [label]

Arguments: [date1] [Variable](#) or [string](#) defining a date in the format of **mm.dd.yy**
 [date2] Variable or string defining a date in the format of **mm.dd.yy**
 [label] A valid [label](#) within the current script file which is branched to if the date condition is satisfied.

Options: none

Form 2

Syntax: IFDATE [cond] goto [label]

Arguments: [cond] [Variable](#) or [string](#) defining a date condition to test against the internal **%comparedate** variable.
 [label] A valid [label](#) within the current script file which is branched to if the date condition is satisfied.

Options: none

Form 3

Syntax: IFDATExx [date] goto [label]

Forms: IFDATE= [date] goto [label]
 IFDATE< [date] goto [label]
 IFDATE> [date] goto [label]
 IFDATE<= [date] goto [label]
 IFDATE>= [date] goto [label]
 IFDATE! = [date] goto [label]

Arguments: [date] [Variable](#) or [string](#) defining a date in the format of **mm.dd.yy** to compare against the date stamp of the file obtained with the most recent GETNEXTFILE script command.
 [label] A valid [label](#) within the current script file which is branched to if the date condition is satisfied.

Options: none

Form 1 of this command is used to compare two specified date strings in the format of **mm.dd.yy**.

Form 2 of this command is used in conjunction with the *most recent* to compare the date of the file with a specified date string.

Form 3 of this command is used in conjunction with the *most recent* GETNEXTFILE to compare the date of the file with a specified date string.

When specifying a date, leading zeroes are required. The following are examples of valid date strings.

```
01-30-00    January 30, 2000
12-25-99    December 25, 1999
```

Syntactically, no space is permitted to the left of the '!', '=', '<' or '>' symbols, and a space is required to the right of these symbols.

Example of Form 1

The following conditional branch is taken if the current date is past June 30, 2003.

```
IFDATE> %date "06-30-03" goto later_date
```

Example of Form 2

The following conditional branch is taken if a local file just obtained with the GETNEXTFILE command has a date stamp after June 30, 2003.

```
GETNEXTFILE "*.txt"
IFDATE> "06-30-03" goto good_file
```

Example of Form 3

The following conditional branch is taken if a local file just obtained with the GETNEXTFILE command has a date stamp after June 30, 2003.

```
GETNEXTFILE "*.txt"
IFDATE> "06-30-03" goto_good
```

Related Command(s): [GETNEXTFILE](#), [IFSIZE](#), [IFTIME](#)

IFERROR -- Conditional branch after testing result code

Syntax:	IFERRORxx	[rcode] goto [label]
Forms:	IFERROR	[rcode] goto [label]
	IFERROR=	[rcode] goto [label]
	IFERROR<	[rcode] goto [label]
	IFERROR>	[rcode] goto [label]
	IFERROR<=	[rcode] goto [label]
	IFERROR>=	[rcode] goto [label]
	IFERROR!=	[rcode] goto [label]
Arguments:	[rcode]	Optional numeric value or \$ERROR variable associated with result codes returned by script commands upon completion; any successful operation returns a value of 0.
	[label]	A valid label within the current script file which is branched to if the conditional test is successful.
Options:	none	

This script command checks the result of the previously executed command to see if **IFERRORxx [rcode]** is true. If so, the script file branches to [label], otherwise execution continues with the next command.

The [rcode] may be omitted if you do not wish to test for a specific error condition. A value of 1 is assumed, which results in testing for any error condition. The following variations of the IFERROR command are supported:

IFERROR=	(equal to)
IFERROR<	(less than)
IFERROR>	(greater than)
IFERROR<=	(less than or equal to)
IFERROR!=	(not equal to)
IFERROR or IFERROR>=	(greater than or equal to)

Syntactically no space is permitted to the left of the '=', '<' or '>' symbols, and a space is required to the right of these symbols.

Consider the following examples.

```
;; if result code is greater than 1046 then branch to label
;; 'next', else continue with the next line
IFERROR> 1046 goto next
;; if result code equals predefined $ERROR variable then branch
;; to label 'next'
IFERROR $ERROR_CONNECT_TIMEOUT goto next
```

Related Command(s): [LOOPIF](#), [GOTO](#)

See also: [Script File Result Codes](#)

IFFILE -- Conditional branch on file existence

Syntax:	IFFILE	[file name] goto [label]
Arguments:	[file name]	Variable or string defining a file or path name; if no path is defined FileLink's working folder is used.
	[label]	A valid label within the current script file which is branched to if the specified file exists.
Options:	none	

This script command checks for the existence of the specified file and branches to [label] if the file exists. FileLink attempts to open the file to ascertain if it exists or not, so make sure that you have the proper privileges to access the file.

For example:

```
IFFILE "c:\Program Files\FileLink\thisfile" goto found_it
```

Note: Wildcard characters in [file name] are not permitted.

Related Command(s): [IFNFILE](#), [WORKINGDIR](#)

IFNFILE -- Conditional branch on file non-existence

Syntax:	IFNFILE	[file name] goto [label]
Arguments:	[file name]	Variable or string defining a file or path name; if no path is defined, FileLink's working folder is used.
	[label]	A valid label within the current script file which is branched to if the specified file does not exist.
Options:	none	

This script command checks for the existence of the specified file and branches to [label] if the file does not exist. FileLink attempts to open the file to ascertain if it exists or not, so make sure that you have the proper privileges to access the file.

For example:

```
IFNFILE "c:\Program Files\FileLink\thisfile" goto not_found
```

Note: Wildcard characters in [file name] are not permitted.

Related Command(s): [IFFILE](#), [WORKINGDIR](#)

IFNO -- Conditional branch if 'No' is clicked in ASK dialog box

Syntax: IFNO goto [label]
Arguments: [label] A valid [label](#) within the current script file.
Options: None

This command not supported when running in a minimized window or as an NT Service.

This script command is used in conjunction with the ASK command. If you click the 'No' button in the ASK dialog box, this command branches to [label].

For example:

```
ASK "Yes or No?"  
IFNO goto answer_is_no
```

Related Command(s): [ASK](#), [IFYES](#)

IFNSTRCMP -- Conditional branch when two string variables are not equal

Syntax: IFNSTRCMP [string1] [string2] goto [label]
Alt Syntax: IFNSTRCMI [string1] [string2] goto [label]
Arguments: [string1] [Variable](#) or [string](#) defining first string to compare.
 [string2] Variable or string defining second string to compare.
 [label] A valid [label](#) within the current script file which is branched to if the two strings do not match.
Options: none

This script command compares two strings and branches to [label] if they are not exactly the same.

Use the alternate IFNSTRCMI command to perform the same comparison but to ignore case during the comparison.

Consider the following example in which the script file accepts a string from a user prompt and branches to a label if the string is not what is expected.

```
PROMPT serial_number "Enter serial number"  
IFNSTRCMP serial_number "010101" goto invalidnumber
```

Related Command(s): [IFSTRCMP](#), [IFSUBSTR](#), [IFNSUBSTR](#)

IFNSUBSTR -- Conditional branch if sub-string is not found in string variable

Syntax: IFNSUBSTR [string] [substring] goto [label]
Alt Syntax: IFNSUBSTRI [string] [substring] goto [label]
Arguments: [string] [Variable](#) or [string](#) defining master string.
[substring] Variable or string defining substring to find.
[label] A valid [label](#) within the current script file which is branched to if the substring is not contained within the master string.
Options: none

This script command attempts to locate a substring with a specified master string and branches to [label] if it is not found.

Use the alternate IFNSUBSTRI command to perform the same comparison but to ignore case during the comparison.

Consider the following simple example that results in a branch to the specified label.

```
SET master_string = "abcdefg"  
SET substring = "xyz"
```

```
IFNSUBSTR master_string substring goto substring_not_found
```

Related Command(s): [IFSTRCMP](#), [IFNSTRCMP](#), [IFSUBSTR](#)

IFNUM -- Conditional branch upon numeric variable comparison

Syntax:	IFNUMxx	[num1] [num2] goto [label]
Forms:	IFNUM=	[num1] [num2] goto [label]
	IFNUM<	[num1] [num2] goto [label]
	IFNUM>	[num1] [num2] goto [label]
	IFNUM<=	[num1] [num2] goto [label]
	IFNUM>=	[num1] [num2] goto [label]
	IFNUM! =	[num1] [num2] goto [label]
Arguments:	[num1]	Variable, string , or numeric constant defining the first numeric value to compare against.
	[num2]	Variable, string, or numeric constant defining the second numeric value to compare against.
	[label]	A valid label within the current script file which is branched to if the numeric condition is satisfied.
Options:	none	

This command is used to compare two variables, strings that contain numeric values (e.g., contain only digits 0 - 9), or numeric constants. The command results in a syntax error if either value is non-numeric.

For example, the following conditional branch is taken if the numeric variable `x` is equal to 1000.

```
SETNUM x = 1000
IFNUM= x 1000 goto equal_value
```

The following conditional branch is taken if the numeric variable `x` is larger than 1000. (Numeric strings are used instead of numeric constants simply as an example of allowed syntax.)

```
SETNUM x = "2300"
IFNUM> x "1000" goto larger_value
```

Syntactically, no space is permitted to the left of the '!', '=', '<' or '>' symbols, and a space is required to the right of these symbols.

Related Command(s): [DEC](#), [INC](#), [SETNUM](#)

See also: [Performing Variable Arithmetic and Numeric Comparisons](#)

IFSIZE -- Conditional branch upon file size comparison

Syntax:	IFSIZExx	[size] goto [label]
Forms:	IFSIZE=	[size] goto [label]
	IFSIZE<	[size] goto [label]
	IFSIZE>	[size] goto [label]
	IFSIZE<=	[size] goto [label]
	IFSIZE>=	[size] goto [label]
	IFSIZE! =	[size] goto [label]
Arguments:	[size]	Variable , string , or numeric constant defining a numeric value to compare against the size of the file obtained with the most recent GETNEXTFILE script command.
	[label]	A valid label within the current script file which is branched to if the date condition is satisfied.
Options:	none	

This command is used in conjunction with the *most recent* GETNEXTFILE compare the size of the file with a specified numeric value.

The following conditional branch is taken if a local file just obtained with the GETNEXTFILE command has a size larger than 1000 bytes.

```
GETNEXTFILE "*" .txt "  
IFSIZE> 1000 goto good_file
```

Syntactically, no space is permitted to the left of the '!', '=', '<' or '>' symbols, and a space is required to the right of these symbols.

Related Command(s): [GETNEXTFILE](#), [IFDATE](#), [IFTIME](#)

IFSTRCMP -- Conditional branch when two string variables are equal

Syntax:	IFSTRCMP	[string1] [string2] goto [label]
Alt Syntax:	IFSTRCMPi	[string1] [string2] goto [label]
Arguments:	[string1]	Variable or string defining first string to compare.
	[string2]	Variable or string defining second string to compare.
	[label]	A valid label within the current script file which is branched to if the two strings match.
Options:	none	

This script command compares two strings and branches to [label] if they are exactly the same.

Use the alternate IFSTRCMPi command to perform the same comparison but to ignore case during the comparison.

Consider the following example in which the script file accepts a string from an operator and compares it to see if it is valid.

```
PROMPT user_id "Enter your User ID"  
IFSTRCMP user_id "BANK11" goto valid_user
```

Related Command(s): [IFNSTRCMP](#), [IFSUBSTR](#), [IFNSUBSTR](#)

IFSUBSTR -- Conditional branch if sub-string is found in string variable

Syntax: IFSUBSTR [string] [substring] goto [label]
Alt Syntax: IFSUBSTR1 [string] [substring] goto [label]
Arguments: [string] [Variable](#) or [string](#) defining master string.
 [substring] Variable or string defining substring to find.
 [label] A valid [label](#) within the current script file which is branched to if
 the substring is contained within the master string.
Options: none

This script command attempts to locate a substring with a specified master string and branches to [label] if it is found.

Use the alternate IFSUBSTR1 command to perform the same comparison but to ignore case during the comparison.

Consider the following simple example that results in a branch to the specified label.

```
SET master_string = "abcdefg"  
SET substring = "cde"  
IFSUBSTR master_string substring goto substring_found
```

Related Command(s): [IFSTRCMP](#), [IFNSTRCMP](#) , [IFNSUBSTR](#)

IFTIME -- Conditional branch upon time comparison*Form 1*

Syntax: IFTIMExx [time1] [time2] goto [label]
Forms: IFTIME= [time1] [time2] goto [label]
IFTIME< [time1] [time2] goto [label]
IFTIME> [time1] [time2] goto [label]
IFTIME<= [time1] [time2] goto [label]
IFTIME>= [time1] [time2] goto [label]
IFTIME! = [time1] [time2] goto [label]

Arguments: [time1] [Variable](#) or [string](#) defining a time in the format of **hh.mm.ss** or **hh.mm**.
[time2] [Variable](#) or [string](#) defining a time in the format of **hh.mm.ss** or **hh.mm**.
[label] A valid [label](#) within the current script file which is branched to if the time condition is satisfied.

Options: none

Form 2

Syntax: IFTIMExx [time] goto [label]
Forms: IFTIME= [time] goto [label]
IFTIME< [time] goto [label]
IFTIME> [time] goto [label]
IFTIME<= [time] goto [label]
IFTIME>= [time] goto [label]
IFTIME! = [time] goto [label]

Arguments: [time] [Variable](#) or [string](#) defining a time in the format of **hh.mm.ss** or **hh.mm** to compare against the time stamp of the file obtained with the most recent GETNEXTFILE script command.
[label] A valid [label](#) within the current script file which is branched to if the time condition is satisfied.

Options: none

Form 1 of this command is used to compare two specified time strings in format of **hh.mm.ss** or **hh.mm**.

Form 2 of this command is used in conjunction with the *most recent* GETNEXTFILE to compare the time of the file with a specified time string.

The time must be expressed in military time (hours 00 through 23), and leading zeroes are required. When times are compared, seconds are always ignored. The following are examples of valid time strings.

```
01.30.00    1:30AM
17.30      5:30PM
12.15      12:15PM
```

Example of Form 1

The following conditional branch is taken if the current time is later than 11AM.

```
IFTIME> %time "11.00.00" goto later_time
```

Example of Form 2

The following conditional branch is taken if a local file just obtained with the GETNEXTFILE command has a time stamp is older than 5:30PM.

```
GETNEXTFILE "*.txt"  
IFTIME< "17.30" goto old_file
```

Syntactically, no space is permitted to the left of the '!', '=', '<' or '>' symbols, and a space is required to the right of these symbols.

Related Command(s): [GETNEXTFILE](#), [IFDATE](#)

IFYES -- Conditional branch if 'Yes' is clicked in ASK dialog box

Syntax: IFYES goto [label]
Arguments: [label] A valid [label](#) within the current script file.
Options: none

[This command not supported when running in a minimized window or as an NT Service.](#)

This script command is used in conjunction with the ASK command. If you click the 'Yes' button in the ASK dialog box, this command branches to [label].

For example:

```
ASK "Yes or No?"  
IFYES goto answer_is_yes
```

Related Command(s): [ASK](#), [IFNO](#),

IMPORT -- Import Configuration Settings

Syntax: IMPORT [file name]
Arguments: [file name] [Variable](#) or [string](#) defining a file or path name; if no path is defined FileLink's working folder is used.
Options: none

This script command imports a set of configuration settings from the specified file and stores them as the current configuration in the Registry. The import operation may be used to restore a previous set of configuration settings, or install settings distributed to multiple FileLink sites.

Settings may also be imported manually using the Import Settings command under the **File** menu or by using the -g command line switch.

Settings are exported using the EXPORT command or via Export Settings command under the **File** menu.

Related command(s): [EXPORT](#)

INC -- Increment a variable by one

Syntax: INC [variable]
Arguments: [variable] A [variable](#) containing from one to six numeric characters.
Options: none

This script command is used to increment a variable by one. If each character in the variable is not numeric (e.g., digits 0 - 9), then the command fails.

Numeric strings used in the INC (and [DEC](#)) command are assumed to be a fixed length of one to six characters and contain leading zeros. When the extent of a variable is exceeded, the value wraps (i.e., a two character string is greater than **99** after adding one then the value wraps to **00**; **9** wraps to **0**; **999** wraps to **000**; etc.)

Caution

This command is primarily intended to provide a mechanism for sequentially naming files, not as a simple numeric function. So its behavior of wrapping from **99** to **00**, for example, may not be appropriate when doing simple arithmetic. The INC command may be used for both purposes but you need to remain aware of the command's behavior.

If the variable is not previously assigned, the variable is created and is set equal to **000**.

Consider the example below where a variable is used as a loop counter. In this case, the leading zero is required even as a loop counter in order for `counter` to become greater than 9.

```
SETNUM counter = 01
;; loop 20 times
:loop
DISPLAY counter
INC counter
IFNUM< counter 20 goto loop
```

Related Command(s): [DEC](#), [IFNUM](#), [SET](#), [SETNUM](#),

LINEIN -- Read one or more characters from COM port

Syntax:	LINEIN	[variable] [/options]
Arguments:	[variable]	A variable to store characters read from the port; if the variable does not previously exist, it is created.
Options:	/allowall	Do not strip unprintable characters; these characters are replaced with the configured LINEIN fill character.
	/flush	Flush the receive buffer before starting read.
	/length=xx	Maximum number of characters to receive; if not specified the maximum of 1020 characters is used; if 0, and either a terminating character or terminating sequence is specified, then LINEIN accepts an unlimited number of characters until the termination condition is matched; all of the characters except the terminating sequence itself are discarded.
	/termchr=lf	Terminate LINEIN when a line-feed is received; this changes the default of a carriage-return terminating character.
	/termchr=none	Do not use either a line-feed or carriage-return as a terminating character.
	/termseq="xxx"	Terminate LINEIN when the specified termination sequence is received.
	/timeout=nn	Time-out in seconds to wait for a character to be received; the default time-out is 30 seconds.

This script command receives characters from an open COM port and saves the characters in a string variable. The various options control how many characters are received.

This command is generally used to accept printable characters from the COM port. By default FileLink removes any unprintable characters before they are saved in the specified variable. If the loss of unprintable character alters the resulting string in an undesirable way by altering character position, for example, you may use the **/allowall** option. When this option is used, the relative character position of the string is preserved by replacing the unprintable characters with the configured LINEIN fill character. The fill character defaults to a space.

By default, LINEIN terminates when a carriage-return character is received. You may alter this behavior by using either the **/termchr=lf** or **/termchr=none** options. LINEIN also terminates if the maximum number of characters has been received, when a user-defined string pattern is detected, or a time-out expires.

If **/termchr=none** is specified and **/termseq** is not, LINEIN terminates after a fixed number of characters has been received - either the number specified by the **/length** option or the default value of 1020 characters.

Consider the following examples.

```
;; read until exactly 10 characters have been received
LINEIN comdata /termchr=none /length=10

;; read until a line-feed is received or 10 seconds elapses
LINEIN comdata /termchr=lf /timeout=10
```

```
;; read until string "/end" is received or end of line
LINEIN comdata /termseq="/end"

;; read until a line-feed or semi-colon is received
LINEIN comdata /termchr=lf /termseq=";"
```

The following is a more specific example showing how the LINEIN command may be used to scan incoming characters for a specific prompt. For the sake of this example, let's assume that the remote system sends the following banner and user name prompt immediately after a connection is established.

```
Welcome to Bozo World
Home of the Funniest Clown on Earth
To chat with Bozo, please sign in

Username:
```

One way to handle this character sequence would be to use four LINEIN statements in the FileLink script file. This is fine when you can be certain that the prompt is always to be found in the fourth line. However, using a single LINEIN command, as shown below, is simpler and more dependable since it is not dependent on knowing how many lines there will be before the prompt appears.

```
LINEIN bozo /termchr=none /termseq="Username:" /timeout=30
```

If you are unsure of the number of characters that may be received before the prompt, or if this number may be greater than 1020, add the `/length=0` option to the command. This results in FileLink discarding all the received characters *except* for the terminating sequence. It is strongly recommended that you use the `/timeout` option in this case to prevent the LINEIN command from hanging in any case where the terminating sequence is not received.

```
LINEIN bozo /termchr=none /termseq="Username:" /length=0 /
timeout=30
```

Related Command(s): [FLUSH](#), [LINEOUT](#), [READFILE](#), [WRITEFILE](#)

LINEOUT -- Write one or more characters to COM port

Syntax:	LINEOUT	[msg] [/options]
Alt Syntax:	SENDCMD	[msg] [/options]
Arguments:	[msg]	Variable or string defining the characters to write to the COM port
Options:	/binchr=xx	Specify a binary (decimal) character, usually unprintable, to send; not permitted with SENDCMD syntax
	/flush	Flush the receive buffer before writing characters to the Port and then flush the receive buffer of characters echoed back after the write completes
	/termchr=lf	Terminate the string by writing a line-feed; this changes the default of a carriage-return terminating character; <i>not</i> permitted with SENDCMD syntax
	/termchr=none	Do not write either a line-feed or carriage-return as a terminating character; <i>not</i> permitted with SENDCMD syntax
	/timeout=xx	Time-out in seconds to wait for the receive buffer to be flushed (if /flush is specified) and/or for a character to be written; the default time-out is 30 seconds

This script command writes characters to an open COM port. By default LINEOUT terminates a string by writing a carriage-return character. You may alter this behavior by using either the **/termchr=lf** or **/termchr=none** options when using the LINEOUT syntax.

The **/binchr=xx** option may be used to send an unprintable (binary) character either appended to the end of the [msg] string or by itself. The **xx** must be a decimal value between 0 and 255. The binary character is sandwiched between the end of [msg] string (if present) and before the terminating character. To send the binary character by itself, the [msg] string should be empty and **/termchr=none** is specified.

The **/flush** option performs two functions. First, it flushes the receive buffer of spurious characters that may already be present *before* the write begins. Also, it flushes any characters from the receive buffer that may be echoed back during the write *after* the write completes successfully. Under certain conditions, you may want to enable this option to insure that a subsequent LINEIN command reads any response that may be received from the remote system rather than spurious or echoed characters to what was just written.

Consider the following examples.

```
;; write a carriage-return terminated string
LINEOUT "output string"
;; write a line-feed terminated string
SET string = "write this string"
LINEOUT string /termchr=lf /timeout=10
;; send Ctrl-C (hex 0x03) to the remote system
LINEOUT "" /termchr=none /binchr=3
```

This command is also used when sending script commands to be executed by FileLink running on a remote system. In this case you may want to use the alternate SENDCMD syntax

to make your script files more readable. Consider the following example.

```
;; request a specific file be sent from remote system
SEND CMD "SEND FILE 'specificfile'"
;; this command could also be written a's
SEND CMD 'SEND FILE "specificfile"'
```

Note

You must use single quotation marks when embedding a script command within an alpha-numeric string argument bracketed by double quotation marks and vice versa.

```
;; either of the following cause an invalid command error
SEND CMD "SEND FILE "specific_file""
SEND CMD 'SEND FILE 'specific_file''
```

Related Command(s): [FLUSH](#), [LINEIN](#), [READFILE](#), [WRITEFILE](#), [SEND CMD](#), [REMOTECMD](#)

LISTDIR -- List local directory to a file

Syntax: LISTDIR [dir name] [file name]
Arguments: [dir name] Optional [variable](#) or [string](#) defining a folder path name to list; if no path is defined FileLink's working folder is used.
 [file name] Optional [variable](#) or string defining a file name to write the folder listing to; if no path is defined FileLink writes to a file named "dirlist.txt" in the working folder.

This script command produces a folder listing of FileLink's working folder or a specified folder and writes it to a file. If [file name] is not specified, FileLink creates a file in the working folder named "dirlist.txt". The folder format written to this file is the same as might be produced using the **dir** command in a DOS window.

This command is useful in creating a folder listing file. Consider the following example where a listing file of a folder is created and sent to the remote system.

```
LISTDIR "\uploads\*.*"  
SENDFILE "dirlist.txt"
```

Related Command(s): [MAKEDIR](#), [WORKINGDIR](#)

LOG -- Control the script log file

Syntax:	LOG	[file name] [/options]
Arguments:	[file name]	Optional variable or string defining a file or path name; if no path is defined FileLink's working folder is used.
Options:	/append	Specify that log data is to be appended to preexisting [file name] (if any); if the file does not exist, it will be created.
	/maxsize=xx	Specify the maximum size of the log file (in Kilobytes).
	/new	Specify that a new log file name is to be created (based on the current date and time) when this command is executed and whenever an existing log file reaches the maximum size (if a size has been specified).
	/off	Turn script file logging off.
	/on	Turn script file logging on (assuming that [file name] has been previously defined on an earlier call to LOG or a log file has been previously defined in the FileLink configurator).

If the [file name] argument is present, this script command creates a new script log file by this name. It is also implied that script logging is to be turned on. The script log file records all commands and status messages that occur during a file transfer session. This file is useful to determine if an unattended session was successful.

The **/new** option instructs FileLink to create a new log file name using the current date and time. Such a file will be created when the command is executed and, if the **/maxsize** option is also specified, whenever the log file exceeds this maximum size. The [file name] argument must be present, but it can be an empty string. FileLink takes the base file name (i.e., the part of the file name before any extension) and appends the current date and time in the fashion shown below. Notice that if no extension is originally specified, FileLink appends **.log** to the final file name.

```
LOG "mylog" /new
// creates log file = mylog Wed Oct 30 15.38.43 2002.log

LOG "mylog.txt" /new
// creates log file = mylog Wed Oct 30 15.38.43 2002.txt

LOG "mylog.xx.log" /new
// creates log file = mylog Wed Oct 30 15.38.43 2002.xx.log

LOG "" /new
// creates log file = Wed Oct 30 15.38.43 2002.log
```

When the **/new** option is used the name of any newly created log file is available in the **%currentlogfile** script variable.

The **/maxsize** option limits the maximum size that a log file can grow to. The size is specified in kilobytes. When the maximum size is reached, FileLink handles this condition in one of two ways. If **/new** is also specified, the current log file is simply closed and a new file is created using the convention described above. If **/new** is not present, FileLink toggles between two

files. When the first file is full, it is closed and a second created and written to. When the second file is full, it is closed and the first file is reopened, cleared, and logging continues. This alternating between files continues until FileLink terminates. When **/maxsize** is present, [file name] is altered as shown below. Notice that if no extension is originally specified, FileLink appends **.log** to the final file name.

```
LOG "mylog" /maxsize=100
// creates log file = mylog_1.log
// this alternates with a file to be named mylog_2.log

LOG "mylog.txt" / maxsize=100
// creates log file = mylog_1.txt
// this alternates with a file to be named mylog_2.txt
```

The **/append** option instructs FileLink to append new log data to a previously existing file specified by [file name]. If [file name] does not exist, it will be created. The **/append** option may be combined with **/maxsize** but may not be used with the **/new** option.

If [file name] is omitted, the **/on** and **/off** options control logging to a previously defined log file. When logging is turned on, new log messages are appended to the existing log file. For example:

```
LOG /off
```

Related Command(s): [LOGMSG](#), [TRACELOG](#), [WORKINGDIR](#)

LOGMSG -- Write a message to the script log file

Syntax: LOGMSG [message]
Arguments: [message] [Variable](#) or [string](#) containing a message to write to the script log file.
Options: none

This script command writes a message to the script log file. Messages written in this manner are highlighted (as are error messages) with a preceding arrow indicator to make them easier to locate when scanning a log file.

Consider the following example.

```
LOGMSG "What a good thing!"
```

This command would produce an entry in the log file similar to the following.

```
Thu Oct 04 11:41:09 -- Line 5:      logmsg "What a good thing!"  
Thu Oct 04 11:41:18              => What a good thing!
```

Related Command(s): [LOG](#)

LOGNTEVENT -- Write a message to the NT application event log

Syntax: LOGNTEVENT [message] [options]
Arguments: [message] [Variable](#) or [string](#) containing a message to write to the NT application event log.
Options: /type=xx Specifies the type of event being logged. If omitted, the default value if 1 is used.

This script command writes a message to the NT application event log. This command is not functional when running FileLink with Windows 98.

The **/type** option allow the event type to be specified. The permitted event types are listed below.

- 1 - Error event
- 2 - Warning event
- 3 - Information event
- 4 - Success Audit event
- 5 - Failure Audit event

Consider the following example.

```
LOGNTEVENT "This goes to the event log" /type=3
```

Related Command(s): [LOG](#)

LOOPIF -- Conditional branch used in conjunction with LOOPCOUNT

Syntax:	LOOPIF	goto [label1] else goto [label2]
Arguments:	[label1]	A valid label within the current script file which is branched to if the conditional test fails.
	[label2]	A valid label within the current script file which is branched to if the conditional test is successful.
Options:	none	

This script command checks the result of the previous command and performs a conditional branch to [label1] if the command failed (the result code is non-zero). The branch to [label2] is taken if no error was encountered.

If [LOOPCOUNT](#) is non-zero, the error path is taken LOOPCOUNT times. If the error condition persists after taking the error path the specified number of times, then the command immediately after the LOOPIF is executed. If LOOPCOUNT is zero, the error path is taken indefinitely.

Consider the following where the dial command is repeated up to three times or until it is successful (whichever comes first).

```
LOOPCOUNT 3
:retry_dial
  DIAL "1-555-1212"
LOOPIF goto retry_dial else goto connected
;; Dialing failed
EXIT
:connected
```

Related Command(s): [LOOPTO](#), [GOTO](#), [LOOPCOUNT](#)

LOOPTO -- Unconditional branch used in conjunction with LOOPCOUNT

Syntax: LOOPTO [label]
Arguments: [label] A valid [label](#) within the current script file which is branched to if
 a specified number of loops has not been attained.
Options: none

This script command directions execution back to [label] and is used in conjunction with [LOOPCOUNT](#) to execute a command or sequence of commands more than one time.

Consider the following example in which the message **Wow!** is transmitted three times.

```
LOOPCOUNT 3  
:many_tries  
LINEOUT "Wow! "  
LOOPTO many_tries
```

Related Command(s): [LOOPCOUNT](#), [LOOPIF](#), [GOTO](#), [IFERROR](#)

LOOPCOUNT -- Set maximum loop repetition

Syntax: LOOPCOUNT [count]
Arguments: [count] Numeric count value
Options: none

This script command sets the number of times a command sequence is repeated. A command sequence is defined with the [LOOPTO](#) and [LOOPIF](#) commands.

Consider the following example in which the message **Wow!** is transmitted three times.

```
LOOPCOUNT 3
:many_tries
LINEOUT "Wow! "
LOOPTO many_tries
```

Related Command(s): [LOOPTO](#), [LOOPIF](#), [GOTO](#), [IFERROR](#)

MAILTO -- Send an e-mail message via default e-mail client

Syntax:	MAILTO	[to name] [subject] [body]
Arguments:	[to name]	Variable or string defining the e-mail address of recipient; to enter the destination e-mail address within your e-mail client, specify an empty string or issue MAILTO with no arguments
	[subject]	Variable or string defining the subject line for the message; to enter a subject line within your e-mail client, specify an empty string or issue MAILTO with only the [to name] argument
	[body]	Variable or string defining the body of the message; to enter the body within your e-mail client, specify an empty string or issue MAILTO with only the [to name] and [subject] arguments

This command may be used obtain an e-mail address, subject line, and/or message body under script control and then invoke the external system default e-mail client. The client is launched and FileLink script continues.

The arguments are optional but if used must be in order shown above. Supply empty string(s) as necessary if you wish to supply a subject, for example, without an e-mail address.

Line breaks may be embedded in [body] with the '\n' character sequence. Not all e-mail clients, however, recognize line breaks and may substitute a space character.

Consider the following example where the default e-mail client is launched with an e-mail address, subject line, and message body.

```
SET email = "sales@acme-widget.com"
SET subj = "Thanks for your order!"
SET body = "We appreciate your business."
MAILTO email subj body
```

Consider the following example where only a multi-line message body is specified.

```
SET body = "Your password is:\n\n4ikk3433"
MAILTO " " " " body
```

Related Command(s): [CREATEMAIL](#), [GETMAIL](#), [SENDMAIL](#)

MAKEDIR -- Create a new local folder

Syntax: MAKEDIR [folder name]
Alt Syntax: NEWDIR [folder name]
Arguments: [folder name] [Variable](#) or string defining the folder name to create.
Options: none

This script command creates a new local folder (also referred to as a directory). Consider the following example in which a folder is created on the E: drive.

```
MAKEDIR "e:\newbie"
```

Folders may be created only one level at a time. For example, if you wanted to create a new subfolder under a new folder named `e:\newbie`, it would require two MAKEDIR commands as shown below:

```
MAKEDIR "e:\newbie\subfolder"       ;;WRONG!  
;; do it this way:  
MAKEDIR "e:\newbie"  
MAKEDIR "e:\newbie\subfolder"
```

Related Command(s): [CHGDIR](#), [DELDIR](#), [WORKINGDIR](#)

MAKEFILENAME -- Create a unique, non-existent file name

Syntax:	MAKEFILENAME	[file name] [ext] [dir] [prefix]
Arguments:	[file name]	A variable where the new file name is saved.
	[ext]	A variable or string defining the file name extension (if [ext], [dir], and [base] options are omitted, .tmp is used as the default extension).
	[dir]	A variable or string defining the target folder for the file (if [dir] and [base] are omitted, the current working folder is used).
	[prefix]	A variable or string defining the desired file name prefix; a four digit number will be appended to the prefix to make it unique (if omitted fl is used).
Options:	none	

This script command creates a unique name for a non-existent file and saves it in a specified script variable. This variable may be used in any FileLink command that needs a uniquely named file. The file is not created until it is used in a command.

It is not required to specify [ext], [dir], or [base], however the arguments are position dependent, so [base] requires both [ext] and [dir] to be present, and [dir] requires [ext].

The file name returned by this command will not exist at the time the command is run, but there is no guarantee that a file of this name will not be created by some other program, so choose a naming combination that is unlikely to be used by any other program.

Consider the following example in which only the [file name] variable is present. The file name created would resemble **c:\program files\FileLink\flXXXX.tmp**, where XXXX would be 0001, 0002, etc. depending on whether there is a previously existing file or not.

```
;; create a unique file name in current working folder
MAKEFILENAME newfile
```

Below is an example where all four arguments are present.

```
;; create a file named "c:\temp\mailXXXX.eml"
MAKEFILENAME newfile "eml" "c:\temp" "mail"
```

Related Command(s): [WORKINGDIR](#)

MESSAGEBOX -- Display text in message box

Syntax:	MESSAGEBOX	[message] [title] [options]
Arguments:	[message]	Variable or string defining a text message to display within a pop-up dialog box (up to 1000 characters).
	[title]	Optional variable or string defining the window title displayed in the dialog box.
Options:	/large	Select this option to display the dialog box in a larger 12 point font rather than the default 8 point.
	/local	By default, FileLink displays a message box in the center of the screen. Use this option if you wish the box to be centered relative to the FileLink window instead.
	/nocrlf	Ignore embedded <code>\n</code> and/or <code>\r</code> carriage control.

This command not allowed when running as an NT Service or in a locked minimized window.

This script command displays a dialog box on your display. The window title and text within the dialog are specified in the command. Control returns to next script command when you close the dialog by clicking on the OK button. This command is useful to display important messages while a script is running.

If FileLink is running a script in a unlocked minimized window then FileLink's window will be restored when this command is performed.

Consider the following example.

```
MESSAGEBOX "This is a whiz-bang message box!"
```



Two embedded formatting or carriage control character sequences are recognized. A `\n` sequence is interpreted as a carriage return and a `\r` sequence is interpreted as a line feed. Use of this carriage control sequences permit you to display multiple lines inside a dialog box. For example:

```
MESSAGEBOX " Line 1 \n\r Line 2 \n\r Line 3 "
```

Use of the option `/nocrlf` suppresses the recognition of the `\n` and `\r` sequences. This is useful if you are displaying file names in the message box that may include either of these two sequences. For example:

```
MESSAGEBOX "Sending c:\newfile\reports.dat" /nocrlf
```

The [message] can be quite large -- up to 1000 characters. When using extremely long messages, we suggest that you precede the command with an [@_modifier](#) to suppress the echoing of the command to the console window and log file to preserve readability. Use care also not to overflow the possible space in the Windows dialog box that this script command will display by including too many embedded carriage control sequences.

The **/large** and **/local** options may not be used together.

Related Command(s): [PROMPT](#), [ASK](#)

MINIMIZE -- Minimize FileLink window

Syntax: MINIMIZE [options]
Arguments: none
Options: /hide Hide the window so that it doesn't appear on the taskbar.
 /lock Lock FileLink minimized; its icon appears on the taskbar but
 the window cannot be restored or maximized.

This command not supported when already running in a minimized window or as an NT Service.

This script command minimizes the FileLink window to an icon.

Use of the **/lock** options minimizes the window such that it cannot be restored or maximized so be sure that this is what you want to do since it cannot be undone.

Related Command(s): [RESTORE](#)

MODEMCMD -- Send AT command string to modem

Syntax: MODEMCMD [cmd]
Arguments: [cmd] [Variable](#) or [string](#) defining the AT command to be sent to the modem
Options: /enableresp Enable modem response; required if MODEMRESP script command is going to be used to read response to the command

This script command sends a user specified command to a Hayes compatible modem. The link cannot be connected when issuing this command.

Consider the following example.

```
MODEMCMD "ATW2"
```

Related Command(s): [MODEMRESP](#), [MODEMDEFAULTS](#), [MODEMRESET](#), [DIAL](#), [ANSWER](#)

MODEMDEFAULTS -- Set modem to factory defaults

Syntax: MODEMDEFAULTS
Arguments: None
Options: None

This script command sends an **AT&F&W** command to restore a Hayes compatible modem to its factory default settings which is the recommended mode for any modem used with FileLink. The link cannot be connected when issuing this command.

If you are using a U.S. Robotics Courier V.Everything, use the following command instead to set factory defaults.

```
MODEMCMD "AT&F1&W"
```

Related Command(s): [MODEMRESET](#), [MODEMCMD](#), [MODEMRESP](#), [DIAL](#), [ANSWER](#)

MODEMDETECT -- Locate first available modem and/or COM port

Syntax: MODEMDETECT

Arguments: None

Options: /firstportok Report success even if modem is not found; effectively finds first COM port in system if no modem is present

This script command scans the system for the first available COM port and/or attached modem. The result is saved in a script variable named **%newport** in the form "COMx" where x is a numeric value between 1 and 48.

When MODEMDETECT is specified without an option, it attempts to find the first responsive modem attached to one of the COM ports in your PC. If found, the port identifier (e.g., COM2) is stored in the [%newport](#) variable. If no modem is found, the command returns an error and **%newport** is set to a null string.

If the **/firstportok** option is included, the command returns still returns the first COM port where a modem is found. However, it behaves differently if no modem is found by returning the first COM port found in the system regardless whether a modem is attached or not. If no modem nor COM port is found, the command returns an error and **%newport** is set to a null string.

The following example shows how to set FileLink to use a COM port where a modem has been detected.

```
MODEMDETECT
IFERROR $ERROR_NO_MODEMS_DETECTED goto no modems
USEPORT %newport
```

Related Command(s): [MODEMRESET](#), [MODEMCMD](#), [MODEMRESP](#)

MODEMRESET -- Send reset to modem

Syntax: MODEMRESET
Arguments: None
Options: None

This script command sends an ATZ command to reset a Hayes compatible modem unless you have configured a modem reset string. If a reset string is defined, it is sent instead of the ATZ command. If the link is connected at the time of issuing this command, the link is dropped and then the modem is reset.

Related Command(s): [MODEMDEFAULTS](#), [MODEMCMD](#), [MODEMRESP](#), [DIAL](#), [ANSWER](#)

MODEMRESP -- Read modem response

Syntax:	MODEMRESP	[variable] [/options]
Arguments:	[variable]	A variable to store characters read from the modem; if the variable does not previously exist, it is created
Options:	/flush	Flush the receive buffer before starting read
	/length=xx	Maximum number of characters to receive; if not specified the maximum of 1020 characters is used
	/timeout=nn	Time-out in seconds to wait for a character to be received; the default time-out is 3 seconds

This script command reads the response from a command sent to a Hayes compatible modem with the MODEMCMD script command. It is not required to read the response from the modem when a command is sent, but you may optionally use this command to confirm that a modem is present or is properly responding to commands. The **/enableresp** option is required on the MODEMCMD command prior to calling MODEMRESP.

In the following example, the variable 'resp' will contain the response from the modem. The response is likely to be **AT** which will be the modem echoing back the command.

```
MODEMCMD "AT" /enableresp
MODEMRESP resp
```

Typically your script should continue call MODEMRESP until the three-second time-out expires. This way you will be sure to get all of the characters of the response. So, your script might look like the following.

```
; send AT command to modem
MODEMCMD "AT" /enableresp
:resp_loop
MODEMRESP resp
IFERROR= $ERROR_OPTIMEDOUT GOTO end_of_resp
; do something meaningful here with the responses...
; ...
GOTO resp_loop
: end_of_resp
```

Related Command(s): [MODEMCMD](#), [DIAL](#), [ANSWER](#)

MOVE -- Move one local file to another location

Syntax: MOVE [src file] [dest file]
Arguments: [src name] [Variable](#) or [string](#) defining a file or path name; if no path is defined FileLink's working folder is used.
 [dest name] [Variable](#) or [string](#) defining a file or path name; if no path is defined FileLink's working folder is used.
Options: none

This script command to moves (copies and deletes) the source file to the destination location.

Full file or path names are required. For example, the following is a valid command.

```
MOVE "c:\test\file" "c:\test2\file"
```

The following is an invalid command in the same environment.

```
MOVE "c:\test\file" "c:\test2"
```

Related Command(s): [APPEND](#), [COPY](#), [DELETE](#), [RENAME](#), [WORKINGDIR](#)

NATO -- Specify a no activity time-out

Syntax: NATO [timeout]
Arguments: [timeout] [Variable](#), [string](#), or numeric constant defining a numeric time-out value.
Options: none

This script command specifies a value, in seconds, used as for a no activity time-out during file transmission and reception. The no activity time-out is a failsafe time-out that permits a script file to recover if a file transfer fails abnormally yet the connection remains established.

This time-out is active *only after a file transfer has begun*. The **/timeout** option on the [SENDFILE](#) and [RCVFILE](#) commands has priority until the first byte of data actually is sent or received.

The no activity time-out is disabled (i.e., set to 0) by default.

Should a no activity time-out occur, it probably indicates a serious error with the connection. In most cases, your script should issue a [DISCONNECT](#) and then reconnect or log back on before attempting another file transfer. In some cases, you may find it necessary to attempt a reconnection multiple times before it will be successful after a no activity time-out has occurred.

Consider the following example in which the no activity time-out is set to 30 seconds.

```
NATO "30"  
SENDFILE "testfile"  
IFERROR= $ERROR_NO_ACTIVITY_TIMEOUT goto disconnect
```

PAUSE -- Pause for specified length of time or until specified hour:minute

Syntax: PAUSE [options]
Arguments: none
Options: /for=xx Delay execution for xx seconds.
 /until=hh:mm Delay execution until this time-of-day (expressed in military time).

This script command suspends the execution of a script file for a specified number of seconds or until a specific time-of-day. More extensive scheduling capabilities are offered with the CRON script command.

Consider the following example in which script execution is delayed until 5:00PM.

```
PAUSE /until=17:00
```

Related Commands: [CRON](#)

PERFORM -- Execute script command contained in character string or variable

Syntax: PERFORM [command]
Arguments: [command] [Variable](#) or [string](#) defining a valid FileLink script command to be executed immediately.
Options: none

This script command takes a variable or string argument and executes it if it was an inline script command. This command is useful to permit script commands to be passed in as a Shortcut Target argument when FileLink is launched or obtained via the PROMPT script command.

Consider the following example of a script file that accepts commands that you type when prompted by FileLink. You would type the STOP or FLUSH commands to end script execution.

```
:loop  
PROMPT var "Enter Command" "FileLink Prompt Window"  
PERFORM var  
GOTO loop
```

Consider the following example where a command is passed an argument from the Shortcut Target command line to set the file transfer protocol to Zmodem.

```
"c:\FileLink\Filelink.exe" &protocol "zmodem"&  
  
PERFORM &1
```

Related Commands: [EXEC](#), [PROMPT](#)

See also [Using Shortcut Target Arguments in Script Files](#)

PGPCOMMAND -- Send a "raw" GnuPG command

Syntax: PGPCOMMAND [command] [log file]
Arguments: [command] A variable or [string](#) defining the command to pass to the GnuPG executable "gpg.exe"
[log file] A variable or string defining the log file where commands and responses from GnuPG (gpg.exe) are recorded when the specified command is executed. This log file may be used for troubleshooting; it may also be parsed by other script commands (e.g., [READFILE](#)) to identify GnuPG results not directly supported by FileLink.

NOTE: THIS COMMAND IS PROVIDED FOR CONVENIENCE TO EXPERIENCED GNUPG USERS ONLY. WE DO NOT OFFER SUPPORT FOR GNUPG COMMANDS BEYOND WHAT FILELINK DIRECTLY PROVIDES VIA THE PGPDECRYPT AND PGPENCRYPT SCRIPT COMMANDS. [USE THE PGPCOMMAND AT YOUR OWN RISK.](#)

This script command allows "raw" GnuPG commands to be passed to "gpg.exe" (the PGP component of FileLink) that are not directly supported by FileLink.

For example, to list the keys on your keyring, the command would be:

```
PGPCOMMAND "--list-keys" "gnupg.log"
```

To list the fingerprints, the command would be:

```
PGPCOMMAND "--fingerprint" "gnupg.log"
```

Be sure to always precede each GnuPG option with two dashes.

The results from PGPCOMMAND are written to the specified log file which is created in the FileLink working folder (unless a fully qualified file name is provided). If you want to have access to this information within your script, write script code using the [READFILE](#) script command to parse this file.

For example, here is the sample output to the log file using "--fingerprint" :

```
040421124415076 :: ***** : END OF COMMAND
040421124508516 :: ***** : START DirectCommand
040421124508516 :: ***** : START INPUT
040421124508516 :: in : "C:\PROGRA~1\ROBO-F~1.0BE\gpg" --
fingerprint > "C:\PROGRA~1\ROBO-F~1.0BE\Logs\command.log"
040421124508516 :: ***** : END OF INPUT
040421124508576 :: ***** : Results of evaluation = 0
040421124508576 :: ***** : Results
040421124508576 :: out : c:/gnupg\pubbring.gpg
-----
pub 1024D/5F9786A6 2003-05-01 Juan Valdez <juanv@columbiacoffee.
org>
```

```
Key fingerprint = 2B96 4426 8FC0 8FFA D6A9 0412 91D3 8355
5F97 86A6
sub 2048g/5416ED79 2003-05-01
```

Using the READFILE command you can read the “gnupg.log” file down to the line containing the fingerprint, use the [SETRIGHT](#) command to extract the fingerprint into a script variable if need be.

Related Command(s): [PGPDECRYPT](#), [PGPENCRYPT](#), [PGPIMPORT](#)

PGPDECRYPT -- Decrypt a PGP encrypted file

Syntax:	PGPDECRYPT	[encrypt file] [target file] [keyring] [options]
Arguments:	[encrypt file]	A variable or string defining the file name of the PGP encrypted file to decrypt; this file may or may not be ASCII armored . Wildcard characters are not permitted in [encrypt file] or [target file] .
	[target file]	A variable or string defining the file name of the decrypted file; if the file exists, it will be overwritten. If an empty string is provided (i.e., "") then the decrypted file is written to the original file name saved when the file was encrypted.
	[keyring]	Optional variable or string defining the location of the PGP keyring used in the decryption; if omitted, FileLink expects the keyring file to be in the current working folder.
Options:	/pgplog=xx	Include this option if you wish to define a log file where commands and responses from GnuPG (gpg.exe) are recorded during the decryption process. This log file may be used for troubleshooting; it may also be parsed by other script commands (e.g., READFILE) to identify GnuPG results not directly supported by FileLink.
	/pgpopt=xx	Any GnuPG (gpg.exe) decryption option not implemented by FileLink and therefore must be explicitly specified (e.g., --skip-verify). This option should only be used by advanced users and/or under the direction of FileLink technical support.
	/pw=xx	The PGP key Passphrase field associated with the PGP key to be used in the decryption; if you have only one key on your keyring and you saved the Passphrase field when creating or selecting the key using the FileLink configurator, you may omit this option.

This script command decrypts a file encrypted using PGP encryption. Files may have been encrypted using the FileLink [PGPENCRYPT](#) script command or any other PGP or GPG encryption application.

To decrypt a file, you must have an existing keyring containing your private key and have imported the public key of the originator of the file. You must specify your passphrase either within the command itself using the **/pw** option or by previously specifying it when creating or selecting a key using the FileLink configurator.

Below is a decryption example where the passphrase has been previously defined and the keyring file exists in the FileLink working folder.

```
PGPDECRYPT "encrypted.txt.gpg" "decrypted.txt"
```

You can also write the decrypted file to a different folder as shown below.

```
PGPDECRYPT "encrypted.txt.gpg" "c:\new\decrypted.txt"
```

If you haven't saved the passphrase when creating the keyring then you will need to provide the passphrase on the PGPDECRYPT command as shown below.

```
PGPDECRYPT "encrypted.txt.gpg" "decrypted.txt" /pw="my
passphrase"
```

Important

FileLink secures your passphrase by saving it in an encoded format in the Windows registry along with its other settings. The passphrase is also never displayed in the FileLink console window nor written to any log file. But be aware that *it does appear in clear-text* in a script file. Therefore, the method of specifying your passphrase during configuration is the most secure.

The original file name of an encrypted file is often embedded within the decrypted file. FileLink allows an encrypted file to be automatically set to this name *and saved in the current working folder* by using the following syntax (Note: the [target file] is an empty string).

```
PGPDECRYPT "encrypted.txt.gpg" ""
```

Important

FileLink has no way of knowing what the original embedded file name is. If you use this option, you must know in advance what the file name is if you intend to perform other actions on the resulting decrypted file from within the FileLink script environment.

For advanced troubleshooting, you may include the **/gpglog** option which results in commands and responses to and from GnuPG (gpg.exe) being written to the specified log file. (GnuPG is the underlying PGP encryption/decryption engine used by FileLink.) The following example encrypts a file and writes to a log file named "encrypt.log".

```
PGPDECRYPT "encrypted.txt.gpg" "decrypted.txt" /gpglog="encrypt.
log"
```

If a fully qualified file name is not specified, the log file will be created in the current FileLink working folder. If the file exists, it will be appended to. Delete the file before each PGPDECRYPT command if you want only a single command to be logged.

In some cases you may be required to decrypt a file using a GnuPG option that is not directly implemented by FileLink. In such cases, the **/gpgopt** option allows you to specify the necessary option(s) in the PGPDECRYPT command and have it passed to GnuPG. The following example encrypts a file and specifies an unsupported option.

```
PGPDECRYPT "encrypted.txt.gpg" "decrypted.txt" /gpgopt="--skip-
verify"
```

Multiple GnuPG options may be passed using **/gpgopt**. When doing so, separate each complete option with a semi-colon as shown below.

```
PGPDECRYPT ... /gpgopt="--skip-verify;--no-verbose"
```

Be sure to always precede each GnuPG option with two dashes.

Related Command(s): [PGPENCRIPT](#), [PGPIIMPORT](#)

PGPENCRIPT -- Encrypt a file using PGP

Syntax:	PGPENCRIPT	[src file] [target file] [keyring] [options]
Arguments:	[src file]	A variable or string defining the file name of the PGP encrypted file to encrypt. Wildcard characters are not permitted in [src file] or [target file] .
	[target file]	A variable or string defining the file name of the newly encrypted file; if the file existed, it will be overwritten.
	[keyring]	Optional variable or string defining the location of the PGP keyring used in the encryption; if omitted, FileLink expects the keyring file to be in the current working folder.
Options:	/armor	Select this option to ASCII armor the [target file].
	/pgplog=xx	Include this option if you wish to define a log file where commands and responses from GnuPG (gpg.exe) are recorded during the encryption process. This log file may be used for troubleshooting; it may also be parsed by other script commands (e.g., READFILE) to identify GnuPG results not directly supported by FileLink.
	/gpgopt=xx	Any GnuPG (gpg.exe) encryption option not implemented by FileLink and therefore must be explicitly specified (e.g., --force-v3-sigs). This option should only be used by advanced users and/or under the direction of FileLink technical support.
	/comment=xx	The PGP key Comment field on your keyring associated with the PGP key of the <i>recipient</i> of the encrypted file; this option may not be necessary if enough information is provided via the /email and/or /user options to uniquely identify the recipient's key on your public keyring. The comment specified must completely match the Comment field in the keyring.
	/email=xx	The PGP key E-mail Address field on your keyring associated with the PGP key of the <i>recipient</i> of the encrypted file; this option may not be necessary if enough information is provided via the /comment and/or /user options to uniquely identify the recipient's key on your public keyring. The e-mail address specified must completely match the E-mail Address in the keyring.
	/pw=xx	The passphrase associated with the private key on your keyring necessary when using the /sign option; if a passphrase was not saved when the key was created or when a keyring was selected using the FileLink Configurator then this option is required when the /sign option is specified so that the encrypted file can be digitally signed; otherwise the passphrase saved at configuration time is used if present.
	/sign	Select this option if you wish to digitally sign the file using your own public key.
	/textmode	Select this option if you wish to have [target file] saved in a text mode format.
	/user=xx	The PGP key User Name field on your keyring associated with the PGP key of the <i>recipient</i> of the encrypted file; this option may not be necessary if enough information is provided via the /comment and/or /email options to uniquely identify the recipient's key on your public keyring. The user name may be a full or partial match with the User Name field in the keyring if /user option is used without /

comment and **/email**. It must be a complete match if either or both of the **/comment** or **/email** options are used.

This script command encrypts a file using PGP encryption. Files may be decrypted using the FileLink [PGPDECRYPT](#) script command or any other PGP or GPG encryption application.

To encrypt a file, you must have an existing keyring containing the private key you wish to use. (If you have multiple private keys on a given keyring, FileLink will always use the first key.) You must also have the public key of the recipient of the file on the keyring and use the **/user**, **/comment**, and/or **/email** options to specify all or part of the [key ID](#) for it to be identified.

Use the **/armor** option if you wish the resulting file to be in ASCII armored format.

Use the **/sign** option if you wish the resulting file to be digitally signed.

Use the **/textmode** option if you wish the resulting file to be a text mode format.

When signing an encrypted file, you must specify your passphrase either within the command itself using the **/pw** option or by previously specifying it when creating or selecting a key using the FileLink configurator.

Important

FileLink secures your passphrase by saving it in an encoded format in the Windows registry along with its other settings. The passphrase is also never displayed in the FileLink console window nor written to any log file. But be aware that *it does appear in clear-text* in a script file. Therefore, the method of specifying your passphrase during configuration is the most secure.

It is typical for encrypted files to have an extension of **.gpg** (except where noted below). In most of the examples below, we add this extension at the end of the original file to create the name of the encrypted version of the file. We recommend that you adopt these conventions as well.

Original File Name: "datafile.txt"

Encrypted File Name: "datafile.txt.gpg"

For ASCII armored files (see below):

Original File Name: "datafile.txt"

Encrypted File Name: "datafile.txt.asc"

In the example below a file is encrypted using a public key on the default keyring that is identified by a user name of "Dick Tracy".

```
PGPENCRYPT "datafile.txt" "datafile.txt.gpg" /user="Dick Tracy"
```

In the example below, more of the key ID is specified.

```
PGPENCRIPT "datafile.txt" "datafile.txt.gpg" /user="Dick Tracy" /
email=dick@detective.com
```

The following example encrypts a file and specifies the keyring is in an alternate location.

```
PGPENCRIPT "datafile.txt" "datafile.txt.gpg" "c:\gnupg" /
user="Dick Tracy"
```

The following example encrypts a file and specifies the output format to be ASCII armored. (The **.asc** extension is typically used for these types of files.)

```
PGPENCRIPT "datafile.txt" "datafile.txt.asc" /user="Dick Tracy" /
armor
```

The following example encrypts a file and digitally signs it.

```
PGPENCRIPT "datafile.txt" "datafile.txt.gpg" /user="Dick" /sign /
pw="my passphrase"
```

Important

When encrypting a file, PGP requires the public key of the recipient so that only the recipient may decrypt the file. The recipient's public key must be present on your keyring at the time of encryption. Keys are imported to your keyring by way of the [PGPIMPORT](#) script command or by using the Manage Key functionality in the FileLink configurator.

For advanced troubleshooting, you may include the **/gpglog** option which results in commands and responses to and from GnuPG (gpg.exe) being written to the specified log file. (GnuPG is the underlying PGP encryption/decryption engine used by FileLink.) The following example encrypts a file and writes to a log file named "encrypt.log".

```
PGPENCRIPT "datafile.txt" "datafile.txt.gpg" /user="Dick" /
gpglog="encrypt.log"
```

If a fully qualified file name is not specified, the log file will be created in the current FileLink working folder. If the file exists, it will be appended to. Delete the file before each PGPENCRIPT command if you want only a single command to be logged.

In some cases you may be required to encrypt a file using a GnuPG option that is not directly implemented by FileLink. In such cases, the **/gpgopt** option allows you to specify the necessary option(s) in the PGPENCRIPT command and have it passed to GnuPG. The following example encrypts a file and specifies an unsupported option.

```
"datafile.txt" "datafile.txt.gpg" /user="Dick" /gpgopt="--force-
v3-sigs"
```

Multiple GnuPG options may be passed using **/gpgopt**. When doing so, separate each complete option with a semi-colon as shown below.

```
PGPENCRIPT ... /gpgopt="--force-v3-sigs;--no-verbose"
```

Be sure to always precede each GnuPG option with two dashes.

Related Command(s): [PGPDECRYPT](#), [PGPIMPORT](#)

PGPIMPORT -- Import a PGP key

Syntax:	PGPIMPORT	[key file] [keyring]
Arguments:	[key file]	A variable or string defining the file name of an exported key to import; if there is more than one key on the keyring FileLink will import only the first key.
	[keyring]	Optional variable or string defining the location of the PGP keyring to import into; if omitted, FileLink expects the keyring file to be in the current working folder.
Options:	None	

This script command imports a PGP key file exported from a FileLink keyring or from another PGP or GPG encryption package. This command requires that a keyring previously exist.

The following example imports a key present in FileLink's working folder.

```
PGPIMPORT "export_key.gpg"
```

The following example imports a key into a keyring present in the specified folder.

```
PGPIMPORT "export_key.gpg" "c:\gnupg"
```

Related Command(s): [PGPDECRYPT](#), [PGPENCRYPT](#)

PLAYSOUND -- Play a sound (.wav) file

Syntax: PLAYSOUND [file name]

Arguments: [file name] [Variable](#) or [string](#) defining a file or path name; if no path is defined FileLink's working folder is used.

Options: none

Play the specified sound (.wav) file. This may useful to indicate the completion of different events.

PRESSANYKEY -- Suspend script execution pending a key press

Syntax: PRESSANYKEY
Arguments: none
Options: none

Suspend script execution pending the pressing of any key on the system keyboard.

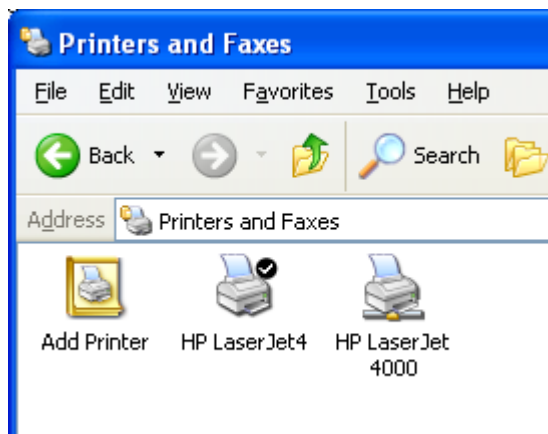
PRINT -- Print a file

Syntax: PRINT [file name] [ptr name]
Arguments: [file name] [Variable](#) or [string](#) defining a file or path name; if no path is defined FileLink's working folder is used.
 [ptr name] Variable or string defining the destination printer (e.g., HP LaserJet 4).
Options: None

This command sends the specified file to a printer. Printing files other than text files is not recommended. Script file execution resumes when the file has been completely accepted by Print Manager.

The [ptr name] used in the PRINT command is different depending on whether the target printer is locally attached or attached via a network.

To determine the [ptr name] of a local printer on Windows XP, double click on **Start | Control Panel | Printers and Faxes**. You will see a display similar to the following:



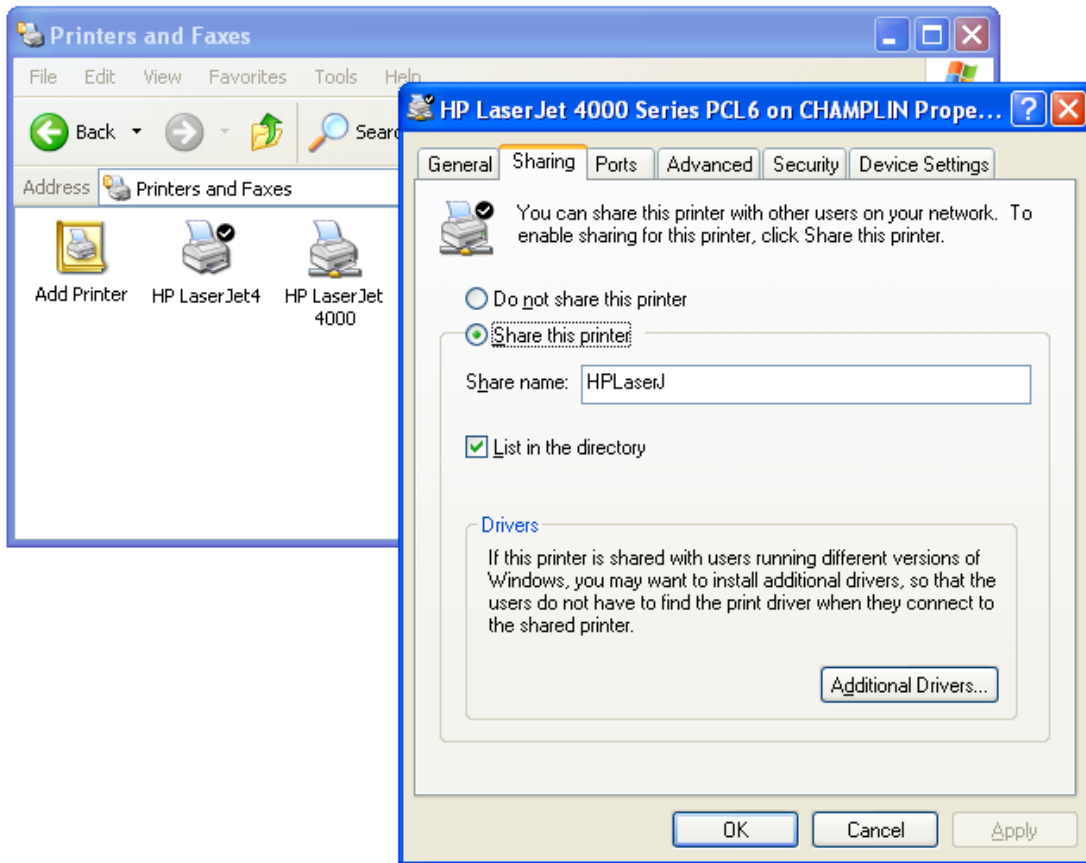
This shows two printers by the names of **HP LaserJet4** and **HP LaserJet 4000** available to your system. **HP LaserJet4** is a local printer (and the current default printer designated by the check mark) and **HP LaserJet 4000** is a shared (i.e., network) printer (designated by the "wire node" printer icon).

The PRINT command is shown below prints to the local printer:

```
PRINT "myfile" "HP LaserJet 4"
```

Determining the [ptr name] of a shared network printer is more complicated. With shared printers, the name is a combination of the hosting computer, and the printer name and is in the form **\\<hosting computer>\<printer name>**.

Under Windows XP, for example, double click on **Start | Control Panel | Printers and Faxes**, then right click on the shared **HP LaserJet 4000** printer and select **Properties**, and finally click the **Sharing** tab. You will see something like the following:



The [ptr name] to use is a combination of the computer name where the printer is physically attached (i.e., CHAMPLIN) and the name of the printer shown in the **Share Name** field. In this case the corresponding command to print to a shared printer would be:

```
PRINT "myfile" "\\CHAMPLIN\HPLaserJ"
```

Related Command(s): [WORKINGDIR](#)

PROMPT -- Display message box with title and prompt, and accept user input

Syntax:	PROMPT	[variable] [message] [title]
Arguments:	[variable]	A variable to store characters typed in the prompt dialog; if the variable does not previously exist, it is created.
	[message]	Variable or string defining a text message to display within a pop-up dialog box; this message is limited to up to three lines of text approximately 50 characters each.
	[title]	Variable or string defining the window title displayed in the dialog box.
Options:	/history=on	When this option is used the last ten responses you type are saved for instant recall. (This history is shared with console command mode.) This option is not permitted with the /password option.
	/history=off	When this option is used no history will be saved.
	/large	Select this option to display the dialog box in a larger 12 point font rather than the default 8 point.
	/local	By default, FileLink displays a message box in the center of the screen. Use this option if you wish the box to be centered relative to the FileLink window instead.
	/nocrlf	Ignore embedded \n and/or \r carriage control.
	/password	Select this option if you are prompting for a password and you do not want what is typed to be readable. This option is not permitted along with the /history option.

[This command not allowed when running as an NT Service or in a locked minimized window.](#)

This script command displays a dialog box on your display. The window title and text within the dialog are specified in the command. Control returns to next script command when you close the dialog by clicking on the OK or Cancel buttons. This command is useful to prompt for file names or other information that is not static during the course of a file transfer session.

If FileLink is running a script in a unlocked minimized window then FileLink's window will be restored when this command is performed.

The script file can detect if the Cancel button has been clicked by testing for result code 1013 or the [\\$ERROR](#) variable \$ERROR_PROMPT_CANCELLED.

Related Commands: [MESSAGEBOX](#), [ASK](#)

See also: Running FileLink With Prompting

PROTOCOL -- Specify default file transfer protocol

Syntax:	PROTOCOL	[protocol] [/options]
Arguments:	[protocol]	A variable or string defining the desired file transfer protocol for the next file send or receive operation; valid protocols are: ASCII Kermit Xmodem Xmodem1K Ymodem Zmodem
Options:	/compress=off	Turn off compression when using Kermit protocol
	/compress=on	Turns on compression when using Kermit protocol
	/no_acks	Do not send positive acknowledgements when receiving files when using the Xmodem1K or Ymodem protocols (called the G option)
	/overwrite=xx	File over-write control on remote system for files sent by FileLink using the Zmodem protocol; replace xx with one of the following: Always always over-write existing file Append append to existing file Diff over-write if file sizes/dates are different New over-write only if newer None use remote system's default Newlong over-write only if newer or longer Protect over-write only if file does not exist
	/recovery=on	Enable <i>crash recovery</i> when using the Zmodem protocol
	/recovery=off	Turns off recovery when using the Zmodem protocol
	/rxsize=xxx	Specify the default receive (Rx) buffer size used by FileLink for the selected COM port. The default value is 8192. The allowable range for this is 4096 to 32,767 bytes
	/txsize=xxx	Specify the default transmit (Tx) buffer size used by FileLink for the selected COM port. The default value is 8192. The allowable range for this is 4096 to 32,767 bytes
	/use_checksum	Use additive checksum block check algorithm when using Xmodem or Xmodem1K protocol
	/width=7	Use 7-bit data characters when using Kermit protocol
	/window=on	Turns on window mode when using the Zmodem protocol
	/window=off	Turn off sliding window mode when using the Zmodem protocol
	/windowsize=xxx	Specify the sliding window size used by FileLink for the Zmodem protocol. The default value is 4096. The allowable range for this is 512 to 16,383 bytes. In general this window size should not be changed as this may prove to be unacceptable by the remote system; the Zmodem buffer size may not exceed either the Tx or Rx buffer sizes; (in previous versions this option was named either /buffersize or /zrxsize)

This script command defines the file transfer protocol to use on subsequent file transfer operations - both send and receive. The selection made in this command overrides the default selections set with the [FileLink Configurator](#).

Consider the following example where the file transfer protocol is set to Zmodem of a script file that prompts you the name of next file to send to the remote system.

```
PROTOCOL "zmodem" /overwrite=Always /recovery=on
```

Changing the Rx, Tx, or Zmodem buffer sizes is usually not necessary. However, in some cases, especially when poor connections are a problem, file transfer reliability may benefit from the use of smaller buffer sizes. This is best determined via trial and error.

Related Commands: [USEPORT](#)

RCVFILE -- Receive one or more files

Syntax:	RCVFILE	[file name] [/options]
Arguments:	[file name]	Optional variable or string defining a file or path name; if no path is defined FileLink's working folder is used
Options:	/flush	Flush the receive buffer before starting a receive file operation.
	/fullpath	If the protocol used allows the sender to define the name of the file received, use the full path name received (not just the name of the file).
	/nostatus	Do not display send file status dialog box
	/stripf	In ASCII file transfers only, remove line-feeds from before writing data to a file
	/timeout=xx	Time-out in seconds to wait for a receive file operation to start; if xx is 0, FileLink waits indefinitely
	/zwriteblock	Do not receive data while doing file writes (Zmodem protocol only); use only if connected at high speed and transmission errors occur

This script command prepares FileLink to receive one or more files from the remote system.

The [file name] argument is required when one of the following file transfer protocols is used:

```
ASCII
Xmodem
Xmodem1K
```

In the cases of the other protocols, the [file name] argument is optional. If it is omitted, the remote system determines the name of file. The **/fullpath** option controls if the remote system also control over which directory the file is written to. If the [file name] argument is present, it overrides the name sent by the remote system.

When using a protocol that permits the remote system to supply the file name, it is possible multiple files may be sent in a single RCVFILE command. You must not override the remote file naming feature under such circumstances - namely, do not use the [file name] argument.

Consider the following example where FileLink waits one minute to receive a file from the remote system using the Zmodem protocol. Note that [file name] is not specified since the remote system specifies the local file name.

```
PROTOCOL "zmodem" /overwrite=Always /recovery=on
RCVFILE /timeout=60
```

When [file name] is not specified, the name of the last file received is saved in the **%lastfile** variable. When a wildcard is used to transfer multiple files, the number of files received is stored in the **%rcvfilecount** variable.

Related Command(s): [SENDFILE](#), [PROTOCOL](#), [LINEIN](#), [LINEOUT](#), [WORKINGDIR](#)

See also: [Using the %lastfile Variable](#)

READFILE -- Read string variable value from text file

Syntax:	READFILE	[file name] [variable] [/options]
Arguments:	[file name]	A variable or string to specify the file name to read; if no Path is defined FileLink's working folder is used.
	[variable]	A variable to store characters read from the file; if the variable does not previously exist, it is created.
Options:	/allowall	Do not strip unprintable characters; these characters are replaced with the configured LINEIN fill character.
	/length=xx	Maximum number of characters to read; if not specified the maximum of 1020 characters is used.
	/record=xx	The specific record number to read; records are delimited by carriage-return/line-feed pairs.
	/record=next	Read the next sequential record; records are delimited by carriage-return/line-feed pairs.
	/termchr=lf	Terminate the read when a line-feed is received; this changes the default of a carriage-return terminating character.
	/termchr=none	Do not use either a line-feed or carriage-return as a read terminating character; any unprintable characters read are replaced with the configured LINEIN fill character.
	/termseq=" xxx"	Terminate the read when the specified termination sequence is read.

This script command reads characters from a specified text file. This command is oriented toward reading complete records of printable characters from a file with each record terminated by a carriage-return/line-feed (CR/LF). The carriage-return/line-feed are not returned in the [variable] string.

IMPORTANT

All records in the file *including the last* must be terminated by CR/LF.

The scope of this command (and the WRITEFILE command) is not to provide full function file I/O to your script files, but rather to provide temporary storage for small amounts of information for use by a script file or an external program.

Under control of command options, reads of character strings not bounded by carriage-return/line-feeds are possible. For example, records of fixed lengths can be read by using the **/length=xx** and **/termchr=none** options.

In a READLINE command where **/termchr=none** and **/termseq=" xx"** are defined, the command returns an error if the specified terminating string is not located within **/length** number of characters (if specified) or within the default of 1020 characters. An empty string is also returned. This provides an easy way to determine if the desired terminating sequence is present or not. The record number used in association with the **/record=next** option is only incremented when there a match if found and the read completes successfully

When the **/record=xx** option is used to read beyond the first record of the file, keep in mind the terminating options in the READFILE command apply to all records. Record numbering begins at one. For example, if you are using the **/termseq=" end"** option to stop reading when the string **end** is read, all records must end with this pattern.

The **/record=next** option allows a file to be read sequentially as long as the specified file name is constant - namely, only one file can be read at a time using this option.

To *rewind* a file (or resume reading from the first record), issue the READFILE command without any arguments (returns no data), specify a different file name, omit the **/record=next** option, or specify **/record=1**.

FileLink removes any unprintable characters read before they are saved in the specified variable. If the loss of unprintable character alters the resulting string in an undesirable way by altering character position, for example, you may use the **/allowall** option. When this option is used, the relative character position of the string is preserved by replacing the unprintable characters with the configured LINEIN fill character. The fill character defaults to a space.

Consider the following examples.

```
;; "rewind" the file
READFILE

;; read the first record of a file
READFILE "datafile" first_record

;; read until a line-feed is found
READFILE "datafile" file_record /termchr=lf

;; read the third record of a file
READFILE "datafile" third_record /record=3

;; read 500 characters
READFILE "datafile" my_string /length=500 /termchr=none
```

Related Command(s): [WRITEFILE](#), [LINEIN](#), [LINEOUT](#), [WORKINGDIR](#)

REMOTECMD -- Perform a script command received via a COM port

Syntax: REMOTECMD [options]
Arguments: none
Options: /flush Flush the receive buffer before starting read.
 /timeout=*nn* Time-out in seconds to wait for a command to be received;
 the default time-out is 30 seconds.

This script command reads characters from an open COM port in exactly the same manner as the LINEIN command. Rather than saving the characters in a variable, the characters are assumed to form a valid FileLink script command. REMOTECMD then does the equivalent of a PERFORM command on the character string.

For more information on how the characters are read see the description of the LINEIN command.

Consider the following example.

```
;; accept and perform a remote command  
REMOTECMD /timeout=10
```

This is equivalent to the following command sequence:

```
LINEIN var /timeout=10  
PERFORM var
```

Related Command(s): [LINEIN](#), [LINEOUT](#), [SENDCMD](#), [PERFORM](#)

RENAME -- Rename a file

Syntax: RENAME [file name1] [file name2]
Arguments: [file name1] [Variable](#) or [string](#) defining a file or path name; if no path is defined FileLink's working folder is used.
 [file name2] Variable or string defining a file name.
Options: None

This script command renames [file name1] to [file name 2].

Related Command(s): [COPY](#), [APPEND](#), [DELETE](#), [WORKINGDIR](#)

RESTORE -- Restore minimized FileLink window to original size

Syntax: RESTORE
Arguments: none
Options: none

This command not supported when running as an NT Service.

This script command restores the FileLink window to its original size after the MINIMIZE command is executed or if FileLink was originally loaded to run minimized.

Related Command(s): [MINIMIZE](#)

RESUME -- Resume script execution from a breakpoint

Syntax: RESUME
Arguments: none
Options: none

This command is intended to be used from the console command line during debugging to resume script execution with the next statement following a BREAK command. (re)run the currently selected script file. This command has the same effect as clicking the Skip To Next Command (**Ctrl + N**) button on the toolbar.

The RESUME command is not supported as a command within a script file itself - it is only for use from the console command line during script debugging.

Related Command(s): [BREAK](#), [GO](#), [STOP](#)

See also: [Debugging Script Files](#)

RETURN -- Return from a called script file or function

Syntax: RETURN [retcode]
Arguments: [retcode] Optional [Variable](#) or numeric value specifying a function return code (valid only within a function body).
Options: None

This script command is used to exit from a called script file or from a function, and resume script execution at the next command from the point of the call.

If the RETURN command appears in a main script (i.e., a script that is not called from another script) or a chained to script (see [CALL](#) and [CHAIN](#)), then its behavior is identical to the [STOP](#) script command.

Use of the RETURN command is not always required. For example, it is not required in the following case.

```
BEGINFUNCTIONS
FUNCTION MyFunction
;; body of MyFunction
RETURN
ENDFUNCTION
ENDFUNCTIONS
```

When there are no more script commands in a function to execute, the RETURN command is assumed as shown below.

```
BEGINFUNCTIONS
FUNCTION MyFunction
;; body of MyFunction
ENDFUNCTION
ENDFUNCTIONS
```

This also applies to called script files. At the end of file of a called script file, a RETURN is assumed.

When used to return from a function, an optional parameter permits there to be a return code from the function. The returned value may be tested using any of the IFERROR script commands and is saved in the [%lasterror](#) script variable. The return code must be numeric otherwise a return code = 0 is assumed. The following is an example of testing for a function return code equal to 1.

```
FUNCTION MyFunction
;; body of MyFunction
RETURN 1

;; call the function
MyFunction
```



```
IFERROR 1 GOTO function_ok
```

In complicated called script files or functions, multiple return points may be desired. For example, the RETURN command may be used as shown in the rather sloppily written function below.

```
BEGINFUNCTIONS
FUNCTION MyFunction afile
:top
RCVFILE afile
IFERROR= $ERROR_SUCCESS GOTO success
;; return on error
RETURN
:success
MESSAGEBOX "a file received"
ASK "Receive again?" "Question"
IFYES goto top
RETURN
ENDFUNCTION
ENDFUNCTIONS
```

Related Command(s): [BEGINFUNCTIONS](#), [ENDFUNCTION](#), [ENDFUNCTIONS](#), [FUNCTION](#)

SENDCMD -- Send script command (same as LINEOUT)

Optional syntax for the [LINEOUT](#) command that may be used when sending script commands to remote system running FileLink.

Related Command(s): [REMOTECMD](#)

SENDFILE -- Send one or more files

Syntax:	SENDFILE	[file name] [/options]
Arguments:	[file name]	Variable or string defining a file or path name; if no path is defined FileLink's working folder is used; wildcard characters are permitted in the file name itself (not the path name portion) as long as the file name conforms to the 8.3 file naming convention; wildcard characters not supported when using ASCII, Xmodem, or Xmodem1K protocols
Options:	/archive	Move [file name] to the designated archive folder after a successful transmission
	/delete	Delete [file name] after it has been successfully sent; local folders are not deleted
	/flush	If the protocol used allows the sender to define the name of the file on the remote system, send the full path name (not just name of the file)
	/fullpath	If the protocol used allows the sender to define the name of the file on the remote system, send the full path name (not just name of the file)
	/iflarger	In Zmodem file transfers only, do not overwrite the host or remote file if it is smaller than, or the same size as, the file being sent from the local PC
	/ifnewer	In Zmodem file transfers only, do not overwrite the host or remote file if the file date and time is newer than, or the same as, the file being sent from the local PC
	/ifnotexist	In Zmodem file transfers only, do not overwrite a host or remote file of the same name if it already exists
	/nostatus	Do not display send file status dialog box
	/timeout=xx	Time-out in seconds to wait for a send file operation to start; if xx is 0, FileLink waits indefinitely

This script command sends one or more files to the remote system.

The protocols listed below automatically send the name of each file to the remote system in a preceding header block. This permits the receiver to automatically create a file with the same name as the original.

Kermit
Ymodem
Zmodem

When using one of these file transfer protocols, multiple files may be sent by using wildcard characters (*, ?) in the [file name]. Consider the following example in which all the files in a particular directory are sent to the remote system using the Zmodem protocol.

```
PROTOCOL "zmodem"
SENDFILE "c:\My Data\*.*" /timeout=30
```

The Kermit file transfer protocol does not specify a way to wait for the transfer to complete. Instead, the receiving side simply receives however much data is sent in the window of time allotted for receiving files.

When a wildcard is used to transfer multiple files, the number of files sent is stored in the % **sendfilecount** variable.

The following protocols do not support the sending of more than one file in a single SENDFILE command nor the use of wildcards.

ASCII
Xmodem
XmodemLK

Note

Automatic transmission of multiple files may be accomplished with these protocols using the GETNEXTFILE command and a conditional looping structure within a script file.

The **/archive** option automatically moves a local file to an archive folder defined using the [ARCHIVEDIR](#) script command after it has been successfully sent. If a file by the same name exists in the archive folder, it is overwritten.

The **/delete** option automatically deletes a local file after it has been successfully sent.

The **/iflarger**, **/ifnewer**, and **/ifnotexist** options may be used to control if host or server files are to be overwritten when sent by FileLink. These options are supported by Zmodem transfers only. These options are mutually exclusive. When using the **/ifnewer** option, the determination of the *newest* file is based on a comparison of the time and date each file was last written to. The file write time comparison is based only on the hour and minute.

The **/archive** option automatically moves a local file to an archive folder defined using the [ARCHIVEDIR](#) script command after it has been successfully sent. If a file by the same name exists in the archive folder, it is overwritten.

Related Command(s): [RCVFILE](#), [LINEIN](#), [LINEOUT](#), [PROTOCOL](#), [WORKINGDIR](#)

SENDMAIL -- Send an e-mail message

Syntax:	SENDMAIL	[server] [to name] [to email] [/options]
Arguments:	[server]	Variable or string defining the server URL or IP address (e.g., smtp.mail.server or 209.198.128.17) of a SMTP mail server; the server port is always set to 25
	[to name]	Variable or string defining the recipient
	[to email]	Variable or string defining e-mail address recipient
Options:	/pw=xx	Define optional password to use when logging on to the mail server (omit if not required by the SMTP server)
	/user=xx	Define optional user name to use when logging on to the mail server (omit if not required by the SMTP server)
	/timeout=nn	Time-out, in seconds, to wait for message to be sent (if omitted the time-out is set to 30 seconds)

This command sends an e-mail message previously created with the [CREATEMAIL](#) command via a SMTP e-mail server. All of the arguments to this command are required, however [to name] may be an empty string.

The same message is sent each time SENDMAIL is called unless CREATEMAIL is called again to change the message.

Consider the following example where an e-mail message is created and then sent.

```
;; create the message
SET from = "Serengeti Sales"
SET email = "sales@serengeti.com"
SET subj = "Thanks for your order!"
SET body = "We appreciate your business."
SET attach = ""
CREATEMAIL from email subj body attach
;; send the message
SENDMAIL "120.33.13.10" "Joe Blow" "joeb@smithco.com"
```

Consider the following example where authentication is required on the SMTP server.

```
SET server = "smtp.mail.server"
SET to = "Ray Johnson"
SET email = "rjj@laugh-in.com"
SENDMAIL server to email /user=smtpid /pw=smtppw
```

Related Command(s): [CREATEMAIL](#), [GETMAIL](#)

SET -- Assign or concatenate string variable(s)

Syntax:	SET	[variable] = [value] [& value2] [& value3] [& value4]
Alt Syntax:	SET	[variable] [&= value]
Arguments:	[variable]	Variable to assign; if the variable does not previously exist it is created.
	[value]	Variable or string defining the value to assign to [variable] or the first concatenation string when followed with the concatenation operator.
	&	The catenation operator; + is also recognized.
	[value2]	Variable or string defining the optional second string to concatenate to [value] with result stored in [variable].
	[value3]	Variable or string defining the optional third string to concatenate to [value] & [value2] with result stored in [variable].
	[value4]	Variable or string defining the optional fourth string to concatenate to [value] & [value2] & [value3] with result stored in [variable].
	[&= value]	Variable or string value to concatenate to [variable] with result stored in [variable] when the alternate concatenation syntax is used.
Options:	none	

This script command assigns a string value to a variable or concatenates up to four strings and assigned the resulting string to a variable.

Consider the following examples:

```
;; assign a variable to a string
SET new_string = "this is a new string"

;; assign a variable to a previously assigned variable
SET another_string = new_string

;; concatenate two strings
SET hello_world = "hello " & "world"

;; concatenate four variables
SET var1 = "FileLink "
SET var2 = "is "
SET var3 = "the "
SET var4 = "greatest."
SET truth = var1 & var2 & var3 & var4

;; concatenate two strings (alternate syntax)
SET hello_world = "hello "
SET hello_world &= "world"
```

Related Command(s): [INC](#), [DEC](#), [SETLEFT](#), [SETRIGHT](#), [SETMID](#)

See also: [Script File Variable Arguments](#)

SETEXTRACT -- Extract delimited substring from a string

Syntax:	SETEXTRACT	[variable] = [string] [delim] [num]
Arguments:	[variable]	Variable to assign; if the variable does not previously exist it is created.
	[string]	Variable or string defining the value from which the substring is to be extracted.
	[delim]	Variable or string defining the delimiter that separates substrings within [string].
	[num]	Variable, string, or numeric constant defining the occurrence of delimited substring to extract.
Options:	none	

This script command extracts the specified occurrence [num] of a delimited substring (excluding the delimiters themselves) from [string] and saves the result in [variable]. The delimiter may be one or more characters and is specified in [delim].

The beginning and end of [string] are seen as delimiters.

Consider the following example.

```
;; extract "is" from string
SET string = "this is a string"
;; "is" is the 2nd occurrence delimited by space (" ")
SETEXTRACT substring = string " " 2
;; substring now contains "is"

;; extract extension from file name of unknown size
;; for example, file name could be:
;; file.ext
;; file.file1.ext
;; file.file1.file2.ext
SETSUBSTR depth = filename "."
SETNUM depth = depth + 1
SETEXTRACT extension = filename "." depth
;; extension now contains "ext"
```

Related Command(s): [SET](#), [SETRIGHT](#), [SETLEFT](#), [SETLEN](#), [SETMID](#), [SETSUBSTR](#)

SETLEFT -- Extract left substring

Syntax:	SETLEFT	[variable] = [value] [cnt] [/options]
Arguments:	[variable]	Variable to assign; if the variable does not previously exist it is created.
	[value]	Variable or string defining the value from which the substring is to be extracted.
	[cnt]	Variable, string, or numeric constant defining the length of the substring; this value must resolve to a numeric value less than or equal to the length of [value].
Options:	/split	Save the unextracted portion of the string back in [value].

This script command extracts the leftmost [cnt] characters from [value] and saves the result in [variable].

If the **/split** option is specified, the unextracted portion of [variable] is assigned back to [variable] replacing the original value.

Consider the following example.

```
;; assign a variable to a string
SET string = "this is a string"
;; extract the leftmost 4 characters
SETLEFT substring = string "4"
```

The resulting `substring` variable contains the value "this". The original `string` variable is unchanged.

Consider the same example using the **/split** option.

```
;; assign a variable to a string
SET string = "this is a string"
;; extract the leftmost 4 characters
SETLEFT substring = string "4" /split
```

The resulting `substring` variable contains the value "this" and the original `string` variable has been reassigned to " is a string".

Related Command(s): [SET](#), [SETRIGHT](#), [SETMID](#)

SETLEN -- Assign length of specified string to a variable

Syntax: SETLEN [variable] = [string]
Arguments: [variable] [Variable](#) to assign; if the variable does not previously exist it is created.
 [string] Variable or [string](#) to get the length of.
Options: none

This script command obtains the length of [string] and saves the result in [variable].

Consider the following example.

```
;; assign a variable to a string
SET string = "this is a string"
;; get the length of the string
SETLEN len = string
;; len will be equal to 16
```

Related Command(s): [SET](#), [SETEXTRACT](#), [SETRIGHT](#), [SETLEFT](#), [SETMID](#), [SETSUBSTR](#)

SETMID -- Extract mid substring

Syntax:	SETMID	[variable] = [value] [cnt] [at]
Arguments:	[variable]	Variable to assign; if the variable does not previously exist it is created.
	[value]	Variable or string defining the value from which the substring is to be extracted.
	[cnt]	Variable, string, or numeric constant defining the length of the substring; this value must resolve to a numeric value less than or equal to the length of [value].
	[at]	Variable, string, or numeric constant defining the position in [value] where the substring extraction is to begin; the first character in a string is at position 1.
Options:	none	

This script command extracts [cnt] characters from within [value] beginning with character number [at] and saves the result in [variable].

Consider the following example.

```
;; assign a variable to a string
SET string = "this is a string"
;; extract 2 characters from mid-string starting at character #6
SETMID substring = string "2" "6"
```

The resulting `substring` variable contains the value "is".

Related Command(s): [SET](#), [SETRIGHT](#), [SETLEFT](#)

SETNUM -- Assign or evaluate numeric variable(s)

Syntax:	SETNUM	[variable] = [num1] [op] [num2]
Arguments:	[variable]	Variable to assign; if the variable does not previously exist it is created.
	[num1]	Variable, string , or numeric constant defining the <i>numeric</i> value to assign to [variable] or the first value to evaluate arithmetically with [value2] based on the numeric operator defined by [op].
	[op]	The arithmetic operator: + (addition), - (subtraction), x (multiplication), or / (division).
	[num2]	Variable, string, or numeric constant defining the second value to evaluate arithmetically with [value1] based on the numeric operator defined by [op].
Options:	none	

This script command performs basic integer arithmetic on variables or strings containing numeric values (e.g., digits 0 - 9) or numeric constants, and assigns the resulting numeric value to a variable. The command results in a syntax error if either [num1] or [num2] are non-numeric. The SETNUM command differs from the [SET](#) command in that SET does not limit the variable to contain only numeric digits (e.g., 0 - 9).

Consider the following examples.

```
;; assign a numeric constant to a variable
SETNUM num = 100

;; assign a previously assigned variable to a variable
SETNUM num = other_num

;; add two variables
SETNUM num = var1 + var2

;; add two numeric constants
SETNUM num = 100 + 100

;; following syntax is equivalent
SETNUM num = "100" + "100"

;; add/subtract/multiply/divide constant and a variable
SETNUM num = other_num + 100
SETNUM num = othernum - 100
SETNUM num = other_num x 100
SETNUM num = other_num / 100
```

Important

The multiplication operator is 'x' (lower-case letter x) and not the expected '*' (asterisk) character. This is because script file comments can begin with an asterisk.

If a numeric variable is to be used in file naming, or other string related operations, where a known string length (with leading zeroes) is desired, the SETNUM command (and INC and DEC commands) may be used as shown below.

```
;; leading zeroes are preserved when enclosed in quotes  
SETNUM num = "001"
```

```
INC num  
;; after incrementing, the result is "002"
```

```
DEC num  
;; after decrementing, the result returns to "001"
```

Note: Leading zeroes are lost if any other arithmetic operation is performed and saved to the num variable.

Related Command(s): [DEC](#), [INC](#), [IFNUM](#), [SET](#)

See also: [Performing Variable Arithmetic and Numeric Comparisons](#)

SETRIGHT -- Extract right substring

Syntax:	SETRIGHT	[variable] = [value] [cnt] [/options]
Arguments:	[variable]	Variable to assign; if the variable does not previously exist it is created.
	[value]	Variable or string defining the value from which the substring is to be extracted.
	[cnt]	Variable, string, or numeric constant defining the length of the substring; this value must resolve to a numeric value less than or equal to the length of [value].
Options:	/split	Save the unextracted portion of the string back in [value].

This script command extracts the rightmost [cnt] characters from [value] and saves the result in [variable].

If the **/split** option is specified, the unextracted portion of [variable] is assigned back to [variable] replacing the original value.

Consider the following example.

```
;; assign a variable to a string
SET string = "this is a string"
;; extract the rightmost 6 characters
SETRIGHT substring = string "6"
```

The resulting `substring` variable contains the value "string". The original `string` variable is unchanged.

Consider the same example using the **/split** option.

```
;; assign a variable to a string
SET string = "this is a string"
;; extract the rightmost 6 characters
SETRIGHT substring = string "6" /split
```

The resulting `substring` variable contains the value "string" and the original `string` variable has been reassigned to "this is a ".

Related Command(s): [SET](#), [SETLEFT](#), [SETMID](#)

SETSUBSTR -- Find number of substrings in string

Syntax:	SETSUBSTR	[variable] = [string] [substr]
Arguments:	[variable]	Variable to assign; if the variable does not previously exist it is created.
	[string]	Variable or string defining the value of string from which to obtain the delimiter count.
	[substr]	Variable or string defining the delimiter string to search for within [string]. A delimiter can be one or more characters in length.
Options:	None	

This script command searches [string] for occurrences of [substr] and saves the number of occurrences found in [variable].

Consider the following examples.

```
;; find the number of file name segments
SET filename = "data.temporary.sbc.xml"
;; find how many "." occur
SETSUBSTR dots = filename "."
;; dots will be equal to 3 (segments equals dots + 1)

;; determine if file name has an extension
SETSUBSTR dots = filename "."
IFNUM= dots 0 goto no_extension
```

Related Command(s): [SET](#), [SETEXTRACT](#), [SETRIGHT](#), [SETLEFT](#), [SETLEN](#), [SETMID](#)

SNAPSHOT -- Take a “snapshot” of the local PC file system

Syntax:	SNAPSHOT	[path] [dbfile] [/options]
Arguments:	[path]	Optional Variable or string defining the starting local path from which to take a “snapshot” of the file system; if omitted, the current working folder is used and [dbfile] defaults to “snapshot_local.sql”.
	[dbfile]	Optional Variable or string defining an alternative to the default “snapshot_local.sql” database file used to save the snapshot.
Options:	/includirs	Take a snapshot of the current or specified folder and all subfolders thereunder.

This script command is used in conjunction with the [DIFF](#) and [GETDIFF](#) script commands to locate individual file differences (i.e., change in size, date/time stamp) within a specified folder (and optional subfolder) tree within the local PC file system.

SNAPSHOT is the first step to establish a baseline (or “snapshot”) of the specified folder(s) from which to determine if any file(s) change. Details about files found are saved in an SQL database file. The database file name defaults to “snapshot_local.sql” but you may name it anything you like via the [dbfile] argument.

As long as unique and consistent [dbfile] arguments are specified, FileLink supports as many separate snapshots as you wish.

Consider the following example which takes a snapshot of the current working folder and any subfolders.

```
SNAPSHOT "*" /includirs
```

In most cases SNAPSHOT need only be run once. Future changes detected in the file system by the DIFF command are automatically updated in the database file unless FileLink is specifically told not to do so.

The total number of files examined in the local PC file system is saved in the **%snapshotfiles** script variable.

Related Command(s): [DIFF](#), [DIFFREWIND](#), [GETDIFF](#)

SPEAKER -- Control modem speaker mode

Syntax: SPEAKER [/options]
Arguments: None
Options: /on Turn speaker on until carrier is detected
 /off Leave speaker off at all times
 /always_on Leave speaker on at all times

This script command controls the speaker mode of an internal or external Hayes compatible modem.

Related Command(s): [DIAL](#), [ANSWER](#)

SRVNAME -- Define service name and control interaction with SrvMonitor

Syntax:	SRVNAME	[name] [/options]
Arguments:	[path]	Optional variable or string defining the service name of this FileLink instance for identification by the SrvMonitor applet.
Options:	/launch	Launch SrvMonitor after defining a service name.
	/off	Turn monitoring off.

This command defines the service name of the current FileLink session so that it can be identified when **SrvMonitor** (a Windows desktop tray applet) is started. This command is provided as an alternative to the [-t command line switch](#).

SrvMonitor is a separate utility that permits running FileLink session(s) to be monitored by any user when FileLink has been launched as an NT service or when it is otherwise running hidden in the background. See [Monitoring a FileLink Service](#) for more information.

If FileLink has been installed as a NT service by **SrvInstaller**, then use of this command is not necessary unless you wish to disable monitoring via the **/off** option. When launched by **SrvInstaller**, the identification string used by **SrvMonitor** is set to the service name specified when the service is created.

The following command defines a service name and enables FileLink interaction with **SrvMonitor**. The name specified will appear in **SrvMonitor** windows and/or menus to identify this instance of FileLink.

```
SRVNAME "MyFileLink"
```

The following command defines a service name and automatically launches **SrvMonitor**. This option is not allowed if FileLink is running as an NT service.

```
SRVNAME "MyFileLink" /launch
```

The following command disables FileLink interaction with **SrvMonitor**. Interaction may be restored by (re)issuing the previous command.

```
SRVNAME /off
```

STOP -- Stops script processing

Syntax: STOP
Arguments: None
Options: None

This script command ends script processing. The action is the same as clicking the Stop button on the FileLink toolbar or pressing the **(Esc)** key while a script file is executing.

If FileLink is running in a minimized window (i.e., as an icon) or as an NT service when STOP is executed, the action is the same as executing the EXIT command - namely, FileLink terminates.

Related Command(s): [EXIT](#)

TERMINAL -- Activate Terminal applet

Syntax: TERMINAL
Arguments: None
Options: None

This command not supported when running in a minimized window or as an NT Service.

This script command suspends script processing and activates the FileLink TTY Terminal applet. The action of this command is the same as clicking on the Start TTY Terminal Applet on the FileLink toolbar. You may switch freely between the TTY Terminal applet and the FileLink script environment.

Each environment shares the same communications session, so you may connect to or disconnect from the remote system, for example, from either a script file or from within the TTY Terminal applet. The same set of configuration settings apply to both environments.

The TTY Terminal applet is useful to understand how a communications session with a particular remote system takes place so that it can be automated with a script file.

See also: [Using the FileLink TTY Terminal Applet](#)

TRACELOG -- Control the trace log file

Syntax:	TRACELOG	[file name] [/options]
Arguments:	[file name]	Optional variable or string defining a file or path name; if no path is defined FileLink's working folder is used.
Options:	/append	Specify that trace data is to be appended to preexisting [file name] (if any); if the file does not exist, it will be created.
	/maxsize=xx	Specify the maximum size of the trace log file (in Kilobytes).
	/new	Specify that a new trace log file name is to be created (based on the current date and time) when this command is executed and whenever an existing trace log file reaches the maximum size (if a size has been specified).
	/off	Turn trace logging off.
	/on	Turn trace logging on

If the [file name] argument is present, this script command creates a new trace log file by this name. It is also implied that logging is to be turned on. The trace log file records trace and diagnostic messages that may be helpful in troubleshooting file transfer failures.

The **/new** option instructs FileLink to create new trace file name using the current date and time. Such a file will be created when the command is executed and, if the **/maxsize** option is also specified, whenever the trace file exceeds this maximum size. The [file name] argument must be present, but it can be an empty string. FileLink takes the base file name (i.e., the part of the file name before any extension) and appends the current date and time in the fashion shown below. Notice that if no extension is originally specified, FileLink appends **.log** to the final file name.

```
TRACELOG "trace" /new
// creates trace file = trace_Wed Oct 30 15.38.43 2002.log

TRACELOG "trace.txt" /new
// creates trace file = trace_Wed Oct 30 15.38.43 2002.txt

TRACELOG "trace.xx.log" /new
// creates trace file = trace_Wed Oct 30 15.38.43 2002.xx.log

TRACELOG "" /new
// creates trace file = Wed Oct 30 15.38.43 2002.log
```

The **/maxsize** option limits the maximum size that a trace file can grow to. The size is specified in kilobytes. When the maximum size is reached, FileLink handles this condition in one of two ways. If **/new** is also specified, the current trace file is simply closed and a new file is created using the convention described above. If **/new** is not present, FileLink toggles between two files. When the first file is full, it is closed and a second created and written to. When the second file is full, it is closed and the first file is reopened, cleared, and logging continues. This alternating between files continues until FileLink terminates. When **/maxsize** is present, [file name] is altered as shown below. Notice that if no extension is originally specified, FileLink appends **.log** to the final file name.

```
TRACELOG "trace" /maxsize=100
```

```
// creates trace file = trace_1.log
// this alternates with a file to be named trace_2.log

TRACELOG "trace.txt" / maxsize=100
// creates trace file = trace_1.txt
// this alternates with a file to be named trace_2.txt
```

The **/append** option instructs FileLink to append new trace log data to a previously existing file specified by [file name]. If [file name] does not exist, it will be created. The **/append** option may be combined with **/maxsize** but may not be used with the **/new** option.

If [file name] is omitted, the **/on** and **/off** options control logging to a previously defined log file. When logging is turned on, new log messages are appended to the existing log file. For example:

```
TRACELOG /off
```

Related Command(s): [LOG](#), [WORKINGDIR](#), [TRACEWIN](#)

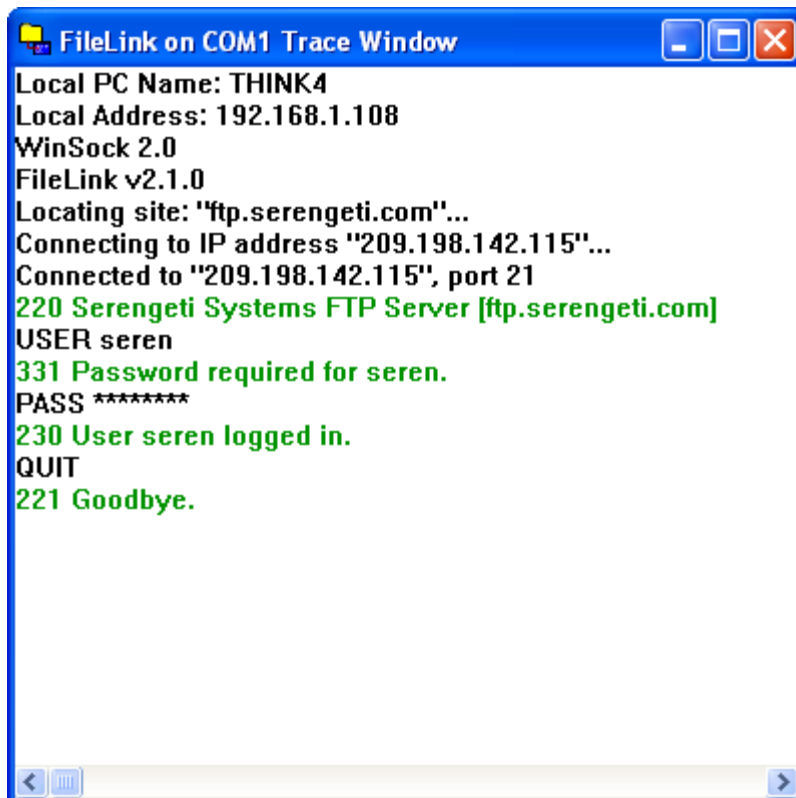
TRACEWIN -- Activate/deactivate trace window

Syntax: TRACEWIN [/options]
Arguments: None
Options: /off Deactivate and remove trace window.
/on Activate and display trace window.

This command not supported when running as an NT Service.

When testing script commands, it is often helpful to activate the FileLink Trace Window. The contents of this window mirror what is written to the trace log file. For example, characters read from and written to a COM port with the LINEIN and LINEOUT commands respectively, are echoed to the trace window giving you an immediate, visual record of what is occurring on the communications link.

Consider the following example where the trace window shows the results of LINEIN and LINEOUT commands on a COM port.



The script command name, the number of characters sent or received, enclosed in (..), and the characters themselves are shown in the window.

The contents of the trace window can be copied to the Clipboard by accessing the System menu (by clicking on the icon in the upper left corner of the window) and then selecting **Copy to Clipboard**. This is useful if you need to save the contents of the trace window for later examination.

Related Command(s): [TRACELOG](#)

UNZIP - Extract file(s) from a zip archive

Syntax:	UNZIP	[zip name] [path] [file] [/options]
Arguments:	[zip name]	A variable or string defining the file name of the zip archive; if .zip extension is omitted, FileLink adds it automatically; if no folder is specified, FileLink creates or opens the archive in its working folder.
	[path]	A variable or string defining the path to the target folder to where files are to be extracted; if no path name is specified, FileLink uses the current working folder. If specified, this folder must exist.
	[file]	A variable or string defining the file name(s) to be extracted.
Options:	/pw=xx	Define password used when the zip archive was created.
	/skipexisting	Do not extract files from the archive that already exist in the target path.
	/skipolder	Do not extract files from the archive that are older than, or have the same date and time as existing files in the target path.
	/subdirs	Use folder names stored in the archive; if this option is not selected, all files will be extracted to [path] regardless of path information saved in the archive.

This script command extracts file(s) from an existing zip archive file. Upon completion of the command, the **%unzipcount** script variable contains the total number of files unzipped by this command.

Consider the following example in which a single file is extracted to FileLink's working folder.

```
UNZIP "zipfile" "" "mydata.xml"
```

The example below extracts all the .xml files in the specified archive to FileLink's working folder.

```
UNZIP "zipfile" "" "*.xml"
```

The example below extracts all the .xml files in the specified archive to FileLink's working folder and into subfolders that may have been saved in the archive. Relative path names of any files found in the archive are restored.

```
UNZIP "zipfile" "" "*.xml" /subdirs
```

The example below extracts a file named `index.html` from a password protected archive and saves in a target folder named `c:\mysite`.

```
UNZIP "zipfile" "c:\mysite" "index.html" /pw=mysecret
```

The example below extracts all newer files in the specified archive to the specified target folder.

```
UNZIP "c:\temp\archive.zip" "c:\my_files" "*.*" /skipolder
```

If you have multiple file to unzip, the example below shows a loop that gathers zip files, one by one, from the current folder and extracts all files in the specified archive to the specified target folder.

```
:loop
GETNEXTFILE "*.zip" /timeout=10
;; branch if no more zip files
IFERROR $ERROR_WAIT_TIMED_OUT goto nofile
UNZIP %nextfile "c:\myfiles" "*.*"
GOTO loop
:no_file
```

Related Command(s): [WORKINGDIR](#), [ZIP](#)

USEPORT -- Specify COM port and/or port settings

Syntax:	USEPORT	[port] [/options]
Arguments:	[port]	Optional variable or string defining the COM port; COM1through COM48 are supported by FileLink; if this argument is omitted, the options apply to the current COM port
Options:	/baudrate=xx	Set the baud rate of the communications link; replace xx with one of the following: 1200 4800 9600 14400 19200 28800 33600 38400 57600 115200
	/carrier=xx	Set the carrier setting of the communications link; replace xx with one of the following: Constant Ignore Switched
	/flowcontrol=xx	Set the flow control setting of the communications link; replace xx with one of the following: None Both Hardware Xon/Xoff
	/parity=xx	Set the parity of the communications link; replace xx with one of the following: Even Mark None Odd Space
	/stopbits=x	Set the number of stop bits used on the communications link; replace xx with one of the following: 1 2
	/wordlength=x	Set the number of data bits per character used on the communications link; replace x with one of the following: 7 8

This script command defines the COM port and/or port settings to be used by FileLink. If the line is not connected, the COM port and settings apply the next time the communications port is opened. If the line is connected, the settings take effect immediately. Changing the COM port itself when the line is connected is prohibited.

The selection made in this command overrides any default selections set with the [FileLink Configurator](#). If the specified COM port has not been previously configured, default settings for the port are created. Not all configuration options must be supplied with the USEPORT command - when omitted, the selection set at configuration time is used.

Consider the following example where COM2 is selected and configured.

```
USEPORT "COM2" /baudrate=28800 /parity=none /wordlength=8
```

Consider the following example where the baud rate of the current port is changed.

```
USEPORT /baudrate=9600
```

The following example restores the default settings for COM2.

```
USEPORT "COM2"
```

Related Commands: [PROTOCOL](#)

WORKINGDIR -- Specify default working folder

Syntax: WORKINGDIR [folder name]
Alt Syntax: CHGDIR [folder name]
 CD [folder name]
Arguments: [path name] A [variable](#) or [string](#) to specify the path name of FileLink's
 default folder.
Options: None

This script command defines the default folder for all file oriented script commands. Anytime a file name is specified in a command without a full path associated with it, FileLink either searches for or creates this file in the designated working folder. The selection made in this command overrides the default selection set with the FileLink configurator.

The current working folder is always maintained in the **%currentlocaldir** variable.

Consider the following example.

```
;; read record in "c:\Program Files\FileLink\example.txt"  
WORKINGDIR "c:\Program Files\FileLink"  
READLINE "example.txt" first_record
```

If a partial path is specified, it and the previous working folder are used to define the new working folder.

```
;; current working folder is "c:\Program Files\FileLink"  
WORKINGDIR "test"  
;; new working folder is now "c:\Program Files\FileLink\test"
```

If ".." is specified, the new working folder is set one folder above the previous working folder.

```
;; current working folder is "c:\Program Files\FileLink"  
WORKINGDIR ".."  
;; new working folder is now "c:\Program Files"
```

Caution

The directory change made by WORKINGDIR is global within the FileLink script environment. If WORKINGDIR is going to be called in a function or a called script, it is always recommended that the current working folder be saved on entry and restored on exit if it is going to be changed. See the description of the **%currentlocaldir** variable for [an example of this](#).

Related Command(s): [ARCHIVEDIR](#), [MAKEDIR](#)

WRITEFILE -- Write character string or string variable value to text file

Syntax:	WRITEFILE	[file name] [string_out] [/options]
Arguments:	[file name]	A variable or string to specify the file name to write to; if no path is defined FileLink's working folder is used; if the file does not exist, it is created.
	[string_out]	A variable or string to be written to the file.
Options:	/append	Append the string to the existing file.
	/hex	Interpret the output string as a hexadecimal value rather than as standard ASCII characters. Use this option if you need to write binary data.

This script command writes characters to the specified text file. This command is oriented toward writing a complete record of printable characters, terminated by a carriage-return/ line-feed, to a file. The scope of this command (and the READFILE command) is not to provide full function file I/O to your script files, but rather to provide temporary storage for small amounts of information for use by a script file or an external program.

By default WRITEFILE either creates a new file or over-writes an existing file with what is written. You may use the **/append** option add records an existing file.

Consider the following example.

```
;; write a string received from the COM port to a file
LINEIN user_id /timeout=0
WRITEFILE "user_info.txt" user_id /append
```

Note: The [%crlf](#) internal variable is available for adding additional line breaks to your output string.

Related Command(s): [READFILE](#), [LINEIN](#), [LINEOUT](#), [WORKINGDIR](#)

ZIP -- Create or add to a zip archive

Syntax:	ZIP	[zip name] [file] [/options]
Arguments:	[zip name]	A variable or string defining the file name of the zip archive; if .zip extension is omitted, FileLink adds it automatically; if no folder is specified, FileLink creates or opens the archive in its working folder.
	[file]	A variable or string defining the folder or file name(s) to add to the zip archive; if no path name is specified, FileLink assumes [file] is in the current working folder.
Options:	/compress=xx	Select the compression mode for the zip archive; options are /compress=none (files are stored in the zip archive but not compressed); /compress=fast (archive file is created as quickly as possible but file may not be as small as it could be); /compress=normal ; /compress=max (archive file is made as small as possible but may take longer to create).
	/create	Create a new zip archive each time; any existing archive file named [zip name] will be deleted and recreated.
	/encryption=xx	Select the encryption method. The default encryption method is XEM compatible. You can also specify /encryption=AES to use AES encryption. AES offers a more secure algorithm, though it can increase the time it takes to compress the file. Also /encryption=none can be used to not encrypt. Note that the encryption setting is ignored if no password is specified.
	/fullpath	Save the fully qualified path or folder names of files as they are stored in the zip archive.
	/pw=xx	Define a password to protect the files added to the zip archive.
	/subdirs	Add files in subfolders beneath [zip name] to the zip archive; [zip name] must be a folder or a path containing a wildcard pattern for this option to be accepted.

This script command creates a zip archive file from the file or files. A new archive file may be created each time or file(s) may be added to an existing archive.

Upon completion of the command, the **%zipcount** script variable contains the total number of files zipped by this command.

Consider the following example in which a single file is zipped into a new archive. The resulting archive file is named `zipfile.zip` and is located in FileLink's working folder.

```
ZIP "zipfile" "mydata.xml" /create
```

The following example adds another file to the same archive.

```
ZIP "zipfile" "mydata2.xml"
```

The example below adds all the .xml files in FileLink's working folder to an existing password protected archive.

```
ZIP "zipfile" "*.xml" /pw=mysecret
```

The example below adds all the .xml files in FileLink's working folder and in any subfolders to an existing archive. Relative path names of any files found in subfolders are saved in the archive.

```
ZIP "zipfile" "*.xml" /subdirs
```

The example below stores (but does not compress) all the files in FileLink's working folder and in any subfolders in a new archive. Relative path names of any files found in subfolders are saved in the archive.

```
ZIP "c:\temp\archive.zip" "*.*" /subdirs /compress=none /create
```

Related Command(s): [UNZIP](#), [WORKINGDIR](#)

Sample Script Files

See the sample scripts listed below.

[Simple Async Dial-Up Connection](#)

[Simple Async Dial-Up Connection With Error Recovery](#)

[Dial-Up Connection Performing a Logon](#)

There are also two functional script files provided with FileLink:

prompt.s	script to provide interactive control where you can type script commands to be executed
ssitest.s	script to conduct prearranged test with Serengeti Systems

Simple Async Dial-Up Connection

The example script file shown below attempts to dial up to three times. Once connected, it sends a file named 'login'. If the send file completes normally, the script file waits for a single file from the remote system. Once a file is received, or the RCVFILE command times out, the script file disconnects and exits.

```
LOOPCOUNT 3
:dial_loop
DIAL "1-555-1212"
LOOPIF goto dial_loop else goto connect
EXIT
:connect
SENDFILE "login"
IFERROR goto sendfail
RCVFILE /timeout=60
:sendfail
DISCONNECT
EXIT
```

Simple Async Dial-Up Connection With Error Recovery

This example builds on the previous one and adds some error recovery and reporting. As before, once connected, it sends a file named 'login'. If the send file completes normally, a second file is sent. If it is also sent successfully, the script file waits for a single file from the remote system. Once a file is received, or if the receive command times out, the script file disconnects and exits.

```
DIAL "1-512-555-1212"
;; check for connection
IFERROR= $ERROR_CONNECT_TIMEOUT goto connect_timeout
;; check for line busy
IFERROR= $ERROR_BUSY_SIGNAL goto line_busy
;; just exit if any other error
IFERROR goto exit
;; send two files in a row
SENDFILE "login" /timeout=30
;; check for send time-out
IFERROR= $ERROR_OPTIMEDOUT goto xmt_timeout
SENDFILE "accounts"
;; wait for host reply
RCVFILE "newdata" /timeout=60
DISCONNECT
:exit
EXIT
:connect_timeout
MESSAGEBOX "***ATTENTION: Cannot connect, dial time-out"
EXIT
:line_busy
MESSAGEBOX "***ATTENTION: Line is busy"
EXIT
:xmt_timeout
MESSAGEBOX "***ATTENTION: Remote unable to receive"
DISCONNECT
EXIT
```

Dial-Up Connection Performing a Logon

In this example, a common sequence of performing a remote logon (being prompted for and sending a user ID and a password) in an async modem environment is shown.

```
DIAL "555-1212" /timeout=60
;; check for connection
IFERROR!= $ERROR_SUCCESS goto Exit
;; Accept a character string command from the remote system
;; (assuming that a prompt for a user name is expected)
;; (we don't really care what this string is)
LINEIN cmdvariable /flush /timeout=60
IFERROR= $ERROR_SUCCESS goto Operation2
GOTO Disconnect
:Operation2
;; Send a user name or ID string to the remote system
LINEOUT "username" /flush /timeout=60
IFERROR= $ERROR_SUCCESS goto Operation3
GOTO Disconnect
:Operation3
;; Accept a character string command from the remote system
;; (assuming that a prompt for a password is expected)
;; (we don't really care what this string is either)
LINEIN cmdvariable /flush /timeout=60
IFERROR= $ERROR_SUCCESS goto Operation4
GOTO Disconnect
:Operation4
;; Send a password string to the remote system
LINEOUT "password" /flush /timeout=60
IFERROR= $ERROR_SUCCESS goto Operation5
GOTO Disconnect
:Operation5
;; here the script might send or receive a file
;; or perform other required operations
. . .
:Disconnect
DISCONNECT
:Exit
EXIT
```

Dial-In Connection With Authorization

In this example, FileLink is acting as host for remote dial-in users in an async modem environment. FileLink prompts for and verifies the user name and password of the caller.

```

ANSWER /timeout=0
;; check for connection
IFERROR!= $ERROR_SUCCESS goto Exit
;; Prompt for user name and wait 60 seconds for a response
LINEOUT "Enter your user name:" /flush
LINEIN username /flush /timeout=60
IFERROR= $ERROR_SUCCESS goto Operation2
GOTO Disconnect
:Operation2
;; Validate that the user name is one we recognize
;; Authorization records have been pre-created in "authfile.txt"
AUTHUSER username "authfile.txt"
IFERROR= $ERROR_AUTHORIZATION_FAILED goto Disconnect
;; User was found in authorization file
;; Prompt for password and wait 60 seconds for a response
LINEOUT "Enter your password:" /flush
LINEIN password /flush /timeout=60
IFERROR= $ERROR_SUCCESS goto Operation3
GOTO Disconnect
:Operation3
;; Validate that the password for this user
AUTHPW username password "authfile.txt"
IFERROR= $ERROR_AUTHORIZATION_FAILED goto Disconnect
;; Password matched for this user
;; Send predefined greeting #1 to this user
AUTHDATA username greeting "1" "authfile.txt"
LINEOUTLINEOUT greeting /flush
:Operation4
;; here the script might send or receive a file
;; or perform other required operations
. . .
:Disconnect
DISCONNECT
:Exit
EXIT

```

In this example, an entry in the authorization file might look like this:

```
benfrankin,bennyf,Thanks for calling
```

Installing FileLink as an NT Service

FileLink may be installed as an NT Service under Windows NT, 2000, and XP. Services are not supported under Windows 98 or ME.

You must have administrator privileges you install the FileLink Service.

An NT Service is a background process which is loaded by the *Service Control Manager* of the Windows kernel. They are often loaded at bootup, before any user logs in, and are often independent of any specific user being logged on at the time. If a service is not launched automatically by the system at boot time, as many services are, it can also be manually launched by a user at the console, via the Windows Control Panel Services tool (under **Administrative Tools** using Windows 2000 and XP), or by another program which interfaces to Windows's Service Control Manager.

The **SrvInstaller** utility is installed along with FileLink. It is installed into the Windows Control Panel under the name **FileLink Service Installer** and it may also be found in the Start menu grouped with other FileLink components. This utility is used to install, optionally start, and to stop and/or remove the FileLink Service.

The **SrvMonitor** utility is also installed along with FileLink. **SrvMonitor** is a Windows desktop tray applet that any user can use to monitor the status of a running FileLink service. See [Monitoring a FileLink Service](#).

A script file must be specified when running FileLink as a Service. The FileLink Service remains active as long as the script file permits it to. The [STOP](#) and [EXIT](#) commands stop the script and terminate the FileLink Service. Furthermore, unless you have selected the **Interact with desktop** option, interactive script commands such as [ASK](#) and [PROMPT](#) are not permitted.

A typical deployment of FileLink as a Service would probably have a script that runs continuously. Such a script file would use, for example, the [CRON](#) command to trigger one or more regularly scheduled activity, or use the [GETNEXTFILE](#) command to signal a new operation is to be initiated.

Important

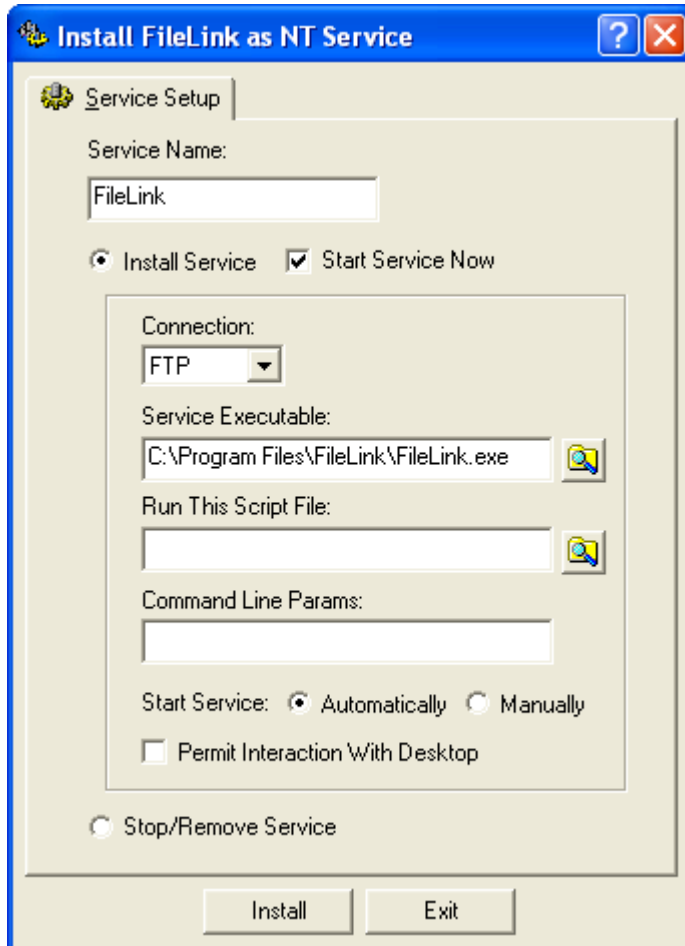
It is strongly recommended that you fully test any script file that you intend to run using FileLink as a normal application program before using it with a Service.

If the FileLink Service is not started at installation time, you use the Windows Control Panel Services tool (under **Administrative Tools** using Windows 2000 and XP) to start the Service, or you should restart Windows (assuming that you selected Automatically as the **Start Service Now** method).

It is recommended that you use the Service installation utility **SrvInstaller** to stop and remove the FileLink Service rather than using the Windows Control Panel Services tool (under **Administrative Tools** using Windows 2000 and XP).

Unlike other NT Services, the FileLink Service does not log all results and errors to the System Event Log. Instead, you should use the FileLink log file to monitor operation of a FileLink Service that does not interact with the desktop.

The main screen of the **SrvInstaller** utility is shown below. Click on the fields within image below for more information.



Shutting Down a Running FileLink Service

Using the **Stop/Remove Service** function of [SrvInstaller](#) has been mentioned elsewhere as the proper way of shutting down a FileLink Service. This always works but it does not always result in a clean and orderly termination when a communications session is in progress. This requires some special attention during the creation of script files used by the FileLink Service.

The idea is to have the script regularly monitor for the presence of a special shutdown script. If it becomes necessary to terminate the FileLink Service, this shutdown script would be copied to a designated folder, the FileLink Service detects its presence, and transfers control to it using the [CHAIN](#) script command at an appropriate time. Then the stop may be issued from [SrvInstaller](#).

The shutdown script, we'll name the file "shutdown.s", might look like the following:

```
; FileLink Service shutdown script
DISCONNECT
EXIT Script_Commands_(QUIT)CONNECT
```

Your production script file(s) are obviously application dependent, but any script used with a FileLink Service should continuously loop looking for something to do. The [GETNEXTFILE](#) command is often used to monitor a folder for a file to transmit, so we'll use this construct to demonstrate how to use a shutdown script.

```
;; FileLink Service production script
:top
;; look for any file in current folder
GETNEXTFILE "*.*" /timeout=10
;; branch if 10 seconds elapsed
IFERROR $ERROR_WAIT_TIMED_OUT goto nofile
;; branch on any other error
IFERROR goto some_error
;; connect with remote system
CONNECT
;; send the file we just found
SENDFILE %nextfile
;; disconnect from remote
DISCONNECT
;; loop back to next file to send
GOTO top
:no_file
;; branch if no shutdown script
IFNFILE "shutdown.s" goto top
;; transfer to shutdown script
CHAIN "shutdown.s"
:some_error
...
```

Obviously there are other ways to skin this cat, but the important thing is to have the FileLink Service execute a DISCONNECT and/or EXIT commands prior having the Service stopped using **SrvInstaller**. This permits the FileLink Service to perform an orderly disconnect (if necessary) and termination of a FileLink communication session.

Monitoring a FileLink Service

The **SrvMonitor** utility, a Windows desktop tray applet, is provided with FileLink to provide a way to monitor the operation of FileLink when it is being run as an NT service (which normally has no interaction with a user desktop) or when it is being run minimized or otherwise hidden from view.

SrvMonitor provides the following real-time information for a user interested in what FileLink is doing:

- Currentt state (e.g., idle, sending a file)
- Currently active script file
- Current line number of the active script
- Scrolling messages issued by FileLink to the unseen console and trace windows

SrvMonitor identifies an instance of FileLink by a unique character string identifier known as a **service name**. FileLink is given a service name in one of three ways:

1. Via the **Service Name** field in **SrvInstaller** when FileLink is launched as an NT service. This is done automatically.
2. Via the **-t** command line switch added manually to shortcut that launches FileLink. See [Command Line Switches](#).
3. Via the [SRVNAME](#) script command when neither (1) nor (2) have been done. The SRVNAME command also allows monitoring to be turned on and off under script control.

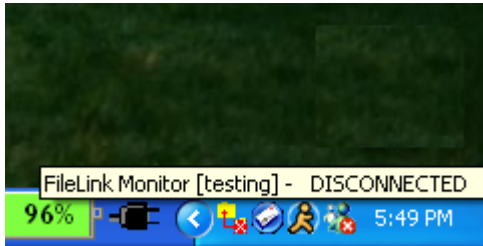
When **SrvMonitor** launches, it locates any existing FileLink service name(s). If only one name is found, **SrvMonitor** immediately associates itself with this instance of FileLink. If more than one name is found, **SrvMonitor** displays a menu that allows you to choose the association you desire.

SrvMonitor also supports a **-t** command line switch which you may define in any shortcut that launches **SrvMonitor**. This permits you to specify a known service name - this is useful if you wish to monitor one of multiple instances of FileLink and suppress the menu where the desired instance must be chosen.

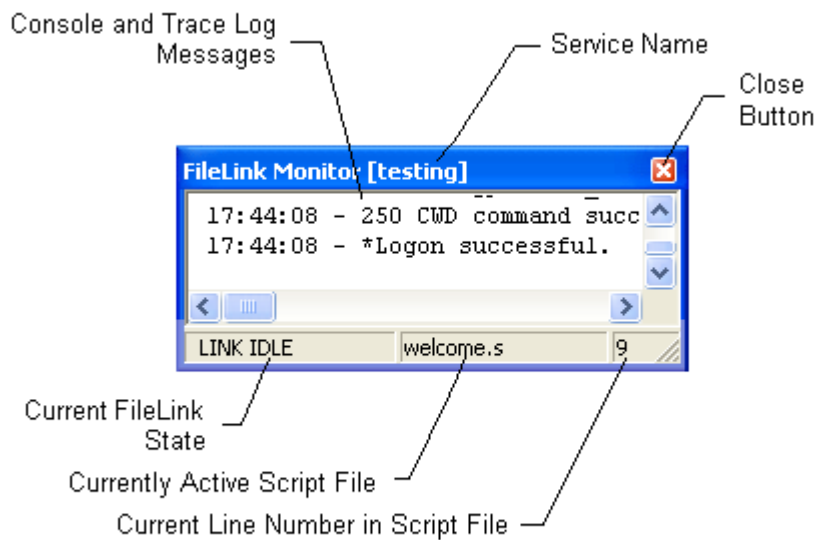
When initially launched, **SrvMonitor** appears as an icon in the Windows desktop tray. The icon itself conveys the state of FileLink as shown below.

-  Service Down
-  Disconnected
-  Link Idle
-  Sending File
-  Receiving File

When you hove the mouse cursor over the icon, you get a little more information as shown below.



When you doubleclick on the icon, **SrvMonitor** opens into a small window like what is shown below.



Click the close button to return **SrvMonitor** to its icon state.

In the minimized state, right click on the icon for a control menu. The control menu is shown below and each menu function is the described.



Show Service Monitor - Opens the **SrvMonitor** window (same action as double clicking on the tray icon).

Hide Service Monitor - Closes the **SrvMonitor** and returns it to its icon state.

Clear All Saved Service Names - Reset **SrvMonitor** to monitor services with new or different names. This is useful when you have selected “Always monitor the instance with this name” option when first starting **SrvMonitor** and you wish to now monitor additional services or one with a different name.

Exit Service Monitor - Terminate **SrvMonitor**.

Using the CronMaker Utility

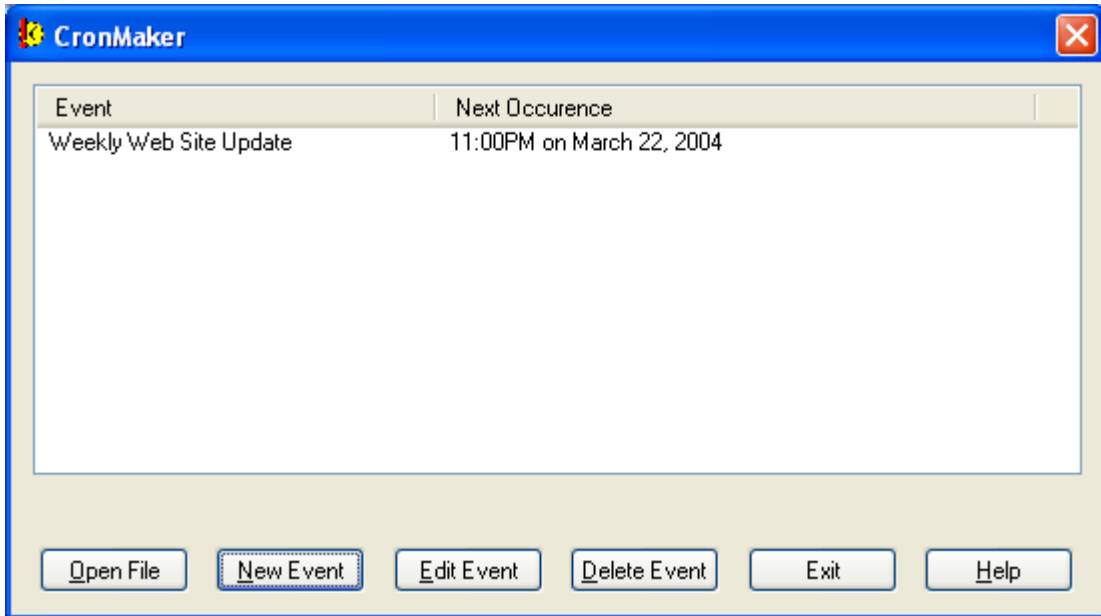
The FileLink CronMaker utility is provided to make dealing with the cron event file used with the [CRON](#) script command much easier.

The format of cron event files is described elsewhere in this help file (see [Cron Event File Format](#)) but the manual creation and management of this file for FileLink scheduling purposes is not intuitive or for the technically faint of heart. The CronMaker utility comes to the rescue.

In CronMaker, you create, edit, and delete *events*. Each event is named (e.g., Weekly Web Site Update) and corresponds to one scheduled CRON action.

When CronMaker is launched from within FileLink via the toolbar or menu control, it automatically attempts to open the default cron event file (name “crontab.txt”) in the current working folder. In the case where CronMaker might be launched from the **Start** menu shortcut, use the **Open File** button to select and open the desired file.

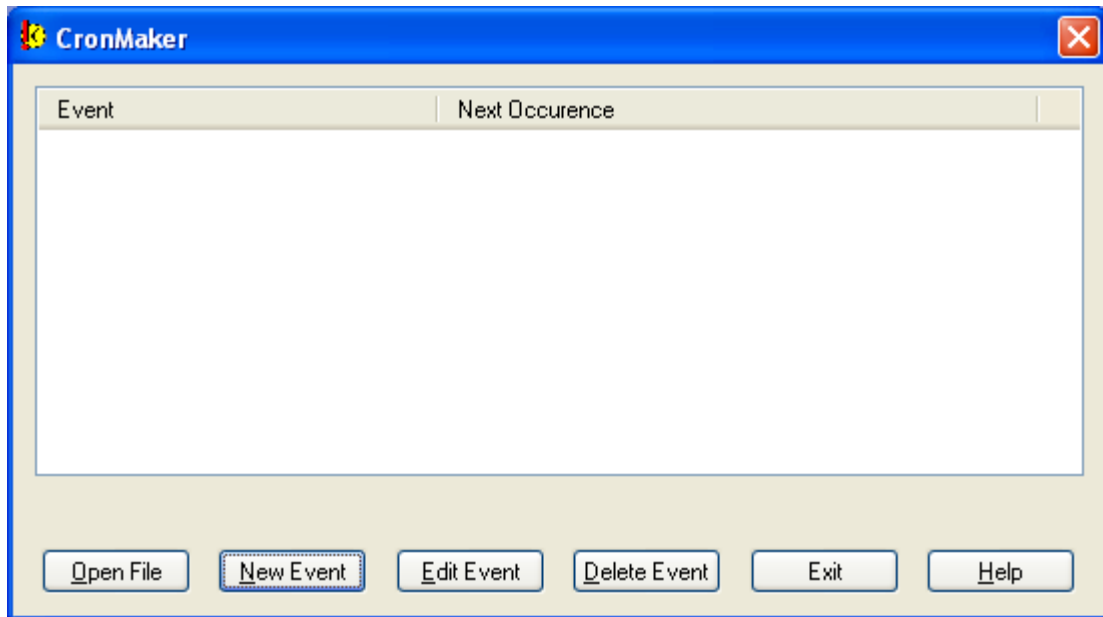
The main CronMaker screen with an example event is shown below. Click on the buttons and event table area within the graphic for more information. The steps showing how this example event was created are presented [here](#).



CronMaker Event Creation Example

The screen shots shown below is an example of how the “Weekly Web Site Update” cron event shown on the previous page was created.

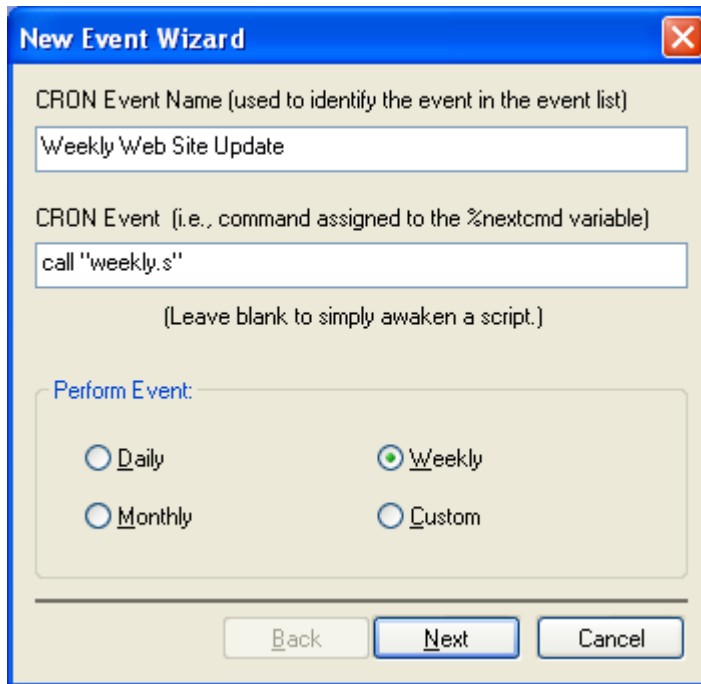
When first started and no events are scheduled, CronMaker displays blank screen as shown below:



Click the **New Event** button to begin.

CronMaker Event Creation Example P2

The New Event Wizard is now active. The first screen you see is shown below.



The screenshot shows a dialog box titled "New Event Wizard" with a close button (X) in the top right corner. The dialog contains the following fields and options:

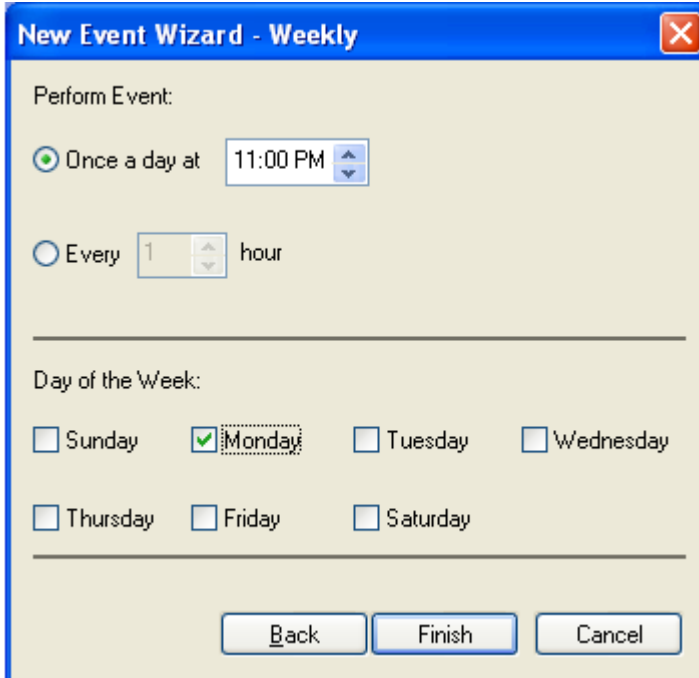
- CRON Event Name (used to identify the event in the event list)**: A text box containing "Weekly Web Site Update".
- CRON Event (i.e., command assigned to the %nextcmd variable)**: A text box containing "call 'weekly.s'".
- (Leave blank to simply awaken a script.)**: A note below the CRON Event field.
- Perform Event:** A section containing four radio button options:
 - Daily
 - Weekly
 - Monthly
 - Custom
- Buttons:** "Back", "Next", and "Cancel" buttons at the bottom.

Here you enter the event name (i.e., Weekly Web Site Update) and the script operation that the [CRON](#) script command is to perform when the event is triggered (assign "call 'weekly.s' to the [%nextcmd](#) script variable). You also indicate when the event should be triggered. In the case of this example, weekly.

When complete, click the **Next** button to continue.

CronMaker Event Creation Example P3

Once the event name and the optional event action is entered and weekly frequency is selected, you'll see the following page from the New Event Wizard.



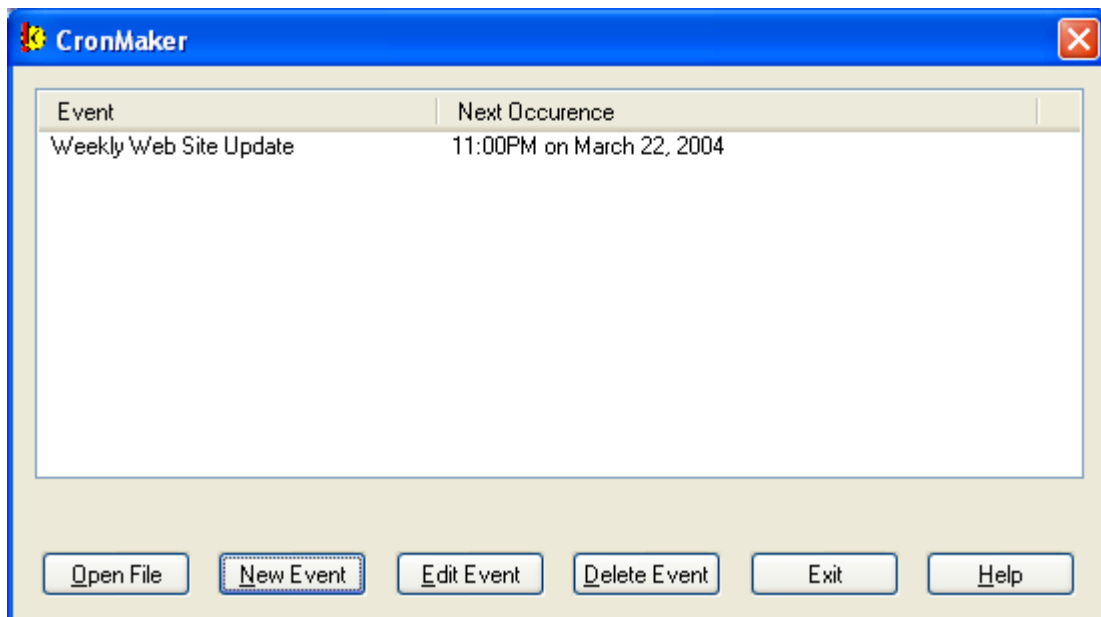
The screenshot shows a dialog box titled "New Event Wizard - Weekly". It has a blue title bar with a close button. The main area is light beige. Under "Perform Event:", there are two radio buttons. The first is selected and labeled "Once a day at" with a time dropdown set to "11:00 PM". The second is labeled "Every" followed by a spinner box containing "1" and the word "hour". Below this is a horizontal line. Under "Day of the Week:", there are seven checkboxes for the days of the week: Sunday, Monday (checked), Tuesday, Wednesday, Thursday, Friday, and Saturday. At the bottom are three buttons: "Back", "Finish", and "Cancel".

Here you select the hour of the day that you want the event to trigger and the day of the week. In this example, 11:00PM on Monday. As you can see, you could also specify that the event trigger hourly, or at multiple hour intervals, on one or more designated days.

When complete, click the **Finish** button to complete the creation of the cron event.

CronMaker Event Creation Example P4

Once the event has been created, CronMaker returns to the [main screen](#).



Here you can create additional events, edit or delete existing ones, or exit from CronMaker.

Cron Event File Format

Borrowed from Unix, a **cron event file** (named **crontab** in the Unix world) contains instructions to the FileLink [CRON](#) script command of the general form: "run this command at this time on this date".

The format of this file is complex and not something the casual user needs to be concerned with. The [CronMaker Utility](#) is provided with FileLink for the direct creation and modification of "crontab.txt" files. The curious may read on for the "techie" details.

Each line of the file has five time and date fields, followed by an optional command to be saved in the **%nextcmd** variable. Blank lines, leading spaces, and tabs are ignored. Lines whose first non-space character is a pound-sign (#) are comments and are ignored. Note that comments are not allowed on the same line as CRON commands, since they will be taken to be part of the command.

Commands are executed by CRON when the **minute**, **hour**, and **month** fields match the current time, and when at least one of the two day fields (**day of month** or **day of week**) match the current day. The time and date fields are:

<u>Field</u>	<u>Definition</u>
Minute	Minutes after hour (0 - 59)
Hour	Hours since midnight (0 - 23)
Day of month	Day of month (1 - 31)
Month	Month (0 - 11; January = 0); or use name
Day of week	Day of week (0 - 6; Sunday = 0); or use name

A field may be an asterisk (*), which always stands for "don't care".

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, 8-11 for an **hour** entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: **1,2,5,9** and **0-4,8-12**.

Step values can be used in conjunction with ranges. Following a range with **/<number>** specifies skips of the number's value through the range. For example, **0-23/2** can be used in the **hour** field to specify command execution every other hour. Steps are also permitted after an asterisk, so if you want to say every two hours, just use ***/2**.

Names can also be used for the **month** and **day of week** fields. Use at least the first three letters of the particular day or month (case doesn't matter) (e.g., Mon, Jan, etc.) Ranges or lists of names are allowed (e.g., Mon-Wed).

The rest of the line following the **day of week** field, if present, specifies the command to be saved in the **%nextcmd** script file variable.

Note

The day of a command's execution can be specified by two fields -- **day of month** and **day of week**. If both fields are

restricted (i.e., aren't *), a match occurs when either field matches the current day. For example, **30 4 1,15 * 5** would cause a match at 4:30AM on the 1st and 15th of each month, plus every Friday.

Consider the following examples.

```
# run a script 5 minutes after midnight, every day
5 0 * * *          call "daily.s"

# send a file at 2:15pm on the first of every month
15 14 1 * *        sendfile "data.txt"

# resume script processing at 10PM on weekdays
0 22 * * 1-5

# resume at 23 minutes after midnight, 2AM, 4AM ..., everyday
23 0-23/2 * * *

# run a script at 5 minutes after 4AM every Sunday
5 4 * * sun        call "Sunday.s"

# run a script every 2 hours every Sunday
0 */2 * * 0        call "Sunday.s"
```

Multiple events may be specified in the "crontab.txt" file to trigger at the different times as shown below:

```
# run a script at 5PM and 11PM everyday
0 17 * * *          call "Daily5PM.s"
0 23 * * *          call "Daily11PM.s"
```

Beware that it is possible for different events in the "crontab.txt" file to trigger at the same time as shown below:

```
# run a script at 5PM everyday
0 17 * * *          call "Daily.s"
# run a script every hour on Sunday
0 * * * sun         call "AllDaySunday.s"
```

In this example, both events are scheduled to trigger at 5PM on Sunday. You probably want to avoid this situation since only the first match event (i.e., running the script "Daily.s") will be acted upon. In this example, the "AllDaySunday.s" script will never be run at 5PM on Sunday.

Refer to the [Using the FileLink CronMaker Utility](#), [Using The %nextcmd Variable](#) and [CRON](#) for more details.

Using COM/OLE to Control FileLink

It is possible to control FileLink from your own custom application written in C++, Visual Basic, VBScript, or other COM/OLE enabled programming language. Anything a script can do -- and much more -- is possible under programmatic control via FileLink's COM/OLE interface.

FileLink is capable of significantly more complex file transfer sessions by having a custom application program written in a COM/OLE programmatic interface (or [API](#)) enabled programming language.

Complete [sample](#) Visual C++ and Visual Basic projects are included within FileLink package to give you an example of how COM/OLE programming is done.

A simple VBScript sample program may be seen [here](#).

COM/OLE Operational Overview

Under normal operation, FileLink reads a script command from a designated file, performs the command, sets a result code that the script may interrogate, and then FileLink reads the next command from the file. The script can make decisions based on the value of the result code and therefore control the course of a file transfer session.

This works great. But as robust as the FileLink script language may be, it still falls short of what can be done using a true programming language like C++ and Visual Basic.

With FileLink in a COM/OLE environment, there is no script file per se. The application program launches FileLink, establishes a named session, and then sends script commands by way of the COM/OLE interface. FileLink performs a command and returns the result code to the application. (The application may block and get the result code when FileLink completes the command, or it may issue the command and continue, and receive the result via an event call. There is never more than a single command pending at one time.)

Upon return, the application program interrogates the result code and determines what to do next. As necessary, more commands are directed to FileLink until the file transfer session is complete -- the session is complete according to the application program's criteria, not FileLink's. At that point, the application closes the session and FileLink terminates.

One of the options when creating a session with FileLink is to keep it completely hidden from view so end users see the user-written application, not FileLink. Each session is uniquely named, so it is possible to invoke multiple, independent FileLink sessions.

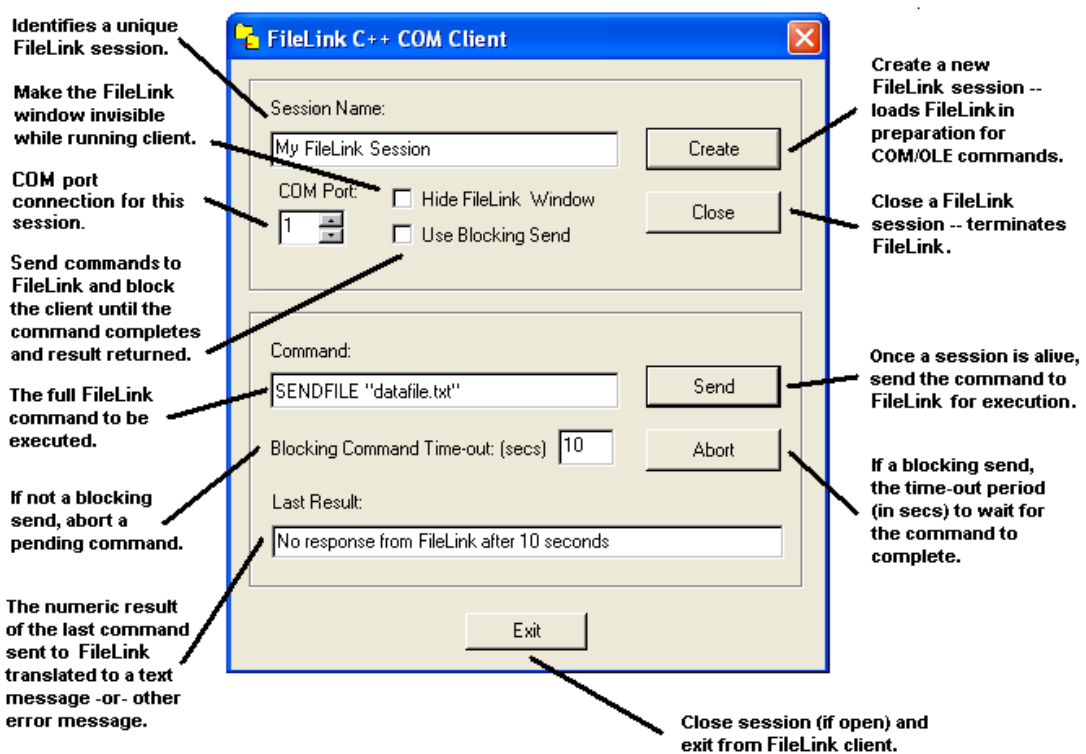
Sample C++ and Visual Basic Project Files

Two fully functional sample programs are provided with FileLink -- one written in Visual C++ and the other in Visual Basic. All source code and project files are included. Controlling applications written in other languages (e.g. VBScript and C#) are possible as long as the languages support the COM/OLE interface.

These programs demonstrate how script commands are sent to FileLink and how result codes are returned, and may be used as a guide for the creation of your own application. By studying the source code for these sample programs, you can learn how to write your application.

The sample programs can be used to exercise limited interactive control over FileLink. Each program allows you to enter script commands from the keyboard and have them executed by FileLink. The results of each command are returned and displayed. Assuming that you do not elect to hide FileLink, you can watch commands in the FileLink window.

When you run the C++ sample, you'll see something that resembles the following image. The C++ and VB programs are functionally equivalent, so the VB sample is virtually identical.



Caution

These sample programs are not intended for production use and you may experience difficulties if you attempt to use them in such a manner.

Note

The time-out parameter specified on a blocking send command call must be long enough for FileLink to complete the requested command. For example, if you ask FileLink to send a large file and the time-out is too short, the sample program will time out before FileLink actually completes the transfer. You may set the time-out to zero to prevent the sample program from timing out.

FileLink COM/OLE Interface Description - A Programmer's View

The COM/OLE methods and events used by an application program to control FileLink are described below.

There are six methods and two events in the FileLink COM/OLE interface.

Methods

- FLStartSession
- FLEndSession
- FLSendCommand
- FLStopCommand
- FLGetVariable
- FLGetVBSVariable

Event

- FLCommandProgress
- FLCommandResult

These methods and event are described in the following sections.

FLStartSession -- Method to initiate a FileLink session

Overview

Method to initiate a named session between FileLink and a user-written application.

C++ Definition

```
long FLStartSession( CString strSessionName, long nComPort, BOOL bBlocking, BOOL  
bHidden, CString strCmdLineArgs )
```

VB Definition

```
FLStartSession( SessionName As String, ComPort As Long, Blocking As Long, Hidden As  
Long, Args As String ) As Long
```

Return Value

The numeric result code returned to indicate the success or failure in initiating a named session with FileLink.

Returns FL_ERROR_SUCCESS if a named session is successfully started.

Returns FL_ERROR_SESSION_EXISTS if a session with the same name already exists.

Returns FL_ERROR_CANNOT_CREATE if a session with FileLink cannot be created.

See [COM/OLE Return Codes](#) below for a complete list of possible return values.

Parameters

strSessionName / SessionName

A string defining the session name. Each unique session loads a separate instance of FileLink.

nComPort / ComPort

A numeric value that specifies which port (1 - 48, where 1 - 48 correspond to COM1 - COM48) to associate with the new session.

bBlocking / Blocking

A boolean that when TRUE results in subsequent [FLSendCommand](#) method calls made during the session being blocking calls. If FALSE, each FLSendCommand is non-blocking.

bHidden / Hidden

A boolean that when TRUE results in FileLink being loaded and run invisibly. If FALSE, the FileLink window appears as it would normally.

strCmdLineArgs / Args

A string defining optional command line arguments to pass FileLink when it is launched.

Remarks

This method returns when FileLink has been launched and the session name is established.

FLEndSession -- Method to terminate a FileLink session

Overview

Method to terminate a named session between FileLink and a user-written application.

C++ Definition

```
long FLEndSession( )
```

VB Definition

```
FLEndSession() As Long
```

Return Value

The numeric result code indicating the success or failure in terminating a session with FileLink.

Returns FL_ERROR_SUCCESS if a session is successfully terminated.

Returns FL_ERROR_VAR_NOT_FOUND if the variable is not defined.

See [COM/OLE Return Codes](#) for a complete list of possible return values.

Parameters

None.

Remarks

This returns after FileLink terminates.

FLSendCommand -- Method to send a script command to FileLink

Overview

Method to send a script command.

C++ Definition

```
long FLSendCommand( CString strCommand, long nTimeout )
```

VB Definition

```
FLSendCommand( Command As String, Timeout As Long ) As Long
```

Return Value

The numeric [result code](#) returned by FileLink after the execution of a script command.

Returns `>= FL_ERROR_FROM_FILELINK` if the command was executed successfully by FileLink. This value corresponds to one of FileLink's [script command result codes](#).

Returns `FL_ERROR_CMD_IN_PROGRESS` if the command is non-blocking and the command has been successfully initiated.

Returns `FL_ERROR_TIMED_OUT` if the command timed out.

See [COM/OLE Return Codes](#) below for a complete list of possible return values.

Parameters

`strCommand` / Command

A string defining the FileLink script command to perform.

To have FileLink simulate line numbers in the log file, precede the command with `#{number}` and a space character.

`nTimeout` / Timeout

A long specifying the period of time (10th of seconds) to wait for a blocking `Send()` to complete. The parameter is ignored when the `Send()` is non-blocking.

Remarks

The `FLStartSession()` method must be called prior to calling `Send()`.

This method blocks or returns immediately depending the selection made in the `FLStartSession()` method.

If non-blocking is selected, the `FLCommandResult()` event is fired upon completion.

If you send a conditional command (e.g., [IFFILE](#) or [IFERROR](#)) the conditional element of the command is evaluated and a TRUE/FALSE result is passed back but no actual branching

action is taken.

For example, if the following script command is sent to FileLink and the file exists FileLink returns \$ERROR_OLE_COMPARISON_TRUE; if file does not exist, the return value is \$ERROR_OLE_COMPARISON_FALSE. The `goto` portion of the command is ignored since FileLink really does not have a script file to branch within.

```
IFFILE "c:\Program Files\FileLink\thisfile" goto found_it
```

If [script labels](#) are sent, they are ignored.

FLStopCommand -- Method to stop a running FileLink script command

Overview

Method to cancel a pending script command from previous non-blocking FLSendCommand().

C++ Definition

```
long FLStopCommand( )
```

VB Definition

```
FLStopCommand() As Long
```

Return Value

A numeric value indicating the success or failure of initiating the cancellation of a FileLink script command.

Returns FL_ERROR_SUCCESS if a stop is initiated -- this does not mean the command has been stopped, rather only that the request has been successfully passed to FileLink

Returns FL_ERROR_NO_CMD_PENDING if no script command is currently in progress that can be stopped.

See [COM/OLE Return Codes](#) for a complete list of possible return values.

Parameters

None.

Remarks

The FLStartSession() and FLSendCommand() methods must be called prior to calling FLStopCommand().

This method returns immediately. The FileLink command may or may not terminate depending on when this method is called. The effect of calling this method is same as pressing the Esc key or clicking the Stop button when FileLink is running interactively. If the command terminates, the FLSendCommand() completion event is fired with the corresponding FileLink result code.

FLGetVariable -- Method to get the value of a FileLink script variable

Overview

Method to get the value of the specified FileLink script variable. Note: this method does not work when the calling program is written in VBScript - use the [FLGetVBSVariable](#) method instead.

C++ Definition

```
long FLGetVariable( CString strVariable, CString strValue )
```

VB Definition

```
FLGetVariable( Variable As String, Value As String ) As Long
```

Return Value

A numeric value indicating the success or failure of obtaining the value FileLink variable.

Returns FL_ERROR_SUCCESS if the variable is found and its value is returned in the *strValue / Value* variable.

Returns FL_ERROR_VAR_NOT_FOUND if the variable is not defined.

See [COM/OLE Return Codes](#) for a complete list of possible return values.

Parameters

strVariable / Variable

A string defining the FileLink script variable name.

strValue / Value

A string that contains the value of the variable upon return.

Remarks

The StartSession() method must be called prior to calling FLGetVariable().

FLGetVBSVariable -- VBS method to get the value of FileLink script variable

Overview

Method used in VBScript to get the value of the specified FileLink script variable. This method may also be in VB programs instead of [FLGetVariable](#) if you wish.

VB Definition

```
FLGetVBSVariable( Variable As String ) As String
```

Return Value

A string containing the value of the specified FileLink variable. An [error is thrown](#) if the variable is not defined or if FileLink is unable to process the request.

Parameters

Variable

A string defining the FileLink script variable name.

Remarks

The StartSession() method must be called prior to calling FLGetVBSVariable().

FLCommandProgress -- Event fired to update SENDFILE/RCVFILE progress

Overview

An event fired when a SENDFILE or RCVFILE script command sent on a non-blocking FLSendCommand() updates the percent sent or received of the current file transfer.

C++ Definition

```
void FLCommandProgress( long nPercent)
```

VB Definition

```
FLCommandProgress (ByVal Percent As Long)
```

Return Value

None.

Parameters

nPercent / *Percent*

A long in the range of 0 to 100 returned by FileLink at random intervals indicating the progress of a SENDFILE or RCVFILE script command.

Remarks

This event is only fired when a non-blocking FLSendCommand() method is called and a SENDFILE or RCVFILE script command has been sent.

FLCommandResult -- Event fired at the conclusion of a non-blocking command

Overview

An event fired when a script command sent on a non-blocking FLSendCommand() completes.

C++ Definition

```
void FLCommandResult( long nResultCode )
```

VB Definition

```
FLCommandResult(ByVal ResultCode As Long)
```

Return Value

None.

Parameters

nResultCode / ResultCode

A long returned by FileLink after the execution or cancellation of a script command that equals one of the FileLink result codes.

Remarks

This event is only fired when a non-blocking FLSendCommand() method is called. The event is fired when the command completes normally or after a command is stopped by calling FLStopCommand().

FLLogMsgs -- Event fired to provide script log information

Overview

An event fired whenever a line of text added to the [FileLink console window](#). Process this event if you wish to implement something similar to FileLink's console window in your application. The following are example lines of text delivered via this event:

*Logon in progress...

*Logon successful.

C++ Definition

```
void FLLogMsgs( BSTR bstrLogMessage)
```

VB Definition

```
FLLogMsgs(ByVal LogMessage As String)
```

Return Value

None.

Parameters

bstrLogMessage / LogMessage

A NULL terminated string containing one line of log text.

Remarks

The string arriving with this event will typically be less than 80 characters in length. Your handler for this event should be as brief as possible since FileLink is suspended until control is returned - typically your handler should save the contents of the delivered string as appropriate and return immediately.

COM/OLE Return Codes

Values returned by automation methods:

<u>Constant Value</u>	<u>Value</u>	<u>Description</u>
FL_ERROR_FROM_FILELINK	0	Any return code greater than or equal to this value comes from FileLink and not from the automation component.
FL_ERROR_SUCCESS	-1	The requested operation completed successfully.
FL_ERROR_NO_SESSION	-2	A FileLink session has not been established by calling FLStartSession().
FL_ERROR_NO_RESPONSE	-3	An internal error indicating that a command sent to FileLink has not returned properly from a COM/OLE call. Make sure that there are no instances of FileLink running and retry. A Windows reboot may be necessary..
FL_ERROR_SESSION_EXISTS	-4	A previous session with the same name already exists.
FL_ERROR_TIMED_OUT	-5	The command sent in a blocking FLSendCommand() call has failed to complete in the allotted time-out period.
FL_ERROR_NO_CMD_PENDING	-6	No command is pending that can be stopped on a call to FLStopCommand().
FL_ERROR_CMD_IN_PROGRESS	-7	A script command sent on a non-blocking FLSendCommand() call is in progress.
FL_ERROR_VAR_NOT_FOUND	-8	The specified FileLink script variable specified in a FLGetVariable() call is not defined.
FL_ERROR_CANNOT_CREATE	-9	An internal error indicating that a named FileLink session cannot be created. Make sure that there are no instances of FileLink running and retry. A Windows reboot may be necessary.
FL_ERROR_INTERNAL_FAILURE	-10	An internal error indicating that an internal Windows error has occurred. A Windows reboot may be necessary.

Sample VBScript Program

FileLink may be called from VBScript (VBS) programs. The following is an example VBS program that loads FileLink, dials the modem to connect to a remote system, retrieves and displays the value of an internal FileLink variable, and then disconnects.

```
' Instantiate a FileLink object
' -----
Set FLReq = Createobject("FLAutomation.Automate")
If err.number <> 0 then
Wscript.quit 1
End If
' Set connection properties and open connection
' -----
ResultCode = FLReq.FLStartSession( "Sample", 1, True, False, "" )
If ResultCode <> -1 then msgbox "FileLink Connection Error
occured!"
FLReq.FLEndSession
WScript.quit 2
End If
' dial the remote system
' -----
cmdString = "DIAL '555-1212'"
ResultCode = FLReq.FLSendCommand( cmdString, 600)
' get the last error
' -----
msgbox FLReq.FLGetVBSVariable("%lasterror" )

' disconnect from remote system
' -----
cmdString = "DISCONNECT"
ResultCode = FLReq.FLSendCommand( cmdString, 600)
' end FileLink session and exit
' -----
FLReq.FLEndSession
Wscript.Quit 0
```

Using Script File Result Codes

Each FileLink script command returns a four-digit numeric result code when it completes to indicate success or failure. When FileLink is used interactively, or if you need to interpret a log file, it is not necessary to know the specific numeric values because FileLink translates all of these values to English phrases in the form of status messages. When writing these messages to the log file, FileLink includes the numeric value in brackets at the end of the message to assist you in building error recovery into your script files.

When writing a script file you may want to take advantage of the [IFERROR](#) command to test for specific result codes. In this case, use these numeric values, or the corresponding

predefined \$ERROR_xxx variable (see below), in the IFERROR command to test for a specific result.

FileLink always returns a zero result code if a script command is completed successfully. Therefore a script file checks for a non-zero result codes to determine if an operation failed. If you choose to check for specific result codes when a command fails, you have more flexibility in recovering from errors.

FileLink defines a set of internal variables referred to as \$ERROR variables. \$ERROR variables allow you to use descriptive variable names rather than a raw numeric result code value when using the IFERROR command. This results in easier to understand script files.

The FileLink \$ERROR variables and corresponding numeric result codes are listed below:

<u>\$ERROR Variable Name</u>	<u>Code</u>	<u>Description</u>
\$ERROR_SUCCESS	0	No error occurred
\$ERROR_INVALID_CMD_LINE	1001	Invalid Shortcut Target command line
\$ERROR_FILE_OPEN_ERROR	1003	Cannot open file
\$ERROR_NO_MORE_VARS	1008	Too many variables assigned
\$ERROR_VAR_NOT_FOUND	1009	Variable not found
\$ERROR_VAR_INVALID	1010	Variable name invalid
\$ERROR_NO_WILD_CARDS	1011	File name wildcard characters not allowed
\$ERROR_COMMAND_INVALID	1012	Missing, invalid, or unrecognized command
\$ERROR_PROMPT_CANCELLED	1013	Cancel button clicked in Prompt dialog
\$ERROR_INVALID_FILE_NAME	1014	Invalid file/path name
\$ERROR_COPY_CANCELLED	1018	File copy canceled
\$ERROR_FILE_NOT_RENAMED	1020	File could not be renamed
\$ERROR_FILE_NOT_DELETED	1021	File could not be deleted
\$ERROR_FILE_NOT_COPIED	1022	File could not be copied
\$ERROR_IS_CONNECTED	1024	Line is already connected
\$ERROR_NOT_CONNECTED	1025	Line not connected
\$ERROR_NO_MODEM_RESP	1027	Modem not responding
\$ERROR_NO_CARRIER	1029	Connect error -- no carrier tone detected
\$ERROR_CONNECT_TIMEOUT	1030	Connect time-out expired (this may also occur on dial or answer commands)
\$ERROR_INVALID_MODEM_CMD	1031	Invalid/unrecognized modem response
\$ERROR_BUSY_SIGNAL	1034	Line is busy
\$ERROR_NO_DIAL_TONE	1035	Connect error -- no dial tone detected
\$ERROR_ANSWER_ERROR	1039	Unable to answer
\$ERROR_CONN_CANCELLED	1042	Connect canceled
\$ERROR_SEND_FILE_ERROR	1046	File read error, canceling transmission
\$ERROR_XMT_ERROR	1050	File transmission error
\$ERROR_ASCII_SEND_FILE_CANCEL	1051	ASCII send file operation canceled
\$ERROR_NO_FILES_FOUND	1053	No files found matching wildcard

		pattern
\$ERROR_SEND_FILE_CANCELLED	1054	Transmission canceled
\$ERROR_FILE_NOT_FOUND	1055	File not found
\$ERROR_LINE_DROPPED	1057	Line dropped, disconnecting...
\$ERROR_FILE_RCV_ERROR	1059	File receive operation failed
\$ERROR_WRITE_ERROR	1063	File write error
\$ERROR_READ_ERROR	1069	File read error
\$ERROR_RCV_ERROR	1070	Unable to receive characters
\$ERROR_RCV_CANCELLED	1073	Receive canceled
\$ERROR_AUTO_NAME_FAIL	1074	File auto-naming failed
\$ERROR_MALLOC_FAILURE	1076	No buffers available
\$ERROR_RCV_FILE_CANCELLED	1077	Receive canceled
\$ERROR_FILE_NAME_REQUIRED	1078	Command requires receive file name
\$ERROR_PTR_FAILURE	1080	Printing failed
\$ERROR_NOT_XE_VERSION	1082	FileLink XE is required
\$ERROR_PTR_CANCELLED	1083	Printing canceled
\$ERROR_ASCII_RCV_FILE_CANCEL	1087	ASCII receive file operation canceled
\$ERROR_CANNOT_RUN_SCRIPT	1088	Cannot run script (Terminal is active)
\$ERROR_LOG_FILE_ERROR	1089	Cannot open script log file
\$ERROR_LOG_NOT_OPEN	1092	Script log file not open
\$ERROR_CANNOT_OPEN_SCRIPT	1096	Cannot open script file
\$ERROR_SCRIPT_EOF	1097	Past end-of-file on script file
\$ERROR_UNKNOWN_COMMAND	1099	Unknown/undefined script command
\$ERROR_INVALID_ARGUMENT	1100	Invalid argument
\$ERROR_INVALID_LABEL	1101	Invalid label
\$ERROR_LABEL_NOT_FOUND	1102	Label not found
\$ERROR_TOO_MANY_LABELS	1103	Maximum number of labels exceeded
\$ERROR_DUPLICATE_LABEL	1104	Duplicate label found
\$ERROR_MAX_FIELDS	1105	More than 10 arguments not allowed
\$ERROR_FILE_POS_ERROR	1106	Cannot position script file
\$ERROR_WAIT_TIMED_OUT	1109	Time-out expired
\$ERROR_WORD_LENGTH_BAD	1110	Selected protocol requires 8-bit data byte
\$ERROR_CHAIN_FAILED	1113	Script file chain command failed
\$ERROR_SCRIPT_READ_ERROR	1116	Error reading script file
\$ERROR_NO_REGISTRY	1120	Registry values not found for COMx
\$ERROR_HW_NOT_CFGD	1121	Hardware error or COM port not found
\$ERROR_EXEC_FAILED	1123	EXEC command failed
\$ERROR_PIPE_FAILED	1125	PIPE command failed
\$ERROR_PIPE_OPEN_TIMEOUT	1127	Pipe open timed out
\$ERROR_PIPE_READ_FAILED	1128	Pipe read failure
\$ERROR_PIPE_READ_TIMEOUT	1129	Pipe read timed out
\$ERROR_PIPE_WRITE_FAILED	1130	Pipe write failure
\$ERROR_PIPE_WRITE_TIMEOUT	1131	Pipe write timed out
\$ERROR_INVALID_PIPE_PROTOCOL	1132	Bad pipe protocol - ignored
\$ERROR_NO_TIMERS_AVAILABLE	1134	Cannot time-out /drop option
\$ERROR_OPTIMEDOUT	1139	Operation timed out
\$ERROR_PIPE_NOT_CREATED	1140	Pipe not previously created

\$ERROR_PIPE_ALREADY_EXISTS	1141	Pipe already exists
\$ERROR_TRACE_LOG_NOT_OPEN	1145	Trace log file not currently open
\$ERROR_TRACE_LOG_ERROR	1146	Error writing to trace log file
\$ERROR_EVENT_LOGGING_ERROR	1147	Error writing to NT event log
\$ERROR_NO_FILE_FOUND	1164	No file found
\$ERROR_NO_DIR_ACCESS	1167	Cannot access/create local folder
\$ERROR_NO_RELATIVE_PATHS	1168	Relative pathnames not allowed here
\$ERROR_RAS_NOT_INSTALLED	1170	Dial-up networking not installed
\$ERROR_RAS_CONNECTION_FAILED	1172	Dial-up networking connection failed
\$ERROR_AUTHORIZATION_FAILED	1175	Authorization not found
\$ERROR_INVALID_MINIMIZED	1176	Script command not permitted when main window is minimized or when running as NT service
\$ERROR_NO_MODEMS_DETECTED	1177	No modems detected in system
\$ERROR_NO_PORTS_DETECTED	1178	No COM ports detected in system
\$ERROR_THREAD_ERROR	1180	Internal thread launch failure
\$ERROR_NO_ACTIVITY_TIMEOUT	1181	No activity time-out expired during file send or receive
\$ERROR_INVALID_FUNCTION_NAME	1185	Invalid function name
\$ERROR_FUNCTION_FILE_ERROR	1186	Function file creation error.
\$ERROR_TOO_MANY_ARGUMENTS	1189	Too many arguments passed to function
\$ERROR_BAD_ARGUMENT_LIST	1190	Number of function arguments do not match declaration.
\$ERROR_TOO_MANY_FUNCTIONS	1192	Too many functions defined or nesting calls too deep
\$ERROR_FUNCTIONS_NOT_RECURSIVE	1193	Functions may not be called recursively
\$ERROR_EMAIL_CANNOT_CREATE	1200	Cannot create e-mail message
\$ERROR_EMAIL_CANNOT_SEND	1201	Cannot send e-mail message
\$ERROR_EMAIL_CANNOT_GET	1202	Cannot get e-mail message
\$ERROR_EMAIL_NO_MESSAGES	1203	No messages on e-mail server
\$ERROR_CRONTAB_EMPTY	1210	crontab.txt file contains no scheduling condition(s)
\$ERROR_SRVMONITOR_FAILED	1210	Unable to launch SrvMonitor
\$ERROR_SRVR_FILE_EMPTY	1229	File exists on server but its length is 0
\$ERROR_INVALID_COMPARISON	1230	Invalid file statistics comparison
\$ERROR_FILE_INFO_UNAVAIL	1231	Local or server file data unavailable
\$ERROR_LOCAL_FILE_NEWER	1232	Local file is newer
\$ERROR_LOCAL_FILE_OLDER	1233	Local file is older
\$ERROR_FILES_SAME_SIZE	1234	Local and server files identical in size
\$ERROR_LOCAL_FILE_LARGER	1235	Local file is larger
\$ERROR_LOCAL_FILE_SMALLER	1236	Local file is smaller
\$ERROR_LOCAL_FILE_LARGERREQ	1237	Local file same size or larger
\$ERROR_LOCAL_FILE_EXISTS	1238	Local file exists
\$ERROR_NO_LOCAL_FILE_EXISTS	1239	Local file does not exist
\$ERROR_LOCAL_FILE_EMPTY	1240	Local file exists but length equals 0
\$ERROR_FILES_SAME_DATETIME	1241	Local and server files have same date and time

\$ERROR_OLE_COMPARISON_TRUE	1250	Comparison results is TRUE
\$ERROR_OLE_COMPARISON_FALSE	1251	Comparison results is FALSE
\$ERROR_OLE_CMD_ACCEPTED	1252	Statement or label accepted
\$ERROR_ZIP_DLL_MISSING	1255	Zip library file is missing
\$ERROR_ZIP_FILE_CREATE	1256	Error updating or creating zip file
\$ERROR_ZIP_FILE_CANCELED	1257	Zip file operation canceled
\$ERROR_ZIP_FILE_EXTRACT	1258	SSH/SSL components not installed
\$ERROR_PGP_NOT_INSTALLED	1270	PGP components not available; this is an installation issue, verify that the PGP components were chosen
\$ERROR_PGP_KEYID_MISSING	1271	User name, comment, and/or e-mail address required to identify recipient of encrypted file
\$ERROR_PGP_UNACCEPTABLE_FILE	1272	Cannot import PGP keys from this file
\$ERROR_PGP_UNRECOGNIZED_FILE	1273	Key file is not in a recognized format
\$ERROR_PGP_KEY_EXISTS	1274	Key already exists in your keyring
\$ERROR_PGP_IMPORT_FAILED	1275	PGP import operation failed
\$ERROR_PGP_NO_KEYRING_FILE	1276	PGP keyring file cannot be created
\$ERROR_PGP_ENCRYPTION_FAILED	1277	PGP encryption operation failed or canceled
\$ERROR_PGP_DECRYPTION_FAILED	1278	PGP decryption operation failed or canceled
\$ERROR_PGP_INITIALIZATION_FAILED	1279	PGP component failed to initialize
\$ERROR_PGP_KEY_DOES_NOT_EXIST	1280	Key does not exist in your key ring
\$ERROR_PGP_PASSPHRASE_BLANK	1281	Passphrase not present
\$ERROR_GNUPG_CMD_FAILED	1282	External GnuPG command failed or canceled
\$ERROR_SNAPSHOT_CANCELLED	1290	SNAPSHOT command canceled
\$ERROR_DIFF_CANCELLED	1291	DIFF command canceled
\$ERROR_DB_NO_FILE_OPEN	1300	No user database file currently open
\$ERROR_DB_NO_QUERY_RESULTS	1301	No results available from previous query
\$ERROR_DB_ALL_RESULTS_RTND	1302	All results returned from previous query
\$ERROR_DB_QUERY_FAILED	1303	Query failed
\$ERROR_DB_QUERY_RESULT_NULL	1304	Query result empty
\$ERROR_DB_RAW_QUERY_RESULTS	1305	Query result processing failed, raw result saved
\$ERROR_LICENSE_VIOLATION	2000	Product not licensed or license removed

The follow constants may be used to assist in writing more legible and self-documenting scripts:

Doing File Comparisons

\$FILE_EMPTY	3000	Other file exists but is empty
\$FILE_LARGER	3001	Other file is larger
\$FILE_SMALLER	3002	Other file is smaller
\$FILE_SAME_SIZE	3003	Other file is same size
\$FILE_NEWER	3004	Other file is newer
\$FILE_OLDER	3005	Other file is older

\$FILE_SAME_DATETIME	3006	Other file has same date/time
----------------------	------	-------------------------------

Identifying Difference From DIFF Commands

\$DIFF_FILE_NOT_FOUND	5001	File not found
\$DIFF_FILE_IS_NEW	5002	File is new
\$DIFF_FILE_SIZE	5003	File size has changed
\$DIFF_FILE_DATETIME	5004	File date/time stamp has changed

Index

- % -

%cr variable 99
 using the 99
 %crlf
 using the 99
 %currentlocaldir variable 100
 using the 100
 %date variable
 using the 101
 %datetime variable
 using the 101
 %dbqueryrawresult variable
 using the 102
 %dbqueryrows variable
 using the 102
 %dbqueryvariables variable
 using the 102
 %difffileid variable
 using the 103
 %difffilename variable
 using the 103
 %difffiles variable
 using the 104
 %difffiletext variable
 using the 103
 %diffnum variable
 using the 104
 %ftpsnapshotfiles variable
 using the 114
 %lasterror variable 105
 using the 105
 %lasterrormsg variable 106
 using the 106
 %lastfile variable
 using the 107
 %lastpath variable
 using the 107
 %lf
 using the 99
 %lnewport variable 109
 using the 109

%nextcmd variable
 using the 108
 %nextfile variable
 using the 110
 %nextfiledate variable
 using the 111
 %nextfiledatetime variable
 using the 111
 %nextfilesize variable 111
 using the 111
 %nextfiletime variable 111
 using the 111
 %nextpath variable
 using the 110
 %port variable
 using the 112
 %rcvfilecount variable
 using the 113
 %sendfilecount variable
 using the 113
 %snapshotfiles variable
 using the 114
 %time variable
 using the 101
 %upzipcount variable
 using the 115
 %zipcount variable 115
 using the 115

- / -

/baudrate 275
 /carrier 275
 /flowcontrol 275
 /parity 275
 /stopbits 275
 /wordlength 275

- 8 -

8.3 file naming convention 251

- A -

accepting remote commands 244
 always on top
 displaying the FileLink window 30

- ANSWER 123
- APPEND 124, 251
 - append files
 - on FTP server 251
 - Applets Menu 28
 - archive directory
 - setting the 125
 - ARCHIVEDIR 125
 - arguments
 - script file 75, 76
 - shortcut target 94
 - arithmetic
 - using in variables 89
 - ASCII armoring 43
 - ASCII file transfers 57
 - ASK 126
 - AUTHDATA 128
 - Authorizing Remote Users 95
 - Authorization File Format 96
 - AUTHPW 129
 - AUTHUSER 130
 - auto-answer 123
 - auto-dial 154

- B -

 - baud rate
 - setting the 275
 - BEGINFUNCTIONS 131
 - binary character 198
 - blocking factor 251
 - BREAK 132
 - break state 132
 - BROWSE 133
 - buffer sizes 238

- C -

 - CALL 134
 - call a script file 134
 - carriage control 211
 - CHAIN 135
 - chain to script file 135
 - change directory
 - on local PC 136
 - character I/O 196, 198
 - CHGDIR 136
 - Clipboard 271
 - COM port
 - setting in script language 275
 - COM/OLE event
 - FLCommandProgress 310
 - FLCommandResult 311
 - FLLogMsgs 312
 - COM/OLE interface 298
 - COM/OLE method
 - FLEndSession 304
 - FLGetVariable 308
 - FLGetVBSVariable 309
 - FLSendCommand 305
 - FLStartSession 303
 - FLStopCommand 307
 - COM/OLE overview 299
 - command line
 - arguments 94
 - shortcut target 19
 - switches 19
 - comments in script files 81
 - concatenate two strings 254
 - concatenation 254
 - conditional branching 179, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 193, 205, 206, 207
 - Configuration
 - for Current User vs. All Users 18
 - configuration settings
 - exporting 165
 - importing 194
 - configuring FileLink 49
 - CONNECT 137
 - CONSOLE 138
 - console window 22
 - entering commands into 32
 - output to 138
 - controlling FileLink
 - from a user application 298
 - controlling script log output 91
 - COPY 139
 - copy file 139
 - Courier V.Everything 37, 215
 - create directory
 - on local PC 209
 - CREATEMAIL 140

CRON 141
 CronMaker utility
 using the 290, 292, 293, 294, 295
 Crontab File Format 296

■ ■ ■

-d switch 19

- D -

date arithmetic 90, 144, 145
 DATEADD 144
 DATESUB 145
 DBCLOSE 146
 DBGETRESULTS 147
 DBQUERY 148
 DBREWIND 149
 DBUSE 150
 debugging
 script files 82
 DEC 151
 decrypting
 PGP 225
 default buffer size 238
 default directory
 setting the 277
 DELDIR 152
 DELETE 153
 delete directory
 on local PC 152
 delete file 153
 detect a modem 37
 DIAL 154
 Dial-In Connection With Authorization 284
 dialog box
 open file 133
 prompt in 126, 237
 Dial-Up Connection Performing a Logon 283
 DIFF 155
 DIFFREWIND 156
 digital signatures
 PGP 44
 direct connect 137
 directory
 listing 200

DISCONNECT 158
 disconnect line 158
 DISPLAY 157
 display dialog box 211
 display variable value 157
 DOSCMD 159
 drag and drop 135

- E -

editing script files 72
 editor
 script file 72
 E-mail in Script Files
 receiving 93
 sending 93
 e-mail messages
 creating 140
 getting 171
 sending 208, 253
 encrypting
 PGP 228
 ENDFUNCTION 160
 ENDFUNCTIONS 161
 Entering Commands in the Console Window 32
 EXEC 162
 execute a script command 222
 execute external program 162
 execute internal DOS command 159
 EXIT 164
 exit code 164
 EXPORT 165

- F -

file date and time
 comparing 179, 191
 file existence
 determining 182, 183
 file I/O 242, 278
 file names
 as script arguments 76
 file size
 comparing 188
 file system
 local 155, 264

file system changes
 locating 168

file transfer
 features 15
 protocol configuration 56

file transfer protocols
 define 238

FileLink Configurator
 File Configuration 51
 Hardware Configuration 52
 Modem Configuration 53
 PGP Configuration 63
 Terminal Configuration 54
 Transfer Configuration 55
 User vs. Machine Configuration 67

FileLink Main Window Details 23

fingerprint 223

FLUSH 166

flush receive buffer 166

forced exit 164

FUNCTION 167

function declarations
 BEGINFUNCTIONS 131
 ENDFUNCTION 160
 ENDFUNCTIONS 161
 RETURN 248

functions declarations
 FUNCTION 167

- G -

General Features 14

GETDIFF 168

GETFILE 170

GETMAIL 171

GETNEXTFILE 173

GETREWIND 176

Getting Online Help 31

GnuPG command
 sending a 223

GO 177

GOTO 178

- H -

Help Menu 29

- I -

icon
 running FileLink as 40

IFDATE 179

IFERROR 181

IFERROR!= 181

IFERROR< 181

IFERROR<= 181

IFERROR= 181

IFERROR> 181

IFERROR>= 181

IFFILE 182

IFNFILE 183

IFNO 184

IFNSTRCMP 185

IFNSUBSTR 186

IFNUM 187

IFSIZE 188

IFSUBSTR 190

IFTIME 191

IFYES 193

IMPORT 194

INC 195

- K -

Kermit file transfers 58

key ID 41

- L -

label
 branching to 178

labels in script files 80

LINEIN 196

LINEOUT 198

LISTDIR 200

LOG 201

log file
 script 201
 trace 269
 writing message to 203

LOGMSG 203

LOGNTEVENT 204

long file names 251
 LOOPCOUNT 207
 LOOPIF 205
 looping 205, 206, 207
 LOOPTO 206

- M -

MAILTO 208
 Main File Menu 25
 MAKEDIR 209
 MAKEFILENAME 210
 MESSAGEBOX 211
 MINIMIZE 213
 minimize window 213
 MODEMCMD 214
 MODEMDEFAULTS 215
 MODEMDETECT 216
 MODEMRESET 217
 MODEMRESP 218
 modems
 detecting 38
 factory defaults 37
 used with FileLink 37
 MONITOR 266
 MOVE 219
 move file 219

- - -

-n switch 19

- N -

NATO 220
 newest or oldest file 170, 173
 NT event log
 writing message to 204
 NT service
 installing as 285
 null-modem 137

- O -

options
 in script file 79

- - -

-p switch 19

- P -

passive-mode 15
 passphrase
 PGP 42
 PAUSE 221
 PERFORM 222
 performing a remote logon 280
 PGP encryption 40
 decrypting a file 225
 encrypting a file 228
 importing keys 232
 PGP key
 create 64
 manage 66
 select 65
 PGPCOMMAND 223
 PGPDECRYPT 225
 PGPENCRYPT 228
 PGPIMPORT 232
 PLAYSOUND 233
 PRESSANYKEY 234
 PRINT 235
 print file 235
 programmatic interface 298
 programming interface
 COM/OLE 302
 events 302
 methods 302
 PROMPT 237
 prompt.s 39
 PROTOCOL 238

- Q -

quitting program 164

- R -

RCVFILE 240
 READFILE 242

reading characters from COM port 196
 receiving files
 in script file 240
 REMOTECMD 244
 RENAME 245
 renaming files 245
 RESTORE 246
 restore window from icon 246
 result codes 181, 314
 RESUME 247
 RETURN 248
 Run Minimized box 40
 running as an icon 40, 213
 Running FileLink 16

■ ■ ■

-s switch 19

- S -

sample programs
 COM/OLE 300
 sample script files 280
 scheduling file transfers 35
 scheduling script commands 141, 221
 Scheduling Script Operation 92
 scheduling tasks 141, 221
 Script Commands
 grouped by function 119
 script file
 prompting in 39
 script language
 features 16
 programming 73
 Scripts Menu 27
 SENDCMD 250
 SENDFILE 251
 sending a user ID and a password 280
 sending files
 in script file 251
 SENDMAIL 253
 SET 254
 SETEXTRACT 256
 SETLEFT 257
 SETLEN 258

SETMID 259
 SETNUM 260
 SETRIGHT 262
 SETSUBSTR 263
 shortcuts
 in Start menu 16
 on desktop 16
 Shutting Down FileLink Service 287
 Simple Async Dial-Up Script 281
 Simple Async Dial-Up Script With Error Recovery 282
 SNAPSHOT 264
 SPEAKER 265
 speaker control 265
 SQL database
 "rewinding" query results from a 149
 close a 146
 creating/opening a 150
 getting query results from 147
 issuing a command or query to a 148
 SrvMonitor
 controlling access to 266
 using the 288
 SRVNAME 266
 STOP 267
 stop script execution 267
 string comparison 185, 186, 189, 190
 system date 78
 System Menu 30
 system time 78

- T -

TERMINAL 268
 Terminal applet
 Connection menu 69
 File Transfer menu 71
 Help menu 72
 Settings menu 70
 starting the 268
 using the 68
 text files
 reading 242
 writing to 278
 The FileLink Script File Editor 72
 time delay 221
 time-out

time-out
 no activity 220
 Tools Menu 26
 trace window 271
 TRACELOG 269
 TRACEWIN 271
 tray icon 288
 typing commands 39
 into 32

- U -

unique file name
 creating 210
 unprintable
 sending character(s) 198
 UNZIP 273
 unzipping
 files 273
 USEPORT 275
 User vs. Machine Configuration 67
 Using functions 85
 Using the CronMaker Utility 290
 Using the FileLink TTY Terminal Applet 68

- V -

variable
 arithmetic 89
 decrement 151
 increment 195
 numeric comparing 187
 variables 110
 %cr 99
 %crlf 99
 %currentlocaldir 100
 %dbqueryrawresult 102
 %dbqueryrows 102
 %dbqueryvariables 102
 %difffileid 103
 %difffilename 103
 %difffiles 104
 %difffiletext 103
 %diffnum 104
 %ftpsnapshotfiles 114
 %lasterror 105
 %lasterrormsg 106

%lastfile 107
 %lastpath 107
 %lf 99
 %newport 109
 %nextcmd 108
 %nextfile 110
 %nextfiledate 111
 %nextfiledatetime 111
 %nextfilesize 111
 %nextfiletime 111
 %nextpath 110
 %port 112
 %rcvfilecount 113
 %sendfilecount 113
 %snapshotfiles 114
 %unzipcount 115
 %zipcount 115
 arithmetic 260
 assigning values to 254
 evaluating numeric expressions 260
 numeric 77
 script file 78
 used in command options 83
 VBScript 314
 sample 314
 Visual Basic 300
 Visual C++ 300

- W -

WORKINGDIR 201, 277
 WRITEFILE 278
 writing characters out COM port 198

- X -

Xmodem File Transfers 59
 Xmodem1K File Transfers 60

- Y -

Ymodem File Transfers 61

- Z -

ZIP 279
 zipping

zipping
 files 279
Zmodem File Transfers 62



Serengeti Systems Incorporated
1108 Lavaca Street, Suite 110 PMB 431
Austin, Texas 78701 USA
www.robo-ftp.com
www.serengeti.com