C-KERMIT USER GUIDE

For UNIX, VAX/VMS, and Some Other Operating Systems

Version 5A(180) BETA

Christine M. Gianone and Frank da Cruz

Columbia University Center for Computing Activities
612 West 115th Street
New York, NY 10025 USA

D*R*A*F*T

26 May 1992

1. C-Kermit

Program:	Frank da Cruz (Columbia University), contributions by many others.
Language:	C
Documentation:	Christine Gianone, Frank da Cruz (Columbia University).
Version:	5A(180) BETA
Date:	May 26, 1992

C-Kermit Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
Transfer text files:	Yes
Transfer binary files:	Yes
International text:	Yes
Wildcard send:	Yes
File transfer interruption:	Yes
Filename collision actions:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	Yes
Alternate block checks:	Yes
Automatic parity detection:	Yes
Dynamic packet length:	Yes
CONNECT mode:	Yes
Terminal emulation:	Depends on implementation
Key mapping:	Yes
Communication settings:	Yes
Transmit BREAK:	Yes (most versions)
Support for dialout modems:	Yes
TCP/IP support:	Yes (some versions, Berkeley sockets)
X.25 support:	Yes (SUN versions with SunLink)
IBM mainframe communication:	Yes
Transaction logging:	Yes
Session logging:	Yes
Debug logging:	Yes
Packet logging:	Yes
Act as server:	Yes
Talk to server:	Yes
Advanced server functions:	Yes
Security for server:	Yes
Local file management:	Yes
Command/Init files:	Yes
Long packets:	Yes
Sliding Windows:	Yes
File attributes packets:	Yes
Command macros:	Yes
Script programming language:	Yes
Raw file transmit and capture:	Yes

IMPORTANT NOTICE: The user manual for the current release of C-Kermit is *Using C-Kermit*, by Frank da Cruz and Christine M. Gianone, published by Digital Press, Burlington, MA. Publication date: Fall 1992. After Fall 1992, call Digital Press at 1-800-344-4825 for further information.

1.1. Acknowledgements

Thanks to the dozens of people who have been involved in C-Kermit development over the years, especially Bill Catchings, Jeff Damens and Chris Maio (formerly of Columbia University), and Herm Fischer of Encino, CA, for their roles in its initial development. And special thanks to those who put so much work into features or testing of version 5A:

Kristoffer Eriksson (Peridot Konsult AB, Örebro, Sweden)
Bo Kullmar (Sveriges Riksbank (Central Bank of Sweden), Stockholm)
Tor Lillqvist (Helsinki University, Finland)
Warren H. Tucker (Tridom Corporation, Mountain Park, Georgia, USA)
Peter Mauzey (AT&T Bell Laboratories, Middletown, New Jersey, USA)
Terry Kennedy (St Peters College, Jersey City, NJ, USA)
Chuck Hedrick (Rutgers University, New Brunswick, NJ, USA)
Joe R. Doupnik (Utah State University, USA)
Hirofumi Fujii (Japan National Lab for High Energy Physics, Tokyo)
Ken Yap (University of Rochester, New York, USA)
Marcello Frutig (Catholic University, Rio de Janeiro, Brazil)
Paul Placeway (BBN, Cambridge, Massachusetts, USA)
Mark Buda (Digital Equipment Corporation, New Hampshire, USA)
William Bader (Software Consulting Services, Nazareth, PA, USA)
Steve Walton (California State University Northridge, USA)
Rick Watson (University of Texas at Austin, USA)
Peter Svanberg (Royal Techn. HS, Sweden)
Michel Robitaille (University of Montreal, Canada)
Kai Uwe Rommel (Technische Universität Muenchen, Germany)
Paul Kline (Drake University, USA)
Chris Adie (Edinburgh University, Scotland, UK)
Chris Armstrong (Brookhaven National Laboratory, New York, USA)
James Harvey (Indiana/Purdue University, USA)
Bob Larson (University of Southern California, USA)
Christian Hemsing (RWTH Aachen, Germany)
Bruce J. Moore (Allen-Bradley Co, Highland Heights, OH)

Kristoffer, Bo, Warren, and Peter comprised the "modem committee", which worked long and hard to rationalize C-Kermit's treatment of modem signals in the many and varied UNIX environments. Kristoffer and Bo also contributed much code to the UNIX-specific support modules. Kristoffer deserves a special Archaeology Achievement Award for a detailed study and unraveling of many years' accretions of incomprehensible "magic" in the infamous `ckutio.c` module (a collection of supposedly simple functions that give the emphatic lie to all claims that UNIX is a "portable operating system").

Ken got the TCP/IP support started by supplying the socket-opening code for the Berkeley UNIX version. Marcello supplied the SunLink X.25 code. Paul Placeway, Rick Watson, Peter Svanberg, Paul Kline, and Michel Robitaille contributed enormously to the Macintosh version (and continue to do so). The two Chris's, and more recently Kai-Uwe, for the OS/2 version; Terry, William, James, Mark for the VAX/VMS version; Steve for the Amiga version; Christian for the OS-9 version; Bruce for the Atari ST

version.

Chuck found and fixed many bugs and contributed many valuable insights, as well as writing the UNIX "man page" for C-Kermit 5A.

And thanks to the indefatigable and omniscient Joe Doupnik (author of MS-DOS Kermit), who worked closely with the authors on the design of the sliding window algorithms, the script programming language, and the international character set support during the joint development effort for MS-DOS Kermit 3.0 and C-Kermit 5A (and in his spare time, Joe lent a hand with the AT&T 7300 UNIX PC version of C-Kermit). And finally, thanks to the dozens (hundreds) of others who tested C-Kermit on dozens (hundreds) of hardware and software platforms and contributed bug reports, fixes, new features, and suggestions:

Chris Adie, Edinburgh U, Scotland (OS/2 support)
Robert Adsett, University of Waterloo, Canada
Larry Afrin, Clemson U
Barry Archer, U of Missouri
Robert Andersson, International Systems A/S, Oslo, Norway
Chris Armstrong, Brookhaven National Lab (OS/2)
William Bader, Moravian College
Fuat Baran, CUCCA
Stan Barber, Rice U
Jim Barbour, U of Colorado
Donn Baumgartner, Dell
Nelson Beebe, U of Utah
Karl Berry, UMB
Dean W Bettinger, SUNY
Gary Bilkus
Marc Boucher, U of Montreal
Charles Brooks, EDN
Bob Brown
Mike Brown, Purdue U
Jack Bryans, California State U at Long Beach
Mark Buda, DEC (VAX/VMS)
Bjorn Carlsson, Stockholm University Computer Centre QZ, Sweden
Bill Catchings, formerly of CUCCA
Bob Cattani, Columbia U CS Dept
Davide Cervone, Rochester University
Seth Chaiklin, Denmark
John Chandler, Harvard U / Smithsonian Astronomical Observatory
John L Chmielewski, AT&T, Lisle, IL
Howard Chu, U of Michigan
Bill Coalson, McDonnell Douglas
Bertie Coopersmith, London, UK
Chet Creider, University of Western Ontario, Canada
Alan Crosswell, CUCCA
Jeff Damens, formerly of CUCCA
Mark Davies, Bath U, UK
S. Dezawa, Fujifilm, Japan
Joe R. Doupnik, Utah State U
Frank Dreano (Honeywell)
John Dunlap, University of Washington
David Dyck, John Fluke Mfg Co.
Bernie Eiben, DEC

Kristoffer Eriksson, Peridot Konsult AB, Oerebro, Sweden
John R. Evans, IRS, Kansas City
Glenn Everhart, RCA Labs
Herm Fischer, Encino, CA (extensive contributions to version 4.0)
Carl Fongheiser, CWRU
Marcello Frutig, Catholic University, Sao Paulo, Brazil (X.25 support)
Hirofumi Fujii, Japan Nat'l Lab for High Energy Physics, Tokyo (Kanji)
Chuck Fuller, PSC
Andy Fyfe, Caltech
Christine M. Gianone, CUCCA
John Gilmore, UC Berkeley
German Goldszmidt, IBM
Alistair Gorman, New Zealand
Chris Green, Essex U, UK
Alan Grieg, Dundee Tech, Scotland, UK
Yekta Gursel, MIT
Jim Guyton, Rand Corp
Michael Haertel
Marion Hakanson
John Hamilston, Iowa State U
Simon Hania, Netherlands
Stan Hanks, Rice U.
Ken Harrenstein, SRI
James Harvey, Indiana/Purdue U (VMS)
Rob Healey
Chuck Hedrick, Rutgers U
Ron Heiby, Technical Systems Division, Motorola Computer Group
Steve Hemminger, Tektronix
Christian Hemsing, RWTH Aachen, Germany (OS-9)
Andrew Herbert, Monash Univ, Australia
Mike Hickey, ITI
R E Hill
Bill Homer, Cray Research
Randy Huntziger, National Library of Medicine
Larry Jacobs, Transarc
Steve Jenkins, Lancaster University, UK
Mark B Johnson, Apple Computer
Eric F Jones, AT&T
Luke Jones, AT&T
Peter Jones, U of Quebec Montreal
Phil Julian, SAS Institute
Mic Kaczmarczik, U of Texas at Austin
Sergey Kartashoff, Inst. of Precise Mechanics & Computer Equipment, Moscow
Howie Kaye, CUCCA
Rob Kedoin, Linotype Co, Hauppauge, NY (OS/2)
Mark Kennedy, IBM
Terry Kennedy, St Peter's College, Jersey City, NJ (VAX/VMS, 2.11 BSD)
Douglas Kingston, morgan.com
Tom Kloos, Sequent Computer Systems
Jim Knutson, U of Texas at Austin
David Kricker, Encore Computer
Thomas Krueger, UWM
Bo Kullmar, Central Bank of Sweden, Kista
John Kunze, UC Berkeley

Bob Larson, USC (OS-9)
Bert Laverman, Groningen U, Netherlands
Steve Layton
David Lawyer, UC Irvine
S.O. Lidie, Lehigh U
Tor Lillqvist, Helsinki University, Finland
Dean Long
Kevin Lowey, U of Saskatchewan (OS/2)
Andy Lowry, Columbia University
David MacKenzie, Environmental Defense Fund, University of Maryland
John Mackin, University of Sidney, Australia
Martin Maclaren, Bath U, UK
Chris Maio, Columbia U CS Dept
Fulvio Marino, Olivetti, Ivrea, Italy
Peter Mauzey, AT&T
Tye McQueen, Utah State U
Ted Medin
Hellmuth Michaelis
Leslie Mikesell, American Farm Bureau
Martin Minow, DEC (VAX/VMS)
Pawan Mistra, Bellcore
Ken Mizialko, IBM, Manassas, VA
Ray Moody, Purdue U
Bruce J Moore, Allen-Bradley Co, Highland Heights, OH (Atari ST)
Steve Morley, Convex
Peter Mossel, Columbia University
Tony Movshon, NYU
Lou Muccioli, Swanson Analysis Systems
Dan Murphy
Gary Mussar
John Nall, FSU
Jack Nelson, University of Pittsburgh
Jim Noble, Planning Research Corporation (Macintosh)
Ian O’Brien, Bath U, UK
John Owens
André Pirard, University of Liege, Belgium
Paul Placeway, Ohio State U (Macintosh & more)
Piet W. Plomp, ICCE, Groningen University, Netherlands
Ken Poulton, HP Labs
Manfred Prange, Oakland U
Frank Prindle, NADC
Tony Querubin, U of Hawaii
Anton Rang
Scott Ribe
Alan Robiette, Oxford University, UK
Michel Robitaille, U of Montreal (Mac)
Kai Uwe Rommel, Technische Universität München (OS/2)
Larry Rosenman (Amiga)
Jay Rouman, U of Michigan
Jack Rouse, SAS Institute (Data General and/or Apollo)
Stew Rubenstein, Harvard U (VAX/VMS)
Bill Schilit, Columbia University
Eric Schnoebelen, Convex
Benn Schreiber, DEC

Dan Schullman, DEC (modems, DIAL command, etc)
John Schultz, 3M
Steven Schultz, Contel (PDP-11)
APPP Scorer, Leeds Polytechnic, UK
Gordon Scott, Micro Focus, Newbury UK
Gisbert W. Selke, WIdO, Bonn, Germany
David Sizeland, U of London Medical School
Fridrik Skulason, Iceland
Dave Slate
Bradley Smith, UCLA
Richard S Smith, Cal State
Ryan Stanisfer, UNT
Bertil Stenstroem, Stockholm University Computer Centre (QZ), Sweden
Peter Svanberg, Royal Techn. HS, Sweden
Andy Tanenbaum, Vrije U, Amsterdam, Netherlands
Markku Toijala, Helsinki U of Technology
Rick Troxel, NIH
Warren Tucker, Tridom Corp, Mountain Park, GA
Dave Tweten, AMES-NAS
G Uddeborg, Sweden
Walter Underwood, Ford Aerospace
Pieter Van Der Linden, Centre Mondial, Paris
Ge van Geldorp, Netherlands
Fred van Kempen, MINIX User Group, Voorhout, Netherlands
Wayne Van Pelt, GE/CRD
Mark Vasoll, Oklahoma State U (V7 UNIX)
Konstantin Vinogradov, ICSTI, Moscow
Paul Vixie, DEC
Dimitri Vulis, CUNY
Roger Wallace, Raytheon
Stephen Walton, Calif State U, Northridge (Amiga)
Jamie Watson, Adasoft, Switzerland (RS/6000)
Rick Watson, U of Texas (Macintosh)
Robert Weiner
Lauren Weinstein
Joachim Wiesel, U of Karlsruhe, Germany
Michael Williams, UCLA
Nate Williams, U of Montana
David Wilson
Patrick Wolfe, Kuck & Associates, Inc.
Gregg Wonderly, Oklahoma State U (V7 UNIX)
Farrell Woods, Concurrent (formerly Masscomp)
Dave Woolley, CAP Communication Systems, London
Jack Woolley, SCT Corp
Frank Wortner
Ken Yap, U of Rochester
John Zeeff, Ann Arbor, MI

1.2. Introduction

C-Kermit is a communication software program written in C that provides a full implementation of the Kermit file transfer protocol, plus terminal connection, modem dialing, file management, and a powerful script programming language. An important goal of C-Kermit is transportability to different computers and operating systems. To date, C-Kermit (one release or another) has been adapted to UNIX (most versions), VAX/VMS, Data General AOS/VS, IBM OS/2, the Apple Macintosh, the Commodore Amiga, the Atari ST, Apollo Aegis, OS-9, NCR 9800/VE, and others.

C-Kermit offers you several methods of interaction. On UNIX, VAX/VMS, and most other systems, you have access to Kermit's interactive command parser, and you can also invoke Kermit with command line options. On workstations like the Macintosh, there is a mouse-and-window interface with pull-down menus, radio buttons, and so forth. In some cases, you have both.

Interactive operation gives you full access to all of C-Kermit's features. Command-line invocation gives you access to a small but useful subset. Command-line operation is described in Section 1.15.

The main part of document concentrates on the general features of C-Kermit. Sections specific to certain systems -- UNIX, VAX/VMS, OS/2, the Amiga, OS-9 -- appear at the end. Material about some of the implementations remains to be added. Macintosh Kermit will be documented in a separate publication.

PLEASE NOTE: This manual describes a full-featured version of C-Kermit. However, some computers do not have address spaces (or C compilers, or linkers) capable of supporting all Kermit's features, so some features might be missing from your version. See the file `ckuins.doc` for details.

1.3. Kermit Software for Other Computers

Kermit software is available from Columbia University for almost every known computer and operating system, written in a wide variety languages. For a catalog of Kermit software and ordering information, write to:

Kermit Distribution
Columbia University Center for Computing Activities
612 West 115th Street
New York, NY 10025
USA

or call:

(USA) (212) 854-3703

MS-DOS Kermit for the IBM PC, PS/2, and compatibles is also available in bookstores, computer stores, and from the publisher:

Christine M. Gianone, *Using MS-DOS Kermit*, Second Edition, Digital Press, Bedford, MA, 1991, 345 pages. Packaged with MS-DOS Kermit for the IBM PC, PS/2, and compatibles on a 5.25-inch diskette.

Order Number: EY-H893E-DP
Digital Press ISBN: 1-55558-082-3
Prentice Hall ISBN: 0-13-952276-X

Internation Edition Order Number: EY-H893E-DI
International Prentice Hall ISBN: 0-13-953043-6

US single-copy price: $34.95; quantity discounts available. Available in computer bookstores or directly from Digital Press. In the USA, call Digital Press toll-free 1-800-344-4825 to order; major credit cards accepted. Overseas, order through your bookstore or your local Digital Equipment Corporation branch.

A German-language edition is also available:

Christine M. Gianone, *MS-DOS Kermit, Das universelle Kommunikationsprogramm*, Verlag Heinz Heise, Hannover, Germany (1991), 414 pages. Translated by Gisbert W. Selke. Packaged with version 3.11 of MS-DOS Kermit for the IBM PC, PS/2, and compatibles on a 5.25-inch diskette, including German language help files. ISBN 3-88229-006-4

1.4. Interactive Operation

C-Kermit's interactive command prompt is "`C-Kermit>`". In response to this prompt, you can type any valid interactive C-Kermit command. C-Kermit executes the command and then prompts you for another command. The process continues until you instruct the program to terminate.

Commands begin with a keyword, normally an English verb, such as "send". You may omit trailing characters from any keyword, as long as you specify sufficient characters to distinguish it from any other keyword valid in that field. Certain commonly-used keywords (such as "send", "receive", "connect") also have special non-unique abbreviations ("s" for "send", "r" for "receive", "c" for "connect").

Command keywords can be entered in either upper or lower case, or any combination. This manual shows command keywords in uppercase for clarity.

Certain characters have special functions while you are typing interactive commands:

`?`	Question mark, typed at any point in a command, will produce a message explaining what is possible or expected at that point. Depending on the context, the message may be a brief phrase, a menu of keywords, or a list of files. Use "?" liberally to feel your way through Kermit's commands.
`ESC`	(The Escape or Altmode key) -- Request completion of the current keyword or filename, or insertion of a default value. The result will be a beep if the requested operation fails.
`TAB`	(The horizontal Tab key) -- Same as ESC.
`DEL`	(The Delete or Rubout key) -- Delete the previous character from the command. You may also use BS (Backspace, Control-H) for this function.
`^W`	(Control-W) -- Erase the rightmost word from the command line.
`^U`	(Control-U) -- Erase the entire command.
`^R`	(Control-R) -- Redisplay the current command.
`^C`	(Control-C) -- Interrupt a command.
`SP`	(Space) -- Delimits fields (keywords, filenames, numbers) within a command.
`CR`	(Carriage Return) -- Enters the command for execution. LF (Linefeed) or FF (formfeed) may also be used for this purpose.
`\`	(Backslash) -- Enter any of the above characters into the command literally. To enter a backslash, type two backslashes in a row (`\\`). A backslash at the end of a command line causes the next line to be treated as a continuation line; this is useful for readability in command files. Backslash is also used to introduce special characters, variable names, and functions. These are explained in Section 1.12.
`^Z`	(Control-Z) -- On systems (like Berkeley UNIX, Ultrix) with job control, Control-Z is supposed to suspend Kermit, i.e. put it into the *background* in such a way that it can be brought back into

the foreground (e.g. with an '`fg`' shell command) with all its settings intact.

You may type the editing characters (`DEL`, `^W`, etc) repeatedly, to delete all the way back to the prompt. No action will be performed until the command is entered by typing carriage return, linefeed, or formfeed. If you make any mistakes, you will receive an informative error message and a new prompt -- make liberal use of '`?`' and ESC to feel your way through the commands. One important command is "help" -- you should use it the first time you run C-Kermit.

Comments may be entered as entire commands, or may appear at the end of any command. Comments are introduced by semicolon (`;`) or pound sign (#).

Interactive C-Kermit accepts commands from files as well as from the keyboard. When you start C-Kermit, the program looks for a special file, the *Kermit Initialization File* in your home or current directory (first it looks in the home directory, then in the current one) and executes any commands it finds there. These commands must be in interactive format, not command-line format. The name of the initialization file is `.kermrc` in UNIX and OS-9, and `CKERMIT.INI` in most other operating systems.

Here is a brief list of C-Kermit interactive commands:

`;`	Introduce a full-line or trailing comment (also #).
`!`	Execute a system command or enter system command interpreter.
`@`	Same as `!`.
ASK	Prompt the user, store user's reply in a variable.
ASKQ	Like ASK, but, but doesn't echo (useful for passwords).
ASSIGN	Assign an evaluated string to a variable or macro.
BUG	Instructions for reporting bugs.
BYE	Terminate and log out a remote Kermit server.
CD	Change Working Directory (also, CWD).
CLEAR	Clear communication device input buffer.
CLOSE	Close a log or other local file.
COMMENT	Introduce a full-line comment.
CONNECT	Establish a terminal connection to a remote computer.
DECLARE	Declare an array.
DECREMENT	Subtract one (or other number) from a variable.
DEFINE	Define a variable or macro.
DELETE	Delete a file or files.
DIAL	Dial a telephone number.
DIRECTORY	Display a directory listing.
DISABLE	Disallow access to selected features during server operation.
DO	Execute a macro.
ECHO	Display text on the screen.
ELSE	ELSE-part of an IF command.
ENABLE	Allow access to selected features during server operation.
END	A command file or macro.
EXIT	Exit from the program, closing all open files and devices.
FINISH	Instruct a remote Kermit server to exit, but not log out.
FOR	Execute commands repeatedly in a counted loop.
GET	Get files from a remote Kermit server.
GOTO	Go to a labeled command in a command file or macro.
HANGUP	Hang up the phone or network connection.
HELP	Display a help message for a given command.
IF	Conditionally execute the following command.
INCREMENT	Add one (or other number) to a variable.
INPUT	Match characters from another computer against a given text.
LOG	Open a log file -- debugging, packet, session, transaction.

MAIL	Send a file as electronic mail to a specified address.
MSEND	Multiple SEND -- send a list of files.
OPEN	Open a local file for reading or writing.
OUTPUT	Send text to another computer.
PAD	(X.25 version only) Give X.25 PAD commands.
PAUSE	Do nothing for a given number of seconds.
PRINT	Print a local file on a local printer.
PUSH	Invoke host system interactive command interpreter.
PWD	Display current working device/directory.
QUIT	Same as EXIT.
READ	Read a line from a local file.
RECEIVE	Passively wait for files to arrive.
REDIAL	Redial the most recently dialed phone number.
REINPUT	Reexamine text previously received from another computer.
REMOTE	Issue file management commands to a remote Kermit server.
RENAME	Change the name of a file.
RETURN	Return from a user-defined function.
RUN	Run a program or system command.
SCRIPT	Execute a UUCP-style login script.
SEND	Send files.
SERVER	Begin server operation.
SET	Set various parameters.
SHOW	Display values of SET parameters.
SPACE	Display current disk space usage.
STATISTICS	Display statistics about most recent transaction.
STOP	Stop executing macro or command file and return to the prompt.
SUSPEND	Put the Kermit program in the backgound.
TAKE	Execute commands from a file.
TELNET	Start a TCP/IP TELNET session.
TRANLATE	Translate a file's character set.
TRANSMIT	Upload a file with no error checking.
TYPE	Display a file on the screen.
VERSION	Display the program version number on the screen.
WAIT	Wait for the specified modem signals.
WHILE	Execute commands repeatedly while a condition is true.
WRITE	Write text to a local file.
XIF	Extended IF command.

On most computers, C-Kermit can be started simply by typing the word "kermit" (followed by carriage return). If no command-line "action options" are included, Kermit will automatically enter interactive prompting mode. Let's begin by looking at a few of C-Kermit's basic interactive commands. Most important of all when you are just beginning are the commands to exit from the program and to get help about it.

The EXIT and QUIT Commands

Syntax: EXIT [*number*] Syntax: QUIT [*number*]

These two commands are identical. Both of them do the following:

- Hangs up the modem, if the communications line supports data terminal ready.
- Relinquish access to any communication line assigned via SET LINE, or any network connection obtained via SET HOST.
- Relinquish any uucp and multiuser locks on the communications line (UNIX only).
- Close all open logs or other files.

- Attempt to insure that your terminal is returned to normal.

After exit from C-Kermit, your default directory will be the same as when you started the program. The EXIT command is issued implicitly whenever C-Kermit halts normally, e.g. after a command line invocation, or after certain kinds of interruptions.

C-Kermit returns an exit status of zero, except when an error is encountered, where the exit status is set to a nonzero number, depending on the operating system. In UNIX, the exit status is the sum of 1 (if any SEND commands failed), 2 (if any RECEIVE or GET commands failed), and 4 (if any REMOTE commands failed). If you give a number after the EXIT or QUIT command, that number is returned instead.

The HELP Command

Syntax: HELP [{ *command*, { SET, REMOTE } *parameter* }]

Brief help messages or menus are always available at interactive command level by typing a question mark at any point. A slightly more verbose form of help is available through the HELP command. The HELP command with no arguments prints a brief summary of how to enter commands and how to get further help. HELP may be followed by one of the top-level C-Kermit command keywords, such as SEND, to request information about a command. Commands such as SET and REMOTE have a further level of help. Thus you may type HELP, HELP SET, or HELP SET PARITY; each will provide a successively more detailed level of help.

The TAKE Command

Syntax: `TAKE` *fn1*

The TAKE command instructs C-Kermit to execute commands from the named file. The file may contain any interactive C-Kermit commands, including TAKE; command files may be nested to any reasonable depth, but it may not contain text to be sent to a remote system during the CONNECT command; Use scripts for that (see sections 1.13 and 1.14).

Echoing of commands from TAKE files and handling of errors during TAKE file execution are controlled by the SET TAKE command.

Comments may be included in take-command files. Whole-line comments may begin with the word COMMENT. Both trailing and whole-line comments can be introduced using semicolon or pound sign, for example:

```
    COMMENT - Commands to dial out out using a Hayes modem...
    set modem hayes        ; Specify the kind of modem
    set line /dev/ttyh8    ; Select a terminal device
    set speed 19200        ; Set the speed
    dial 7654321           ; Dial the number
```

TAKE-command files are in exactly the same syntax as interactive commands. If you want to include special characters like question mark or backslash that you would have to quote with backslash when typing interactive commands, you must quote these characters the same way in command files. Long lines may be continued by ending them with a single backslash or a dash:

```
    set\
    modem\
    hayes
    set-
    line /dev-
    /ttyh8
```

Continued lines cannot have trailing comments. If you put a trailing comment after the dash, then Kermit will not treat the line as continued. If you put a dash at the end of a trailing comment, then Kermit will treat the next line as a continuation of the comment.

An implicit TAKE command is executed upon your C-Kermit initialization file (`.kermrc` in UNIX, `CKERMIT.INI` elsewhere) when C-Kermit starts up, upon either interactive or command-line invocation. The initialization file may contain SET or other commands you want to be in effect at all times. For instance, you might want override the default action when incoming files have the same names as existing files -- in that case, put a command like:

```
    set file collision overwrite
```

Under UNIX, you may also use the shell's redirection mechanism to cause C-Kermit to execute commands from a file:

```
    kermit < cmdfile
```

or you can even pipe commands in from another process:

```
    command | kermit
```

1.5. Establishing the Connection

In order to talk about a connection between two computers, we a way to distinguish between them. The *local* computer is the one that you are using most directly -- a PC or workstation on your desk, or a timesharing computer that you are using from a terminal. Kermit lets you make a connection from the local computer to a *remote* computer, for example by dialing it up with a modem. The Kermit program on your local computer is said to be in "local mode" and the Kermit program on the remote computer is in "remote mode".

C-Kermit can be used in either *remote* or *local* mode. If you are using C-Kermit in remote mode, you can skip ahead to Section 1.6, File Transfer.

If you are using C-Kermit in local mode, you must have one of three kinds of connections to the remote computer:

1. A hardwired asynchronous serial connection, either directly to the other computer, or through some kind of terminal server or switching device.

2. A dialup connection, which you establish by dialing with a modem.

3. A network connection.

To prepare C-Kermit for these connections, several preliminary steps are necessary using the SET command, which is described in Section 1.11, below:

1. Direct connection: SET LINE *device-name*, SET SPEED *bits-per-second*.

2. Dialup connection, which you establish by dialing with a modem: SET MODEM *modem-type*, SET LINE *device-name*, SET SPEED *bits-per-second*, DIAL *phone-number*.

3. Network connection: SET HOST *network-host-name*. If your version of C-Kermit supports more than one type of network, give the appropriate SET NETWORK command before the SET HOST command (the default network type is TCP/IP). (Also see the TELNET command, described below.)

In addition, you must establish whatever other communication related settings are appropriate for the connection, including PARITY, DUPLEX, FLOW, and HANDSHAKE. Having done all this, you can use the CONNECT command to begin a terminal session with the remote computer, and once connected, you can initiate file transfers.

The SET LINE Command

Syntax: `SET LINE` [*device*]

Close any previously open network, terminal, or modem connection. If a device name is given, try to open the device for communication. If no device name is given, revert to the default mode of communication (normally remote mode).

The SET HOST Command

Syntax: `SET HOST` [*host* [*service*]]

Close any previously open network, terminal, or modem connection. If the current network type (see SET NETWORK) is TCP/IP, the *host* field can be specified as an IP host name or a numeric IP host address, followed optionally by a TCP service name or port number (the default service is TELNET, TCP port number 23). If the network type is X.25, the *host* field is an X.121 address, and there is no service field. If no host name is given, revert to the default mode of communication (normally remote mode).

The CONNECT Command

The CONNECT command (C is a special abbreviation for CONNECT) links your terminal to another computer as if it were a local terminal to that computer, through the device specified in the most recent SET LINE or SET HOST command, or through the default device if your system is a PC or workstation. All characters you type at your keyboard are sent out the communication line (and if you have SET DUPLEX HALF, also displayed on your screen), and all characters arriving at the communication port are displayed on the screen. Current settings of speed, parity, duplex, and flow-control are honored, and the data connection is 7 bits wide unless you have given the command SET TERMINAL BYTESIZE 8. If you have issued a LOG SESSION command, everything you see on your screen will also be recorded to your session log. This provides a way to "capture" files from remote systems that don't have Kermit programs available.

To get back to your own system, you must type the "escape character", which is Control-Backslash (`^\`) unless you have changed it with the SET ESCAPE command, followed by a single-character command, such as C for "close connection". Single-character commands may be entered in upper or lower case. They include:

`C`	Return to C-Kermit. If you gave an interactive CONNECT command, return to the C-Kermit prompt. If you gave a -c or -n option on the command line, close the connection and return to the system prompt.
`B`	Send a BREAK signal (about 0.275 sec).
`L`	Send a Long BREAK signal (about 1.5 sec).
`0`	(zero) send a null.
`S`	Give a status report about the connection.
`H`	Hangup the phone.
`!`	Escape to the system command processor "under" Kermit. Exit or logout to return to your CONNECT session.
`Z`	Suspend Kermit (UNIX only).
`\nnn`	A character in backslash-code form.

`^\` Send Control-Backslash itself (whatever you have defined the escape character to be, typed twice in a row sends one copy of it).

Uppercase and control equivalents for (most of) these letters are also accepted. A space typed after the escape character is ignored. Any other character will produce a beep.

The connect command simply displays incoming characters on the screen. It is assumed any screen control sequences sent by the host will be handled by the firmware or emulation software in your terminal or PC[1].

It is sometimes useful to see exactly what characters the host is transmitting, rather than having your terminal or emulator interpret them for you. If you give the command SET DEBUG SESSION, then during CONNECT, C-Kermit will display control characters using "uparrow" notation, for example `^A` for Control-A, `^B` for Control-B, etc, `^[` for ESC. 8-bit characters are preceded by a tilde (~) character.

The DIAL and REDIAL Commands

Syntax: `DIAL` *telephone-number*
Syntax: `REDIAL`

The DIAL command controls dialout modems; you should have already issued a SET MODEM command to identify the type of modem to be used for dialing, and then SET LINE and SET SPEED commands to identify the terminal device to which the modem is connected, plus any desired SET CARRIER and SET DIAL commands to modify the behavior of the DIAL command itself.

In the DIAL command, you supply the phone number and the Kermit program feeds it to the modem in the appropriate format and then interprets dialer return codes and modem signals to inform you whether the call was completed. These actions are based upon built-in knowledge of the control sequences and responses of each make and model of modem that C-Kermit knows about.

The telephone-number may contain imbedded modem-dialer commands appropriate to the selected modem type, such as comma for Hayes pause, or ‘&’ for Ventel dialtone-wait and ‘%’ for Ventel pause (consult your modem manual for details).

The REDIAL command is exactly like the DIAL command, except it uses the phone number you specified in your most recent DIAL command.

At the time of this writing, support is included for the following modems:

- AT&T 7300 Internal Modem
- AT&T 2212C, 2224B, 2224CEO, and 2296A switched network modems in AT&T mode
- AT&T Digital Terminal Data Module (DTDM)
- Cermetek Info-Mate 212A
- Concord Condor CDS 220
- Courier HST
- DEC DF03-AC
- DEC DF100 Series
- DEC DF200 Series
- General DataComm 212A/ED
- Hayes Smartmodem and compatibles
- IBM/Siemens/Rolm 9751 CBX DCM
- Microcom AX-9624

[1]Several C-Kermit implementations include terminal emulators, notably Macintosh and OS/2

- Penril
- Racal Vadic
- Telebit Trailblazer, T1000, T1600, T2500
- US Robotics 212A
- Ventel

Support for new modems is added to the program from time to time; you can check the current list by typing SET MODEM ?.

There are also two "generic" modem types -- DIRECT (i.e. no modem at all, so that no attempt is made to deal with modem signals), and UNKNOWN (which tells C-Kermit to attempt to honor modem signals, but leaves the dialing mechanism unspecified). NONE is a synonym for DIRECT.

The device used for dialing out is the one selected in the most recent SET LINE command (or on a workstation, the default line if no SET LINE command was given). On UNIX systems, the DIAL command attempts to lock the terminal device's path and to establish a call on an exclusive basis. If it is desired to dial a call and then return to the shell (such as to do Kermit activities depending on standard in/out redirection), it is necessary to place the dialed call under one device name (say, "`/dev/cua0`") and then escape to the shell *within Kermit* on a linked device which is separate from the dialed line (say, "`/dev/cul0`"). This is the same technique used by uucp (to allow locks to be placed separately for dialing and conversing).

Because modem dialers have strict requirements to override the carrier-detect signal most UNIX implementations expect, the sequence for dialing is more rigid than most other C-Kermit procedures. Example:

```
kermit
C-Kermit>set modem hayes      ; SET MODEM first!
C-Kermit>set carrier auto     ; Then SET CARRIER if necessary,
C-Kermit>set line /dev/acu2   ; Then SET LINE,
C-Kermit>set speed 2400       ; Then SET SPEED,
C-Kermit>set dial display on  ; Then SET DIAL if necessary,
C-Kermit>dial 9,5551212       ; and finally, DIAL.
Connected!                         ; If DIAL completes OK,
C-Kermit>connect              ; you can CONNECT.
logon, do things
^\c
C-Kermit> ...
C-Kermit>quit                 ; Disconnect & unlock the line.
```

In general, C-Kermit requires that the modem provide the "carrier detect" (CD) signal when a call is in progress, and remove that signal when the call completes or the line drops. If a modem switch setting is available to force CD, it should normally not be in that setting. C-Kermit also requires (on most systems) that the modem track the computer's "data terminal ready" (DTR) signal. If a switch setting is available to simulate DTR asserted within the modem, then it should normally not be in that setting. Otherwise the modem will be unable to hang up at the end of a call or when interrupts are received by Kermit.

If you want to interrupt a DIAL command in progress (for instance, because you just realize that you gave it the wrong number), type a Control-C get back to command level. If you are a UNIX user and your "interrupt character" is defined as something besides Ctrl-C, use that instead.

PROBLEMS: The DIAL command hangs up the phone (see HANGUP) prior to dialing. Because of the problems described for HANGUP, certain UNIX configurations are unable to use DIAL when it does this. To work around this problem, give the command SET DIAL HANGUP OFF before giving the DIAL command. If DIAL still doesn't work right, start over again but this time give the command SET CARRIER OFF prior to SET LINE.

See the command summary for a description of SET DIAL options. You can use SET DIAL to modify the modem initialization string, the timeout period for call completion, and so on.

Hayes and Hayes-compatible, as well as Telebit, modem dialing is supported for both word and digit result codes, and C-Kermit will also attempt to adjust its communication speed in case these modems complete a call at a lower speed than it was placed at (if you have a speed-matching modem, you can defeat this action by giving the command SET DIAL SPEED-MATCHING ON before dialing).

The HANGUP Command

The HANGUP command attempts to hang up the modem on a local-mode dialout connection established by SET LINE, or to hang up the network connection established by SET HOST, in order to break the connection. On terminal devices, Kermit accomplishes the hangup by momentarily turning off the Data Terminal Ready (DTR) RS-232 signal.

The means used to drop DTR and bring it back up again are highly dependent on the computer, the version of UNIX, the communication device, the device driver, the modem being used, and even the cable that connects the modem to the communication device. Therefore you might find that this command fails to operate as it should.

If the HANGUP command fails to hang up a phone connection, then maybe your modem is configured to ignore DTR. You should make sure your modem is configured to pay attention to the computer’s DTR signal. Another possibility is that your cable is giving the modem a constant DTR signal, looped back from its own DSR signal; in that case, replace your cable with a regular straight-through modem cable.

If the HANGUP command does indeed hang up the phone connection, but then you can’t communicate with the modem any more to establish a new connection, your version of UNIX may have dropped the DTR signal without bringing it back up again properly. To work around this problem, give the SET LINE command (with no device name) to close the current device, and then another SET LINE command specifying the original device to open it again.

The TELNET Command

Syntax: `TELNET` [*host* [*service*]]

The TELNET command can be used in C-Kermit implementations that support TCP/IP networking. TELNET is simply a shorthand for SET HOST followed by CONNECT. If you give an IP host name (or number), Kermit closes any currently open connection and begins a new one to the given host. If the TCP/IP connection is made successfully, Kermit puts itself into CONNECT mode for terminal emulation. If you give a TELNET command without a hostname, the currently active TCP/IP session (if any) is resumed; if there is no current session, you’ll get an error message. The *service* can be a TCP port number or service name; the default is 23, the TELNET port.

1.6. File Transfer

Kermit programs can transfer files correctly and completely, using the Kermit file transfer protocol. A Kermit program must be running on both computers. Here is the normal procedure:

1. Run Kermit on your local computer.

2. Make the appropriate communication settings (such as modem type, line, speed, parity, etc).

3. Establish a connection to the remote computer (DIAL if necessary).

4. CONNECT to the remote computer.

5. Log in to the remote computer if necessary.

6. Start Kermit on the remote computer and tell it to send (or receive) a file (or files).

7. "Escape back" to the local computer.

8. Tell the local computer to receive (or send) a file (or files).

When transferring files, C-Kermit converts between upper and lower case filenames and between LF and CRLF line terminators automatically, unless told to do otherwise. When binary files must be transferred, the program should be instructed not to perform LF/CRLF conversion (`-i` on the command line or SET FILE TYPE BINARY interactively; see below).

If C-Kermit is in local mode, the screen is continously updated to show the progress of the file transer. A dot is printed for every four data packets, other packets are shown by type:

I	Exchange Parameter Information
R	Receive Initiation
S	Send Initiatiation
A	Attribute Packet
F	File Header
G	Generic Server Command
C	Remote Host Command
N	Negative Acknowledgement (NAK)
E	Fatal Error
T	Indicates a timeout occurred
Q	Indicates a damaged, undesired, or illegal packet was received
%	Indicates a packet was retransmitted

You may type the following interruption commands during file transfer:

F	Interrupt the current File, and go on to the next (if any).
X	Interrupt the entire Batch of files, terminate the operation.
R	Resend the current packet.
E	Error: terminate the current operation immediately and return to prompt.
A	Display a status report for the current operation.

EMERGENCY EXIT: When running C-Kermit in remote mode, if you have started a protocol operation (sending or receiving a file, server command wait, etc), you can type two Control-C's directly to the UNIX Kermit program ("connect" first if necessary):

Control-C Control-C

This will cause the program to display,

```
^C^C...
```

and return you to the C-Kermit prompt.

Before initiating a file transfer, you should be sure that all communication and protocol-related settings are correct. Version 5A of Kermit attempts to make things easier for you in this area in several ways, for example by recognizing file attributes (text or binary, character set, etc) from the other Kermit if it sends them. If your parity is set to NONE, C-Kermit automatically recognizes parity bits on received packets, and adjusts its parity accordingly. But it's better to set these C-Kermit parameters directly yourself to avoid any confusion that might result from these automatic adjustments.

The SEND Command

Syntax: `SEND` *fn* [*rfn1*]

Send the file or files denoted by *fn* to the other Kermit, which should be running as a server, or which should be given the RECEIVE command. The SEND command may be abbreviated to S, even though S is not a unique abbreviation for a top-level C-Kermit command. Each file is sent under its own name (as described above, or as specified by the 'set file names' command). If the second form of the SEND command is used, i.e. with *fn1* denoting a single file, *rfn1* may be specified as a name to send it under. For example:

```
    send sows.ear silk.purse
```

sends the file `sows.ear` but tells the other Kermit that its name is `silk.purse`.

The MSEND Command

Syntax: `MSEND` *fn* [*fn* [*fn* [...]]]

The MSEND command also sends one or more files, but unlike the SEND command, it allows you to give a list of files to send, and it does *not* allow you to specify an alternate name. The MSEND file specifications are separated by spaces.

Both the SEND and MSEND commands allow you to use wildcard (meta) characters to specify groups of files. Exactly which wildcard characters are available depend on the host operating system and other factors. In VAX/VMS, they are '`*`' and '`%`'. In UNIX they are '`~`', '`*`', and '`?`'. If '`?`' is to be included, it must be prefixed by '`\`' to override its normal function of providing help. The '`*`' character matches any string, and '`?`' (UNIX) or '`%`' (VMS) matches any single character.

When *fn* contains '`*`' or '`?`' characters, there is a limit to the number of files that can be matched, which varies from system to system. If you get the message "Too many files match" then you'll have to make a more judicious selection. For example, if *fn* was of the form:

```
    usr/longname/anotherlongname/*
```

then C-Kermit's string space will fill up rapidly -- try using CD to change your directory to the path in question and reissuing the command.

Wildcards do not descend through a directory tree. Only files in the current or specified directory are sent.

In UNIX versions of C-Kermit only, '`~`' is treated as a meta character if it is the first character in the file specification. If it is followed immediately by a slash, a space, or end of line, then your login directory name is substituted. If it is followed immediately by a username, then that user's login directory name is substituted. (In VAX/VMS, you can use logical names in the file specification for the same purpose.)

Wildcard Notation for UNIX

Normally, C-Kermit expands wildcard characters itself by searching through the disk and matching names against the pattern you specify. But Kermit's pattern-matching abilities are limited to the metacharacters listed above. UNIX shells like csh and ksh, on the other hand, offer notations for file groups, like '`[bcdfhjlns]og`' or '`*.{txt,doc}`' that Kermit cannot handle on its own. Kermit lets you get at these capabilities with the command:

```
SET WILDCARD-EXPANSION { KERMIT, SHELL }
```

The default is KERMIT, meaning Kermit itself expands wildcards in SEND and MSEND, as well as what it receives while in server mode from GET commands. The SHELL option means Kermit calls upon the user's preferred shell (via the SHELL environment variable, or failing that, the user's login shell via getpwuid) to expand them.

Advantages of the SHELL method include more flexibility in file selection, consistency with the user's shell, and ability to handle a larger number of files. But there are drawbacks. It's noticably slower. Characters in filenames that are special to the shell (like |, &, ‘, etc) must be quoted with \ (this is not necessary when Kermit expands). And the same Kermit command (SEND, MSEND, or GET sent to a server) may behave differently depending on what your login shell is. For example, ‘‘`{aaa,bbb}`’’ is expanded by the csh, but taken literally by sh and ksh; ‘‘`[a-z]`’’ is expanded by csh and ksh, but taken literally by sh.

The expansion is done by passing "echo " concatenated with your filespec to the shell. This assumes that echo is a built-in shell command, or if it's not, then the echo program is in the user's path (risky). "echo" is used rather than "ls -d" because ls says "xxx not found" if xxx does not exist (unfortunately, the csh version of "echo" says "echo: no match" if you give it a string containing metacharacters and no files match, wherease sh and ksh simply echo the string back at you). It doesn't matter much, because C-Kermit checks each word that is returned for its existence as a file (but of course, you could have a file called "echo:", "no", or "match"...). And then what happens if your have a file called "`-n`"? ("`echo -n`" is a command. Refer to a file called "`-n`" in the current directory as "`./-n`").

Text vs Binary Files

When C-Kermit sends each file, it also sends certain information about the file in an "attribute packet", provided the other Kermit agrees to accept attribute packets. This information includes the size, type (text or binary, determined from the "-i" command-line option or the "set file type" command), creation date, and a code to let the other Kermit know that the file is being sent from a UNIX system. The other Kermit may accept or refuse the file based upon these attributes, for example, if it doesn't have enough disk space to store a file of the specified size.

The file type attribute allows C-Kermit, when sending a file, to tell the receiving whether it should be in text or binary mode. Therefore, if the receiving Kermit has this feature, it is not necessay to give it a SET FILE TYPE command to "match modes" with C-Kermit.

Note -- C-Kermit sends only from the current or specified directory. It does not traverse directory trees. If the source directory contains subdirectories, they will be skipped. By the same token, C-Kermit does not create directories when receiving files. If you have a need to do this, you can pipe tar through C-Kermit, as shown in the example in section 1.15, or under AT&T System III/V UNIX you can use cpio, or BACKUP under VAX/VMS, etc.

Another Note -- The SEND command skips over "invisible" files that match the file specification; UNIX systems usually treat files whose names start with a dot (like `.login`, `.cshrc`, and `.kermrc`) as invisible. If you want to send these, include the dot in your filespec, as in:

```
    send .*
```

The RECEIVE Command

Syntax: `RECEIVE` [*fn1*]

Passively wait for files to arrive from the other Kermit, which must be given the SEND command -- the RECEIVE command does not work in conjunction with a server (use GET for that). If *fn1* is specified, store the first incoming file under that name. The RECEIVE command may be abbreviated to R.

Incoming file data is normally decoded and stored according to whether C-Kermit is in text or binary mode. But if the other Kermit sends the file-type attribute, this will override C-Kermit's file-type setting on a per-file basis. Therefore, it is possible for another Kermit program to send C-Kermit a mixture of text and binary files, so long as the type of each file is indicated in the Attribute packet. For text files, C-Kermit will perform character-set translation if a known transfer character set is indicated in the Attribute packet (see section on International Characters).

C-Kermit also attempts to store the incoming file with the creation date that is specified in the Attribute packet. C-Kermit does not attempt to verify disk space against the announced size of the incoming file.

You can control C-Kermit's use of these attributes using the SET ATTRIBUTE command.

The STATISTICS Command

The STATISTICS command displays information about the most recent Kermit file transfer or other protocol transaction, including file and communication line input and output, timing and efficiency, as well as what encoding options were in effect (such as 8th-bit prefixing, repeat-count compression, packet lengths, window size).

Non-Protocol Data Transfers

It is also possible to transfer files with other computers that do not have a Kermit program available. To send a file to such a computer, use the TRANSMIT command:

Syntax: `TRANSMIT` *fn1*

This command sends the named file without error checking, obeying current settings for file type (text or binary), parity, and duplex.

In text mode, send the file a line at a time, with character translations done according to the current file and transfer character sets, using the SET TRANSMIT PROMPT character (linefeed, `\10`, by default) as a line turnaround character. That is, send a line from the file, wait until the prompt character comes in response, then send the next line, and so on. If zero (0) is specified for the prompt character, then send the whole file without waiting for any response or echo from the host. Don't try this unless you know that the host can successfully process long continuous bursts of input characters. Each line is terminated by a carriage return, just as you would type it at a terminal. Linefeeds are stripped unless you have given the command SET TRANSMIT LINEFEED ON. The computer to which you are transmitting the file should be prepared to receive it, for instance into a text editor.

In binary mode, send all the characters of the file with no modification and no line turnaround handshake (the TRANSMIT PROMPT setting is ignored). Use binary mode only if you know that the computer or device to which you are transmitting the file can receive arbitrary patterns of characters at full speed.

If you want to transmit 8-bit data over a 7-bit connection (i.e. PARITY is not NONE), you can use SET TRANSMIT LOCKING-SHIFT ON to have Kermit send SO/SI shifts around your 8-bit characters. The TRANSMIT command can be interrupted by typing Ctrl-C. Also see SET TRANSMIT.

The opposite of TRANSMIT is LOG SESSION, which lets you capture files or screen data from remote computers during CONNECT, also without error checking.

XMIT may be used as a synonym for TRANSMIT in the TRANSMIT, SET TRANSMIT, and SHOW TRANSMIT commands.

1.7. Being and Using a Kermit Server

The SERVER command places C-Kermit in "server mode" on the currently selected communication device or network connection. All further commands must arrive as valid Kermit packets from the Kermit on the other end of the line. If you are running C-Kermit on the remote computer, then you should escape back to your local computer after you give C-Kermit the SERVER command, and then issue all further commands to the C-Kermit server from your local Kermit's command prompt. It is also possible to work the other way: your local computer is the Kermit server and the remote computer issues the commands -- this can only work if the remote Kermit is driven by a TAKE command file (which should end with the command FINISH).

The C-Kermit server can respond to the following commands from the client:

Client Command	Server Response
GET	Sends files
SEND	Receives files
MAIL	Sends incoming files as e-mail to specified address
BYE	Attempts to log itself out
FINISH	Exits to level from which it was invoked
REMOTE CD	Changes working directory (also, remote cd)
REMOTE DIRECTORY	Sends directory listing
REMOTE DELETE	Removes files
REMOTE HELP	Lists these capabilities
REMOTE HOST	Executes a UNIX shell command
REMOTE KERMIT	Sends a Kermit command to a Kermit server
REMOTE LOGIN	Login to a Kermit server that requires this
REMOTE LOGOUT	Logout from a Kermit server
REMOTE PRINT	Receives a file and prints it
REMOTE SET	Changes its settings
REMOTE SPACE	Reports about its disk usage
REMOTE TYPE	Sends files to your screen
REMOTE WHO	Shows who's logged in

If the Kermit server is directed at an external line (i.e. it is in "local mode") then the console may be used for other work if you SET FILE DISPLAY OFF and run it in the background; normally the program expects the console to be used to observe file transfers and enter status queries or to interrupt commands. The way to get C-Kermit into background operation from interactive command level varies from system to system (e.g. on Berkeley UNIX you would halt the program with `^Z` and then use the C-Shell 'bg' command to continue it in the background). The more common method is to invoke the program with the desired command line arguments, including "`-q`", and with a terminating "&".

The C-Kermit server will accept a list of files in GET commands sent by the client, similar to the list you can give to the MSEND command, e.g.:

```
    MS-Kermit>get ~joe/new.txt /etc/termcap ../*.c
```

The file specifications are separated by spaces. If you need to include a space in a filename, quote it with a backslash, for example:

```
    MS-Kermit>get node"user\ passwd"::dev:[dir]name.ext
```

If you need to include a backslash, use two of them.

When the UNIX Kermit server is given a REMOTE HOST command, it executes it using the shell invoked upon login, e.g. the Bourne shell, the Korn Shell, or the Berkeley C-Shell (or whatever the user has set her SHELL environment variable to be).

The server can issue periodic NAK packets. You can control the rate at which this happens via SET SERVER TIMEOUT. Specifiying a value of zero instructs C-Kermit not to do this at all. A nonzero value, n, makes a NAK appear every n seconds during server command wait. This is useful if the client Kermit is not capable of timeouts, and a packet that it sends to the server is lost. The default server timeout is zero.

SECURITY. Before putting C-Kermit into server mode, you can give it commands to restrict the types of access it allows to clients. The command is DISABLE. You can turn these accesses back on using the ENABLE command. Here are the items that can be controlled in this way, with the effect of DISABLE noted. CAUTION: If you leave HOST enabled, the client can get around some of these restrictions with REMOTE HOST commands.

BYE	Ignore BYE commands from the client, stay in server mode.
CD	Don’t let the client change directories, or send files to, or get files from, or delete or type files in, any but the current directory, nor inquire about space in any but the current directory.
DELETE	Don’t let the client delete any files.
DIRECTORY	Don’t let the client request a directory listing.
FINISH	Ignore FINISH commands from the client, stay in server mode.
GET	Don’t let the client GET files.
HOST	Ignore REMOTE HOST commands from the client.
SEND	Don’t accept files that the client tries to send.
SET	Refuse REMOTE SET commands from the client.
SPACE	Refuse REMOTE SPACE commands from the client.
TYPE	Refuse REMOTE TYPE commands from the client.
WHO	Refuse REMOTE WHO commands from the client.
ALL	All of the above.

Using a Kermit Server

C-Kermit may itself request services from a remote Kermit server. To send a file to a Kermit server, use the SEND command, just as you would to send a file to a Kermit program that has been given the RECEIVE command. But you cannot use the RECEIVE command to get a file from a Kermit server, because the RECEIVE command just waits passively for a file to arrive, but the server has not been told which file to send. For this, you need the GET command:

Syntax: `GET` *rfn*

```
         or: GET
                 rfn
                 fn1
```

The GET command requests a remote Kermit server to send the named file or files, it must be used in place of RECEIVE. Since a remote file specification (or list) might contain spaces, which normally delimit fields of a C-Kermit command, an alternate form of the command is provided to allow the inbound file to be given a new name: type GET alone on a line, and you will be prompted separately for the remote and local file specifications, for example:

```
    C-Kermit>get
     Remote file specification: profile exec
     Local name to store it under: profile.exec
```

If a ‘?’ is to be included in the remote file specification, you must prefix it with ‘\’ to suppress its normal function of providing help.

If you have started a multiline GET command, you may escape from its lower-level prompts by typing a carriage return in response to the prompt, e.g.

```
    C-Kermit>get
     Remote file specification: foo
     Local name to store it under: (Type a carriage return here)
    (cancelled)
    C-Kermit>
```

After the GET command has been entered, the file transfer proceeds exactly as if you had given a SEND command to the other Kermit and a RECEIVE command to this one.

In addition to SEND and GET, the following commands may also be sent from C-Kermit to a Kermit server. If the server does not support a command that you send to it, it will respond with a message like "Unknown server command" or "Unimplemented REMOTE command".

REMOTE CD [*directory*]
: Request the server to change its default directory (and/or device) to the one you specify. If none is specified, the server is requested to return to its primary default directory (normally the login directory).

REMOTE DELETE *rfn*
: The server is requested to delete the specified file or files.

REMOTE DIRECTORY [*rfn*]
: The server is requested to send a directory listing of the specifed files. If no files are specified, then the server should send a listing of all the files in the current directory.

REMOTE HELP
: The server is requested to send to your screen a list of the commands that you may issue to it from your local Kermit.

REMOTE HOST *command*
: The server is requested to ask its host operating system to execute the given command.

REMOTE KERMIT *command*
: The remote server is requested to execute the given command, which is in its own (Kermit program) syntax.

REMOTE LOGIN *userid password* [*account*]
: If a remote Kermit server has been set up to require to provide a user ID and password before you can gain access to it, use this command to send it the user ID and password.

REMOTE LOGOUT
: Remove your access from a remote Kermit server that you have REMOTE LOGIN’ed to.

REMOTE PRINT *file* [*options*]
: The file is sent to the server, which is requested via Attribute packet to print it on its printer using the specified print options.

REMOTE SET *parameter value*
: The server is requested to set the given parameter to the specified value. Use question mark to find out what parameters and values are available. The meanings of these parameters and values are explained in the section on the SET command, below. Perhaps the most useful one is REMOTE SET FILE TYPE { TEXT, BINARY }.

REMOTE SPACE [*dir*]
: The server is requested to report on available disk space in the specified device and/or directory, or in the current area if none specified.

REMOTE TYPE [*rfn*]
: The server is requested to send the specified file for display on your screen.

REMOTE WHO [*user*]
: The server is requested to send a list of the users who are logged in on its computer. If a username is given, information about that user is requested.

BYE *and* FINISH:
: When connected to a remote Kermit server, these commands cause the remote server to terminate; FINISH returns it to Kermit or system command level (depending on the implementation or how the program was invoked); BYE also requests it to log itself out.

1.8. International Character Sets

Text files may be written in languages other than English, and in most cases that means that they contain non-ASCII characters such as accented letters, special punctuation marks, etc. Many different character sets have been devised to represent different languages. These character sets are sometimes specific to a particular computer manufacturer, and sometimes they conform to one standard or another. The Kermit file transfer protocol permits Kermit programs to exchange files written in different character sets by translating between each computer's local file character set and a standard set that is used "on the wire". Here are the relevant commands:

SET FILE TYPE TEXT
: Character set translation is not done at all unless the file type is text.

SET TRANSFER CHARACTER-SET *name*
: Tell C-Kermit which character set is to be used on the wire, i.e. within the Kermit packets. The choices are TRANSPARENT (no translation, this is the default), ASCII (translate to/from US ASCII), LATIN1 (ISO 8859-1 Latin Alphabet 1), CYRILLIC-ISO (ISO 8859-5 Latin/Cyrillic Alphabet), and JAPANESE-EUC. Synonym: SET XFER CHARACTER-SET.

SET FILE CHARACTER-SET *name*
: Tell C-Kermit which character set is used in the local file, ASCII, or any of a dozen or so 7-bit national replacement character sets (NRCs) such as ITALIAN, NORWEGIAN, etc, or an 8-bit character set like LATIN1, DEC-Multinational, DG-International, NEXT (for NeXT workstations), CP437 or CP850 (for IBM or Xenix systems), APPLE (for A/UX on the Macintosh); CP866, KOI8-CYRILLIC, CYRILLIC-ISO, and SHORT-KOI for Cyrillic; and JAPANESE-EUC, DEC-KANJI, SHIFT-JIS, and JIS7 for Japanese Kanji.

SET LANGUAGE *name*
: Enable certain language-specific translations, especially for DUTCH, ICELANDIC, GERMAN, RUSSIAN, and the Scandinavian languages. In particular, SET LANGUAGE RUSSIAN makes Kermit treat ASCII as if it were SHORT-KOI when it is a FILE or TRANSFER character set, and the corresponding TRANSFER or FILE character set is an 8-bit Cyrillic set.

TRANSLATE *file1 cs1 cs2* [*file2*]
: Translate the local file *file1* from the character set *cs1* into the character set *cs2*. Both character sets may be selected from C-Kermit's repertoire of file character sets. The result is stored in *file2* or (if *file2* is not specified), displayed on the screen. If LANGUAGE is set to RUSSIAN, ISO Latin-Cyrillic is used as the intermediate standard character set during translation, otherwise ISO Latin Alphabet 1 is used. Does not yet work for Kanji.

The SET commands are described in more detail later.

When sending a file, C-Kermit translates the file from the specified file character set into the specified transfer character set, and an identifier for the selected transfer character set is included in the file attribute packet, if use of attribute packets has been negotiated and a SET ATTR [CHARACTER-SET] OFF command has not been given.

When receiving, C-Kermit translates from the specified transfer character set to the specified file character set or, if a transfer character set is identified in the incoming attribute packet it is used instead. If the file character set is ASCII and special characters are being received, then they are translated by stripping diacritical marks and aiming for the closest possible 1-to-1 translation on a per-character basis. But if a SET LANGUAGE command has been given, special translations can be done. For example if LANGUAGE has been set to GERMAN and the local file character set is ASCII, and a file arrives with a transfer character set of LATIN1, then C-Kermit will translate German umlaut-vowels into the corresponding vowels followed by the letter e, and German double-s into two s's. Similar translations take effect for other languages.

If unwanted effects appear because of Kermit's translation, you can disable this feature with SET ATTRIBUTE CHARACTER-SET OFF.

When transferring text that contains long runs of 8-bit data over a 7-bit communication channel, efficiency can be improved with the following command:

SET TRANSFER LOCKING-SHIFT { OFF, ON, FORCED }
: Specifies whether locking shifts should be used by Kermit when encoding and decoding packets. A locking shift is a special character, Ctrl-N (Shift-Out, SO), that means that all the following characters, up to the next Ctrl-O (Shift-In, SI), are to have their 8th bits set to 1 upon receipt. For long runs of 8-bit characters, this is more efficient than Kermit's regular "single shift" method of prefixing each 8-bit character by an &-sign. Synonym: SET XFER LOCKING-SHIFT. The options are:

  ON This is the default setting. If PARITY is not NONE, try to negotiate the use of locking shift protocol with the other Kermit, and use it if the other Kermit agrees. If PARITY is NONE, locking shifts won't be used.

  OFF Don't use locking shifts.

  FORCED
  Use locking shifts, regardless of the parity setting and negotiations. Automatically disables the use of 8th-bit prefixing (single shifts). For the file sender, this command lets data be sent to the receiver with embedded SO and SI characters, which can be processed properly by many terminals, printers, and other devices. For the file receiver, this command forces the treatment of SO and SI characters in the data as shift commands.

Locking shifts are most effective for text written in non-Roman alphabets and encoded in standard 8-bit character sets like ISO Latin/Cyrillic, ISO Latin/Arabic, ISO Latin/Hebrew, ISO Latin/Greek, or Japanese EUC. Locking-shift status is displayed by the SHOW and STATISTICS commands.

1.9. Logging Things

Kermit's actions during terminal connection and file transfer can be logged in several ways, using the LOG command:

Syntax: `LOG {DEBUGGING, PACKETS, SESSION, TRANSACTIONS}` [*fn1* [{ APPEND, NEW }=NEW]]

C-Kermit's progress may be logged in various ways. The LOG command opens a log, the CLOSE command closes it. In addition, all open logs are closed by the EXIT and QUIT commands. A name may be specified for a log file; if the name is omitted, the file is created with a default name as shown below. If the keyword APPEND is included at the end of the command, then if the specified file exists, it is appended to rather than written over; if it does not exist, a new file is created.

LOG DEBUGGING
: This produces a voluminous log of the internal workings of C-Kermit, of use to Kermit developers or maintainers in tracking down suspected bugs in the C-Kermit program. Use of this feature slows down the Kermit protocol and fills up your disk. Default name: `debug.log`.

LOG PACKETS
: This produces a record of all the packets that go in and out of the communication port. This log is of use to Kermit maintainers who are tracking down protocol problems in either C-Kermit or any Kermit that C-Kermit is connected to. Default name: `packet.log`.

LOG SESSION
: This log will contain a copy of everything you see on your screen during the CONNECT command, except for local messages or interaction with local escape commands. Default name: `session.log`.

LOG TRANSACTIONS
: The transaction log is a record of all the files that were sent or received while transaction logging was in effect. It includes time stamps and statistics, filename transformations, and records of any errors that may have occurred. The transaction log allows you to have long unattended file transfer sessions without fear of missing some vital screen message. Default name: `transact.log`.

The CLOSE command closes the named log, e.g. CLOSE DEBUG, CLOSE SESSION.

Note: Debug and Transaction logs are a compile-time option; C-Kermit may be compiled without these logs, in which case it will run faster, it will take up less space on the disk, but the commands relating to them will not be present.

1.10. Local File Management

C-Kermit allows the following local file management functions from its interactive command level:

CD [*directory-name*]
: Changes Kermit's working device and/or directory to the one given, or to the user's default directory and/or device if the name is omitted. This command affects only the Kermit process and any processes it may subsequently create. You may use CWD and SET DEFAULT as synonyms for CD.

DELETE *fn*
: Deletes (removes, erases) file *fn*, which may be the name of a single file or a wildcard specification containing `*` and/or `?` characters. You may use RM as a synonym for DELETE.

DIRECTORY [*fn*]
: Displays a listing of the files whose names match *fn*, which may be the name of a single file or a wildcard specification containing `*` and/or `?` characters. If no *fn* is given, C-Kermit lists all files in the current device and/or directory. The format of and information provided in the listing depends on the operating system. You may use LS as a synonym for DIRECTORY.

PWD
: (Print Working Directory) Display the current working (default) file device and/or directory. You may use SHOW DEFAULT as a synonym for PWD.

PRINT *fn1* [*options*]
: Print the local file *fn1* on a local printer. Options can be included after the filename, for example, to select a particular printer. The options are in the format of the local system's printing command.

RENAME *fn1 fn2*
: Changes the name of file *fn1* to *fn2*. You may use MV as a synonym for RENAME.

SPACE
: Displays information about disk space and/or quota in the current directory and device.

TYPE *fn*
Display the named file on the screen. May be interrupted with Ctrl-C, and on most systems the display can be stopped and resumed with Ctrl-S and Ctrl-Q, respectively. You may use CAT as a synonym for TYPE.

{ @, !, RUN, PUSH }
The *command* is executed by your computer's operating system command interpreter (UNIX shell, VMS DCL, etc). If no command is specified, then an interactive session is started; exiting from this session, e.g. by typing Control-D or 'exit' to a UNIX shell, or LOGOUT to VMS DCL, will return you to C-Kermit command level. Use the '!' command to provide file management or other functions not explicitly provided by C-Kermit commands. !, @, RUN, and PUSH are all synonyms for the same command.

1.11. The SET and SHOW Commands

Since Kermit is designed to allow diverse computers to communicate, it is often necessary to issue special instructions to allow Kermit to adapt to peculiarities of the other computer or the communication path. These instructions are accomplished by the SET command. The following parameters may be SET:

ATTRIBUTES	Turn Attribute packet processing on or off.
BACKGROUND	Force foreground or background mode.
BLOCK-CHECK	Level of packet error detection.
BUFFERS	Send and receive packet buffer sizes.
CARRIER	Treatment of carrier on terminal connections.
CASE	Controls treatment of alphabetic case.
COMMAND	Character set size for commands.
COUNT	For counted loops.
DEBUG	Log or display debugging information.
DEFAULT	Default directory.
DELAY	How long to wait before sending first packet.
DIAL	Parameters for DIAL command.
DUPLEX	Specify which side echoes during CONNECT.
ESCAPE	Prefix for "escape commands" during CONNECT.
FILE	Set various file parameters.
FLOW-CONTROL	Communication line full-duplex flow control.
HANDSHAKE	Communication line half-duplex turnaround character.
HOST	Specify network host name.
INCOMPLETE	Disposition for incompletely received files.
INPUT	Control behavior of INPUT command.
LANGUAGE	Enable language-specific character-set translations.
LINE	Communication line device name.
MACRO	Control aspects of macro execution.
MODEM-DIALER	Type of modem-dialer on communication line.
PAD	(X.25 systems only) X.3 PAD parameters.
PARITY	Communication line character parity.
PROMPT	The C-Kermit program's interactive command prompt.
RECEIVE	Parameters for inbound packets.
RETRY	Packet retransmission limit.
SCRIPT	Parameters for the SCRIPT command.
SEND	Parameters for outbound packets.
SERVER	Parameters for server operation.
SESSION-LOG	(UNIX only) Session log file type, text or binary.
SPEED	Communication line speed.
SUSPEND	Enable/Disable SUSPEND.

TAKE	Control aspects of TAKE file execution.
TERMINAL	Terminal parameters.
TRANSFER	File transfer parameters.
TRANSMIT	Control aspects of TRANSMIT command execution.
UNKNOWN	Specify handling of unknown character sets.
WINDOW	File transfer packet window size.
WILDCARD	(UNIX only) Kermit vs shell wildcard expansion.
X.25	(X.25 systems only) X.25 call parameters

The SHOW command may be used to display current settings. Here is a summary of settings available in C-Kermit, listed alphabetically.

SET ATTRIBUTES

Tells C-Kermit whether to exchange file attribute (A) packets, or whether to include specified attributes within the A packets it sends, or whether to pay attention to specific attributes in A packets it receives. When the use of A packets has been negotiated, C-Kermit enables and uses the ones listed below. When sending files, C-Kermit responds to an Attribute refusal from the other computer by not sending the specified file. For example, if C-Kermit announces in the A packet that the file is 100K long, the other Kermit could refuse the file because of insufficient disk space, and then C-Kermit would not send it.

SET ATTRIBUTE CHARACTER-SET { ON, OFF }

Turn the character-set attribute ON or OFF. If ON, C-Kermit includes a code for the transfer character set in the A packet when sending a file, and when receiving a file, C-Kermit will translate from the character set (if any) specified in the incoming A packet into the current file character set (see SET FILE), if the transfer character set is known to C-Kermit (see SET UNKNOWN).

SET ATTRIBUTE DATE { ON, OFF }

If ON, C-Kermit includes the file's creation date in the A packet when sending a file, and stores incoming files with the creation dates (if any) given in the incoming A packets. Beware: on some computers, incremental backups might skip over files with creation dates older than the most recent backup.

SET ATTRIBUTE DISPOSITION { ON, OFF }

The MAIL and REMOTE PRINT commands work by setting "dispositions" of Mail and Print in the A packet when sending a file. SET ATTR DISP OFF will cause C-Kermit to ignore this and store such files on disk rather than mailing or printing them.

SET ATTRIBUTE LENGTH { ON, OFF }

When sending files, C-Kermit puts their length in the A packet, so the receiving system has an opportunity to check and/or allocate disk space in advance, and to refuse the file if there is not enough disk space. If you believe that files are being unjustly refused on the basis of length, you can SET ATTR LENGTH OFF. When receiving files, C-Kermit presently ignores the length announced in the A packet.

SET ATTRIBUTE OS-SPECIFIC { ON, OFF }

Certain versions of Kermit convey operating-system specific file information in the A packet, which is primarily useful when transferring files between like systems, such as VMS to VMS. If you find this feature is interfering with successful file transfer, SET ATTR OS OFF.

SET ATTRIBUTE SYSTEM-ID { ON, OFF }

The A packet also includes a code identifying the file's operating system of origin, so that the receiver of the file can decide whether to pay attention to the OS-SPECIFIC attribute. Use SET ATTR SYS OFF to disable this feature.

SET ATTRIBUTES ALL { ON, OFF }
: You can turn all the above off using SET ATTR ALL OFF, and back on using SET ATTR ALL ON.

SET ATTRIBUTES { ON, OFF }
: This command turns the attribute mechanism itself on or off, without disturbing the settings of the individual attributes. It is on by default.

SET BACKGROUND { OFF, ON }

Use SET BACKGROUND off to make your prompts and messages appear in case they have disappeared because Kermit thinks it is running in the background (for example, if you are running it through an output filter, as in "kermit | vt100"). You can accomplish the same effect using the `-z` option on the command line.

SET BLOCK-CHECK {1, 2, 3}

Determines the level of per-packet error detection. "1" is a single-character 6-bit checksum, folded to include the values of all bits from each character. "2" is a 2-character, 12-bit checksum. "3" is a 3-character, 16-bit cyclic redundancy check (CRC). The higher the block check, the better the error detection and correction and the higher the resulting overhead. Type 1 is most commonly used; it is supported by all Kermit implementations, and it has proven adequate in most circumstances. Types 2 or 3 should be used when transferring 8-bit data or when using long packets.

SET BUFFERS *n1 n2*

Allows you to change the total buffer space used for sending packets (*n1*) and receiving packets (*n2*), if Kermit has been configured to allow dynamic buffer allocation. The bigger you make them, the longer your packets can be, and the more window slots you can use. To see the default values for your version of Kermit, type SHOW PROTOCOL.

SET CARRIER {AUTO, ON [*n*], OFF}

Specifies C-Kermit's treatment of the Carrier (CD, DCD, or RLSD) RS-232 signal on terminal device connections obtained via SET LINE:

SET CARRIER ON
: Means to require carrier at all times; SET LINE will not return until carrier appears (handy for setting Kermit up to wait for a call to come in), and a fatal error will occur if carrier disappears during CONNECT or file transfer. A nice side effect is that when you log out from the remote computer, you will be put back at C-Kermit prompt level automatically if the remote system drops carrier properly. If you want to set a time limit on how long SET LINE will wait for carrier, you can include an optional number after SET CARRIER ON to specify the number of seconds to wait before timing out and returning to the prompt, for example SET CARRIER ON 30.

SET CARRIER OFF means to ignore carrier at all times. Useful for direct, nonmodem connections, or misbehaving modem connections. Should only be used when necessary, because it takes away the ability of Kermit to automatically detect a broken phone connection.

SET CARRIER AUTO means to require carrier during CONNECT but not at other times. AUTO is the default. SET CARRIER ON does not affect the DIAL command.

SET CASE { ON, OFF }

Tells C-Kermit whether to pay attention to or ignore alphabetic case in string matching operations, including INPUT and IF. Default is OFF, i.e. ignore case.

SET COMMAND BYTESIZE { 7, 8 }
: Normally, C-Kermit's command processor strips off the 8th bit of any character you type, in case your data connection to C-Kermit has parity. If there is no parity and you want to use an 8-bit international character set in your commands (for example, in the ECHO command, or in filenames), use SET COMMAND BYTESIZE 8. The command also applies to the connection between your terminal (if any) and C-Kermit during terminal emulation (CONNECT), as opposed to the connection between C-Kermit and the remote host (see SET TERMINAL BYTESIZE).

SET COUNT *n*
: For use with counted loops. See the section on script language programming.

SET DEFAULT *directory*
: Change default directory. Equivalent to CD.

SET DIAL DISPLAY { ON, OFF }
: Show dialing activity on screen.

SET DIAL HANGUP { ON, OFF }
: Normally, the DIAL command will attempt to hang up the phone before dialing a new call. This is done by momentarily dropping the Data Terminal Ready (DTR) signal on the currently selected terminal device. Unfortunately, the method for doing this is ill-defined in some versions of UNIX, and in some cases DTR goes down and stays down, which can prevent all further communication between the computer and the modem. If you find that the DIAL command does not work for you at all, try giving the command SET DIAL HANGUP OFF before giving the DIAL command. This will prevent DIAL from hanging up the phone before dialing the new call.

SET DIAL INIT-STRING *string*
: Replace Kermit's built-in initialization string with *string*, which may contain backslash codes to represent control characters.

SET DIAL MNP-ENABLE { ON, OFF }
: Enable or disable MNP protocol negotiations by the modem. Normally off. This command presently applies only to Telebit modems.

SET DIAL KERMIT-SPOOF { OFF, ON }
: Enable modem's "Kermit spoof" if it has one.

SET DIAL SPEED-MATCHING { OFF, ON }
: Tells whether the modem's interface speed is locked. Default is OFF, meaning the modem changes its interface speed to match the connection speed, for example if you place a call at 2400 bps and it is answered at 1200 bps. Use ON if your modem is configured to do speed matching.

SET DIAL TIMEOUT *number*
: Seconds to wait for a response from the modem when dialing before giving up and declaring that the call could not be completed. If you don't give this command, Kermit calculates its own dial timeout based on the modem type, length of the phone number, and transmission speed. If that interval isn't long enough, use this command to specify a longer one.

SET DEBUG { ON, OFF, SESSION }
: SET DEBUG ON is equivalent to LOG DEBUG. SET DEBUG SESSION means to display incoming control and 8-bit characters on the screen using special notation (see CONNECT). SET DEBUG OFF turns off all debugging.

SET DELAY *n*
: How many seconds to wait before sending the first packet after a SEND command. Used in remote mode to give you time to escape back to your local Kermit and issue a RECEIVE command before the first Kermit packet appears. Normally 5 seconds.

SET DUPLEX {FULL, HALF}
: For use during CONNECT. Specifies which side is doing the echoing; FULL means the other side, HALF means C-Kermit must echo your keystrokes itself. Normally FULL. Use half when communicating with IBM mainframes over linemode connections, and on similar half-duplex or local-echo connections. Synonym: SET LOCAL-ECHO {OFF, ON}.

SET ESCAPE-CHARACTER *cc*
: For use during CONNECT to get C-Kermit's attention. The escape character acts as a prefix to an escape command, for instance to close the connection and return to C-Kermit or UNIX command level. The normal escape character is Control-Backslash (ASCII 28). See CONNECT.

SET FILE *parameter value*
: Establish file-related parameters:

SET FILE BYTESIZE { 7, 8 }
: Normally 8. If 7, strip the 8th bit from file data during file transfer.

SET FILE CHARACTER-SET *name*
: tells the encoding of the local file, ASCII by default. The names DUTCH, GERMAN, FRENCH, etc, refer to 7-bit ASCII-based national replacement character (NRC) sets. CP866, CYRILLIC-ISO, and KOI8-CYRILLIC refer to sets used in the Soviet Union and other countries that use the Cyrillic alphabet. Latin-1 is the 8-bit ISO 8859 Latin Alphabet 1. JAPANESE-EUC, DEC-KANJI, JIS7, and SHIFT-JIS are Japanese Kanji sets. Type SET FILE CHAR ? for a complete list. When receiving files, C-Kermit translates from the transfer character set specified in the most recent SET TRANSFER CHARACTER-SET command or else the one announced in the Attribute packet (if any) into the file character set. When sending files, C-Kermit translates from the current file character set into the current transfer character set.

SET FILE COLLISION *action*
: Tells what to do when a file arrives that has the same name as an existing file. The actions are:

BACKUP
: (default) Rename the old file to a new, unique name and store the incoming file under the name it arrived with. The "new unique" name is simply a new generation on VAX/VMS. In UNIX, a "generation number" appended to the filename, separated by a tilde, as in `oofa.txt.~8~`[2]

> WARNING: UNIX C-Kermit presently offers no method of automatically limiting the number of versions of a file that can be created, so if too many of them pile up, you'll have to delete the ones you don't need by hand. Or, because the version number syntax is compatible with GNU EMACS backup files, you can use EMACS to do this for you.

In OS/2, Kermit constructs names of the form `FZZ`*n*`.BAR`, where `FZZ.BAR` is the name they share and *n* is a "generation number"; if `FZZ.BAR` exists, then the new file will be called `FZZ00001.BAR`. If `FZZ.BAR` and `FZZ00001.BAR` exist, the new file will be `FZZ00002.BAR`, and so on. If the common name were more than 6 characters long (eg `GOODDATA.DAT`), then the new name for the arriving file would be `GOODD001.DAT` and so on.

[2]The new name for the arriving file is of the form `foo.~`*n*`~`, where foo is the name they share and *n* is a "generation number"; if *foo* exists, then the new file will be called `foo.~1~`. If `foo` and `foo.~1~` exist, the new file will be `foo.~2~`, and so on. If the new name would be longer than the maximum length for a filename, then characters are deleted from the end first, for instance, `thelongestname` on a system with a limit of 14 characters would become `thelonges.~1~`.

OVERWRITE
: Overwrite (replace) the existing file. The existing file is gone, destroyed. Even if the file transfer fails or the incoming file is refused. Use with caution.

APPEND
: Append the incoming file to the end of the existing file. This option is useful for adding information to a log file, but it should be used with caution to avoid, for example, joining two files of different types (like text and binary).

DISCARD
: Refuse and/or discard the incoming file. This option is handy for resuming multi-file transmissions that were broken. Only those files that were not successfully transferred before will be accepted.

RENAME
: Give the incoming file a unique name, like the BACKUP option, except that the incoming file gets the new name, rather than the existing file.

UPDATE
: Accept the incoming file only if it is newer than the existing file. This feature depends on the creation date field in the attribute packet.

SET FILE DISPLAY { ON, OFF }
: Normally ON; when in local mode, display progress of file transfers on the screen (stdout), and watch the keyboard for interruptions. If OFF (-q on command line) none of this is done, and the file transfer may proceed in the background oblivious to any other work concurrently done at the keyboard.

SET FILE NAMES {CONVERTED, LITERAL}
: Normally CONVERTED, which means that outbound filenames have device, directory, and/or path specifications stripped, lowercase letters raised to upper, tildes and extra periods changed to X's, and an X inserted in front of any name that starts with period. In VMS, the generation number is also stripped. For UNIX only, incoming filenames have uppercase letters lowered. LITERAL means that none of these conversions are done; therefore, any directory path appearing in a received file specification must exist and be write-accessible. When literal naming is being used, the sender should not use path names in the file specification unless the same path exists on the target system and is writable.

SET FILE RECORD-LENGTH *n*
: Tells the record length for fixed-format files, or the maximum record length for variable-format files, on systems like VAX/VMS where record-length is a meaningful concept.

SET FILE TYPE {BINARY, TEXT}
: The file type is normally text, which means that conversion is done between the local computer's record format and Kermit's standard transfer format, for example between UNIX newline characters and Kermit's carriage-return/linefeed sequences. BINARY means to transmit file contents without conversion. Binary ('`-i`' in command line notation) is necessary for binary files, and desirable in all file transfers between like systems to cut down on overhead.

For VAX/VMS, these are the file type options:

SET FILE TYPE TEXT
: Regular Text mode. Applies to incoming files only. When SENDing files, C-Kermit determines their type automatically.

SET FILE TYPE BINARY [{ FIXED, UNDEFINED }]
: Applies to incoming files only. Binary mode: incoming files are stored with no translation or conversion. The default record format is FIXED, but you can specify UNDEFINED too.

SET FILE TYPE IMAGE
This one applies to both incoming and outbound files. For incoming files, it's just like BINARY FIXED. For outbound files, it means just send the blocks of the file as they are stored on the disk, and ignore the file's RMS attributes. SET FILE TYPE BLOCK is a synonym (for compatibility with Kermit-32).

SET FILE WARNING { ON, OFF }
SET FILE WARNING is an old command, somewhat misnamed. SET FILE COLLISION (above) should be used instead. SET FILE WARNING ON is equivalent to SET FILE COLLISION RENAME and SET FILE WARNING OFF is equivalent to SET FILE COLLISION OVERWRITE.

SET FLOW-CONTROL {DTR/CD, NONE, RTS/CTS, XON/XOFF}
Normally XON/XOFF for full duplex flow control. Should be set to NONE if the other system cannot do Xon/Xoff flow control, or if you have issued a SET HANDSHAKE command. If set to XON/XOFF, then HANDSHAKE should be set to NONE. This setting applies during both terminal connection and file transfer. *Warning:* This command may have no effect on certain UNIX systems, where Kermit puts the communication line into "rawmode" and rawmode precludes flow control.

The DTR/CD and RTS/CTS options are two kinds of "hardware flow control", using special wires (other than the data wires) in the connector. These forms of flow control (especially RTS/CTS) are especially useful with high-speed modems and similar devices. But these options are only available in those versions of C-Kermit whose underlying operating system supports them.

Certain versions of UNIX, such as that on the NeXT, provide RTS/CTS flow control in a different way, namely in the device driver. For example, /dev/cua is the first dialout port on the NeXT without RTS/CTS flow control, and /dev/cufa is the same port, but with built-in RTS/CTS flow control.

SET HANDSHAKE {XON, XOFF, CR, LF, BELL, ESC, NONE}
Normally NONE. Otherwise, half-duplex communication line turnaround handshaking is done during file transfer, which means C-Kermit will not reply to a packet until it has received the indicated handshake character or has timed out waiting for it; the handshake setting applies only during file transfer. If you SET HANDSHAKE to other than NONE, then FLOW should be set to NONE.

SET HOST *name*
For communicating over a network rather than a terminal device. Presently supported only for Berkeley-based or other UNIX implementations using the socket interface to a TCP/IP network, for VAX/VMS with the TGV MultiNet TCP/IP package, and for SunLink X.25. For TCP/IP networks, the *name* is the hostname or IP host number of a host on the network, optionally followed by a colon and an IP service number. By default, C-Kermit connects to the telnet (virtual terminal server) socket on TCP/IP connections. For X.25 networks, the *name* is the X.121 address.

SET INCOMPLETE {DISCARD, KEEP}
Disposition for incompletely received files. If an incoming file is interrupted or an error occurs during transfer, the part that was received so far is normally discarded. If you SET INCOMPLETE KEEP then partial files will be kept.

SET INPUT {CASE, ECHO, TIMEOUT-ACTION}
Controls the behavior of the INPUT command. See the section on script programming.

SET KEY *n* [*string*]

Key mapping for CONNECT mode. Tell Kermit that when you press the key whose code is *n* during CONNECT mode, it should substitute the *string* in its place. The string is entered in the native character set of your computer, or you can use backslash codes. The characters are subject to the same translation and shifting rules as the characters you would enter manually at the keyboard during CONNECT mode. The code *n* must be a number between 0 and 255 (higher numbers are possible on some computers, like OS/2, if Kermit knows how to read your computer's keyboard scan codes). A key mapping for code *n* applies to *all* keys that produce this code, for example SET KEY \9 \27 assigns the code 27 (ESC) to the TAB key and to the CTRL-I key. To find out a key's code, enter the SHOW KEY command, then press the key or key combination. Entering the SET KEY for key *n* without specifying a *string* results in removing any previous key mapping for key *n,* i.e. *n* sends itself. Key mappings do not apply to escape-character arguments. If the escape character appears in a SET KEY *string*, it is treated as an ordinary data character, but if a key definition is exactly one character long, and that character is the escape character, it is treated as the escape character.

SET LANGUAGE {ICELANDIC, GERMAN, NORWEGIAN, RUSSIAN, ...}

For use with international text file transfer. If you tell Kermit what language a text file is written in, then Kermit might be able to apply certain transliteration tricks when translating between the file character set and the transfer character set. See the section on international character sets.

SET LINE [*terminal-device-name*]

The device name for the communication line to be used for file transfer and terminal connection, e.g. `/dev/ttyi3` on a UNIX system or `TXA0:` on a VAX/VMS computer. If you specify a device name, Kermit will be in *local mode*, and you should remember to issue any other necessary SET commands, such as SET SPEED (on UNIX, at least, the SET SPEED command is not strictly necessary -- Kermit will use the line's current speed, but you should use SET SPEED anyway, so that you be sure that speed you desire is actually being used).

If you omit the device name, Kermit will revert to its default mode of operation. If you specify the default device name (`/dev/tty` in UNIX, `TT:` in VMS), Kermit will enter remote mode (useful when logged in through the "back port" of a system normally used as a local-mode workstation).

Whenever you give a SET LINE command, C-Kermit closes any currently open communication device before attempting to open the new one. Therefore SET LINE is also useful for closing and hanging up a dialed connection.

When UNIX Kermit enters local mode, it attempts to synchronize with other programs (like uucp) that use external communication lines so as to prevent two programs using the same line at once; before attempting to lock the specified line, it will close and unlock any external line that was previously in use. If your system does not allow you to have write access to the uucp lock directory, then you will receive a message like "Sorry, access to lock denied." In this case, you must ask your system administrator to ensure that uucp lockfiles are set up correctly and Kermit is installed correctly. If Kermit were to use the external line without proper coordination with uucp (and even other copies of Kermit), then two or more users could find themselves using the same line at the same time, which would prevent all useful communication.

If you SET LINE to a communication port that has a modem attached, and you have SET CARRIER ON, then the SET LINE command will not return until carrier appears on the device. This is useful for setting up a Kermit program that other people can dial in to. You can control the amount of time Kermit will wait for carrier using SET CARRIER ON *n*, where n is the number of seconds to wait for a connection before timing out. You can also interrupt a blocked SET LINE command by typing Ctrl-C.

SET MACRO { ECHO, ERROR } { ON, OFF }
: Tells whether the individual commands that comprise a macro should be echoed on the screen during macro execution (normally they are not), and whether an error during macro execution should terminate the macro immediately (normally it does not). See the section on script programming.

SET MODEM-DIALER *name*
: The type of modem dialer on the communication line; the *name* identifies the modem type: DIRECT, HAYES, RACALVADIC, VENTEL, etc. "Direct" indicates either there is no dialout modem, or that if the line requires carrier detection to open, then SET LINE will hang waiting for an incoming call. HAYES, VENTEL, and the others indicate that SET LINE will prepare for a subsequent DIAL command for the given dialer. UNKNOWN means a modem is attached, but of an unknown or unsupported type. Support for new dialers is added from time to time, so type SET MODEM ? for a list of those supported in your copy of Kermit. Also see the description of the DIAL and SET DIAL commands. *NOTE:* the SET MODEM command must be given *before* the SET LINE command if you plan to use the DIAL command.

SET NETWORK {TCP/IP, X.25}
: Select the type of network that is to be used for SET HOST connections. TCP/IP is presently available for UNIX systems that support the Berkeley sockets library (such as BSD-based UNIXes, HP-UX, Xenix with Excelan TCP/IP) and for VAX/VMS with the TGV MultiNet library. X.25 is available only for SUNs with the SunLink product (see section on X.25 support).

SET PAD
: (See X.25 section)

SET PARITY {EVEN, ODD, MARK, SPACE, NONE}
: Specify character parity for use in packets and terminal connection, normally NONE. If other than NONE, C-Kermit will seek to use the 8th-bit prefixing mechanism for transferring 8-bit data, which can be used successfully only if the other Kermit agrees during the automatic feature negotiation phase; if not, 8-bit data cannot be successfully transferred. In Berkeley-based UNIX implementations, if you SET PARITY to other than NONE, this will also enable Xon/Xoff flow control during file transfer if FLOW is set to XON/XOFF; otherwise, flow control is not done (because the communication line must be opened in "raw mode" for 8-bit data, which precludes the use of Xon/Xoff flow control in Berkeley UNIX).

SET PROMPT [*text*]
: The given text will be substituted for "`C-Kermit>`" as this program's prompt. If the text is omitted, the prompt will revert to "`C-Kermit>`". If the text is enclosed in { curly braces }, the braces are stripped and any leading and trailing blanks are retained. The text may contain backslash codes.

SET QUIET {ON, OFF}
: Normally OFF. Various informational messages are issued by the user interface. ON suppresses these messages (but not error messages). Equivalent to '-q' on the command line.

SET RECEIVE *parameter value*
: (See SET SEND.)

SET RETRY *n*
: Specify the maximum number of times a particular packet can be retransmitted (because of timeout or transmission errors) before Kermit gives up and declares the file transfer a failure.

SET {SEND, RECEIVE} *parameter value*
: These commands are used to modify the normal formats and procedures used by the Kermit file transfer protocol. Normally they are not necessary, but they can be used to overcome unusual obstacles, or to improve Kermit's performance.

The SET RECEIVE command lets you tell the other Kermit how it should format the packets it sends to you. Give SET RECEIVE commands to the Kermit that is going to receive files. If you plan to transfer files in both directions, give SET RECEIVE commands to both Kermits.

The SET SEND command rarely needs to be used. It is for overriding what the the other Kermit requests, and should be necessary only if you cannot use a SET RECEIVE command to modify the parameter in question on the receiving Kermit.

SET {RECEIVE, SEND} END-OF-PACKET *cc*
: Specifies the control character that marks the end of a Kermit packet. Normally 13 (carriage return), which most Kermit implementations require.

SET {RECEIVE, SEND} PACKET-LENGTH *n*
: Specify the maximum packet length, normally 90. Shorter packet lengths can be useful on noisy lines, or with systems or front ends or networks that have small buffers. The shorter the packet, the higher the overhead, but the lower the chance of a packet being corrupted by noise, and the less time to retransmit corrupted packets. If you request a length greater than 94, "long packets" are used, which is a feature that not all other Kermit programs support (most popular ones do). C-Kermit can send and receive packets up to about 2000 characters in length. If you use longer packets, you should also request a stronger error checking method (see SET BLOCK-CHECK). SET SEND PACKET-LENGTH overrides the value requested by the other Kermit during protocol initiation unless the other Kermit requests a shorter length.

SET {RECEIVE, SEND} PAD-CHARACTER *cc*
: SET RECEIVE PAD-CHARACTER allows C-Kermit to request the other Kermit to use *cc* as a pad character. Default *cc* is NUL, ASCII 0. C-Kermit normally does not need to have incoming packets preceded with pad characters. SET SEND PAD-CHARACTER designates a character to send before each packet. Normally, none is sent. Outbound padding is sometimes necessary for communicating with slow half duplex systems that provide no other means of line turnaround control. It can also be used to send special characters to communications equipment that needs to be put in "transparent" or "no echo" mode, when this can be accomplished in by feeding it a certain control character.

SET {RECEIVE, SEND} PADDING *n*
: How many pad characters to request or send, normally 0.

SET {RECEIVE, SEND} START-OF-PACKET *number*
: The normal Kermit packet prefix is Control-A (1); this command changes the prefix C-Kermit puts on outbound packets. The only reasons this should ever be changed would be: Some piece of equipment somewhere between the two Kermit programs will not pass through a Control-A; or, some piece of of equipment similarly placed is echoing its input. In the latter case, the recipient of such an echo can change the packet prefix for outbound packets to be different from that of arriving packets, so that the echoed packets will be ignored. The opposite Kermit must also be told to change the prefix for its inbound packets (use SET RECEIVE START on one Kermit and SET SEND START on the other).

SET {RECEIVE, SEND} TIMEOUT *n*
: Normally, each Kermit partner sets its packet timeout interval based on what the opposite Kermit requests. SET RECEIVE TIMEOUT allows you to override the normal procedure and specify a timeout interval for C-Kermit to use when waiting for packets from the other Kermit. If you specify 0, then no timeouts will occur, and C-Kermit will wait forever for expected packets to arrive (relying on the other Kermit to provide the timeout function). SET SEND TIMEOUT specifies the number of seconds to wait for a packet before timing it out and retransmitting or requesting retransmission.

SET SCRIPT ECHO {ON, OFF}
: Tell whether the SCRIPT command should echo its interactions with the remote host on your screen. ON by default.

SET SERVER DISPLAY {ON, OFF}
: Specify whether C-Kermit, when in server mode, should put a file transfer display on the screen when it is in local mode. Normally OFF.

SET SERVER TIMEOUT *n*
: Specify the time interval *n* in seconds for the C-Kermit server to send NAK packets while waiting for a command packet. These NAKs are intended to break deadlocks in case a client Kermit that cannot time out sends a command packet which is lost. However, the server command-wait NAKs can interfere with originate/answer devices that are to be used for answering. For example, you can run a C-Kermit server on a modem line that normally dials out, so that people can dial in to it and give Kermit commands. While waiting for the phone call to come, the server NAKs might "wake up" the modem and put it into originate mode, preventing the incoming call from being answered.

SET SESSION-LOG { BINARY, TEXT }
: Specify how the session log is to be written. TEXT is the default, meaning that lines are written using the convention of the local system; for example, UNIX session logs will have carriage returns (and certain other extraneous characters like NUL) omitted. BINARY means to record all characters in the session log.

SET SPEED *n*
: The transmission speed in bits per second ("baud rate") for the communication line specified in SET LINE (but not SET HOST). This command cannot be used to change the speed of your own console terminal. Some computers are set up in such a way that you must give this command after a SET LINE command before you can use the line. Type SET SPEED ? to see what speeds are available. Use speeds greater than 9600 with caution, since they are not necessarily supported by the communication devices on your computer or the other computer, or the communication path between them.

SET SUSPEND { OFF, ON }
: (UNIX only) On UNIX versions that support job control, C-Kermit can normally be put in the background by typing the suspend character (usually Ctrl-\), or giving C-Kermit the SUSPEND (or Z) command, or by using the Z connect-mode escape. SET SUSPEND OFF disables this feature. This is useful when suspending doesn’t work right (e.g. on certain UNIX implementations that have bugs) or when you are running Kermit under the Bourne shell.

SET TAKE { ECHO, ERROR } { ON, OFF }
: SET TAKE ECHO tells whether commands from a TAKE file are displayed on the screen as they are executed (normally they are not). SET TAKE ERROR controls whether execution of a TAKE command file should be terminated if an error occurs (normally it is not).

SET TERMINAL *parameter value*
: Used for specifying terminal parameters.

  SET TERMINAL BYTESIZE { 7, 8 }
  : tells the character size to be used on the communication line between the local C-Kermit and the remote Kermit during terminal emulation (C-Kermit CONNECT command). It’s 7 by default, which means that the high-order (8th) bit is stripped from each incoming and outgoing character. Use 8 for 8-bit character sets like ISO Latin Alphabet 1, but this will work only if the connection really is 8-bits no-parity. Compare with SET COMMAND BYTESIZE.

SET TERMINAL CHARACTER SET <remote-set> [<local-set>]
For use during CONNECT. Specify the character set that is being sent to C-Kermit by the remote computer, and specify which local character set to translate it into. If the local character set is omitted, the current file character set is used. The choices for the character sets are the same as for C-Kermit's file character sets (see SET FILE CHARACTER-SET). SET TERMINAL CHARACTER-SET TRANSPARENT (which is Kermit's startup default) means no translation is done. Equivalently, no translation is done if the local and remote sets are specified to be the same. When they are different, translation goes through the Latin Alphabet 1 unless the local character set is Cyrillic-based in which case it goes through the Latin/Cyrillic Alphabet. Kanji terminal character-sets are not supported.

SET TERMINAL LOCKING-SHIFT { OFF, ON }
Tells C-Kermit whether to use Shift-In/Shift-Out (Ctrl-O and Ctrl-N) to switch between 7-bit and 8-bit characters during CONNECT. Applies to the Kermit-to-remote part of the connection only, and it applies to both the characters you type and to those Kermit receives from the remote. OFF by default.

SET TERMINAL NEWLINE { ON, OFF }
OFF is the default. When ON, Kermit sends CRLF whenever you type CR during terminal emulation.

SET TERMINAL TYPE { VT100, TEK }
(OS/2 C-Kermit only) Select the type of terminal to emulate.

Terminal parameters can be displayed with the SHOW TERMINAL command.

SET TRANSFER CHARACTER-SET *name*
Tells what character set should be used for file data within Kermit packets during transfer of text files. The choices are:

TRANSPARENT
Don't translate characters at all.

ASCII
Means to use the 7-bit ASCII (American Standard Code for Information Interchange) character set. If the local file character set contains characters that do not occur in ASCII (like accented letters), represent them in ASCII the best way possible.

CYRILLIC-ISO
Means to use the ISO 8859-5 Latin/Cyrillic Alphabet, which is capable of representing Russian, Ukrainian, Bulgarian, Serbian, etc.

LATIN1
Means to use ISO 8859 Latin Alphabet 1, which is capable of representing most western European languages (English, German, Spanish, Dutch, Italian, Norwegian, Danish, Swedish, Portuguese, etc).

JAPANESE-EUC
Means to use Japanese Extended UNIX Code, which is a mixture of 7-bit Roman (Japanese ISO 646, similar to ASCII), single-byte Katakana, and double-byte JIS X 0208 Kanji.

SET TRANSMIT *parameter value*
Controls the behavior of the TRANSMIT command, used for uploading files to computers that don't have Kermit programs. See TRANSMIT. The parameters are:

ECHO { OFF, ON }
Controls whether the characters that are sent to the other computer are also displayed on your screen. The default is ON.

EOF *string*
String to send after sending the file, for example SET TRANSMIT EOF \4 to send a Ctrl-D.

FILL *character*
ASCII value of character to insert into blank lines. Some computers ignore blank lines, others might terminate the upload when they receive a blank line. Use this command to insert a character, such as space (32) or X (88), in any blank line.

LINEFEED { ON, OFF }
Transmit linefeed as well as carriage return at the end of each line. Normally, only CR is sent.

LOCKING-SHIFT { ON, OFF }
Whether to send the locking-shift characters SO (Ctrl-N) and SI (Ctrl-O) around 8-bit characters when transmitting and PARITY is not NONE.

PAUSE *n*
Number of milliseconds (1000 milliseconds = 1 second) to pause after sending each line of a text file, or each character of a binary file (according to SET FILE TYPE). Maximum 1000, default 0.

PROMPT *n*
ASCII value of character to look for from host before sending next line, normally LF (10), in text mode only; 0 means not to wait for any responses from the host -- just send all the characters in a steady stream. This command has no effect in binary mode (i.e. when FILE TYPE is set to BINARY).

Synonym: SET XMIT.

SET UNKNOWN-CHAR-SET { DISCARD, KEEP }
Tells what to do if a file arrives whose Attribute packet announces a transfer character set unknown to C-Kermit.

SET WILDCARD-EXPANSION { KERMIT, SHELL }
(UNIX only) Tells who should expand wildcard characters in SEND and similar commands: KERMIT (the default) or the user's preferred UNIX shell (such as sh, csh, or ksh).

SET WINDOW *n*
Select a window size. This refers to Kermit's sliding window packet transport protocol. Normally, Kermit sends a packet, waits for the reply, then sends the next packet, and so on. This is called "stop and wait" operation, corresponding to a window size of 1, and is supported by all Kermit programs. If you select a window size greater than 1, and if the other Kermit supports sliding windows, then multiple packets (up to the window size) can be sent before any replies are required. This allows file transfer to operate efficiently over connections that have long delays, like over public data networks or through satellites.

With a sufficiently large window size, transmission can be continuous with no pauses or delays. The maximum window size is 31, but sizes greater than 5 or 10 are rarely necessary.

Sliding windows may be used in conjunction with long packets, but the maximum length for packets decreases with increasing window size.

Kermit uses the "selective retransmission" technique, so that if packets are damaged during sliding windows transfers, only the damaged packets are retransmitted.

SET X.25
(See X.25 section)

The SHOW Command

Syntax: `SHOW` *category*

The SHOW command with the default argument of "parameters" displays the values of most of the SET parameters described above. If you type a category name after SHOW, then a more detailed report of parameters within the named category are displayed, for example SHOW COMMUNICATIONS, SHOW PROTOCOL. Type SHOW ? to see a list of the available categories.

1.12. Backslash Notation

\ (backslash) in any command means that what follows is not ordinary text, but rather a code, variable, or function whose value is to be substituted into the command at that point. The character after the backslash identifies which kind of quantity this is:

`%` A user-defined simple (scalar) variable
`&` an array reference
`$` an environment variable
`v` (or `V`) a built-in variable
`f` (or `F`) a function
`d` (or `D`) a decimal (base 10) number
`o` (or `O`) an octal (base 8) number
`x` (or `X`) a hexadecimal (base 16) number
`\` the backslash character itself
`\b`
   (or `\B`) the BREAK signal (OUTPUT command only)
`\l`
   (or `\L`) a Long BREAK signal (OUTPUT command only)
a decimal digit
   a 1-3 digit decimal number.
anything else
   The following character is taken literally.

Variables, arrays, functions, etc, are explained in the next section. Numbers can be expressed in backslash notation in the following ways:

`\{...}`
   A grouped number, braces discarded, e.g. \{17}5 is not the same as \175.

`\`*nnn*
   (1-3 decimal digits) replaced by binary number 0-255, e.g. `\13` represents carriage return (ASCII 13).

`\d`*nnn* or `\D`*nnn* - Same as `\`*nnn*

`\o`*nnn* or `\O`*nnn* (1-3 octal digits) replaced by binary number 0-255.

`\x`*nn* or `\X`*nn* (2 hexadecimal digits) replaced by binary number 0-255.

Numbers expressed in backslash notation are typically used to express nonprintable ASCII characters within character strings, or in commands that want you to enter the ASCII value of a character. Examples:

```
    echo \7Wake up!\7           ; A message with beeps
    echo \27[H\27[J             ; VT100 clear-screen sequence
    set pad-character \o177     ; The ASCII character DEL
```

1.13. Macros, Variables, and Script Programming

C-Kermit's script programming language is based upon that of MS-DOS Kermit, and it is in most ways upwards compatible with it, meaning that many MS-DOS Kermit script programs (or TAKE files in general) will work in C-Kermit with little or no modification. C-Kermit also has a SCRIPT command that provides a terse but cryptic shorthand for automating certain kinds of interactive operations (see Section 1.14).

1.13.1. Variables

A variable name is of the form `\%`*i*, where *i* is a single letter or digit. Letter-variables are different from digit-variables, which are used as macro parameters. The alphabetic case of a letter-variable doesn't matter: `\%a` is the same variable as `\%A`.

All variables have character strings as values. A variable is considered to exist if its value is a string of at least one character in length, otherwise it is "undefined" and does not exist.

A value is given to a variable using the DEFINE or ASSIGN command:

DEFINE *name* [*text*]
: The named variable (or macro, see below) is created, with *text* as a value. If a macro or variable of the given name already exists, its definition is replaced with the new one. The given text is copied into the definition literally. If it contains any variable names, functions references, etc, these are simply copied, rather than evaluated.

ASSIGN *name text*
: The text is evaluated, and then the named variable or macro's value is set to the evaluated text. This differs from DEFINE, which does not evaluate the text, but rather copies it literally. The distinction between ASSIGN and DEFINE doesn't matter unless the text contains variable names. With ASSIGN, the contents of the variable is copied, whereas with DEFINE, the name of the variable is copied.

To illustrate the difference between DEFINE and ASSIGN:

```
def \%a Monday
def \%b Today is \%a
assign \%c Today is \%a
def \%a Tuesday
echo \%b
echo \%c
```

The definition of `\%b` is "`Today is \%a`", so that whenever the value of `\%a` changes, so does the value of `\%b`. The definition of `\%c`, however, is fixed as "`Today is Monday`", because "`Monday`" was the value of `\%a` at the time that `\%c` was assigned.

A variable can be used in any Kermit command simply by referring to its name:

```
echo \%a
```

and variables can be undefined by DEFINEing or ASSIGNing nothing to them, for example:

```
define \%a
```

You can define variables with long names, too:

```
define telephone-number 7654321
```

But you have to refer to them in a special way, `\m(`*name*`)`, e.g.:

```
    echo \m(telephone-number)
```

A special kind of variable is called an *array*. This is simply a list, in which each element has a number. An array reference looks like `\&a[i]`. The `\&` means this is an array element rather than a simple variable, the letter tells which array it is (a, b, c, ..., z), and the brackets enclose an *index* which tells which member of the array is being referred to: 1, 2, 3, etc. The index can be a number, or it can be a variable or function (even another array element) that has a numeric value. Array indices must be (or evaluate to) zero or greater.

Arrays have to be declared before you can use them, so Kermit will know how big the array is and can create storage for it. The command is DECLARE, for example:

```
    DECLARE \&A[100]
```

This tells Kermit to create an array called `\&a` with 100 elements. Once an array is declared, its elements can be used just like simple variables:

```
    DEFINE \&A[3] Tuesday
```

An entire array can be destroyed like this:

```
    DECLARE \&A[0]
```

If you refer to an element of an array that is not declared, the reference is replaced by the empty string.

The array `\&@[]` is predeclared. It contains the command line that the Kermit program was invoked with, one word per array element. The number of elements in the array is given in the built-in variable `\v(args)` (described later). Example:

```
    $ kermit -p e -b 3
```

Here `\&@[0]` is the Kermit program's file specification, like `/usr/local/bin/kermit`; `\&@[1]` is `-p`, `\&@[2]` is `e`, and so on.

The simple variables `\%a..\%z` and the arrays `\&a[]..\&z[]` are all *global*, which means they can be referenced from anywhere in your script program, including from within a TAKE file or macro.

You can list the names and values of all existing global simple variables with the SHOW GLOBALS command, and you can list the names and dimensions of all declared arrays with SHOW ARRAYS.

1.13.2. Macros

A Kermit macro is a list of one or more Kermit commands that can be referred to by a single name. Like variables, macros are created by the DEFINE (or ASSIGN) command:

```
    define ibm set parity mark, set duplex half, set handsh xon
```

and unlike variables, macros do not have weird-looking names; macro names can be ordinary words.

Once a macro is defined, you can execute all of its commands simply by typing the macro's name:

```
    C-Kermit>ibm
```

If you have defined a macro that has the same name as one of Kermit's built-in commands, you will have to insert the word DO before the name in order to execute the macro rather than the built-in command:

```
    C-Kermit>do ibm
```

(you can also use the word DO to invoke any macro). In these examples, "IBM" and "DO IBM" are the *macro invocations*. If you follow a macro invocation by one or more "words" (a word is a sequence of characters delimited by spaces, or else at the end of a line), then each of these words is assigned to a digit-variable that is accessible to the commands within the macro definition. These trailing "words" are called the *macro parameters*. The first parameter is assigned to `\%1`, the second to `\%2`, and so on, up to a maximum of nine. The variable `\%0` contains the name of the macro:

```
define demo echo \%0: \%1 \%2 \%3 \%4 \%5 \%6 \%7 \%8 \%9
demo this is a test of macro argument passing
```

Variables that do not have corresponding parameters are undefined (empty). The number of arguments passed to the macro is available in the named variable \v(argc)[3] Macro parameters are *local* to the macro they are passed to. If macro A invokes macro B, macro B gets its own set of parameters, which are separate from macro A's, so that macro A still has its own parameters available after macro B has completed. The variables `\%0..\%9` may be used as global variables at "top level", when no macros are active, by assigning values to them in the normal way. To demonstrate:

```
define xx echo \%1, yy Sneezy, echo \%1
define yy echo \%1, zz Grumpy, echo \%1
define zz echo \%1
xx Sleepy
```

Try this, and you'll see how the value of `\%1` is saved and restored at each level.

Macro parameter words may be grouped into single parameters by enclosing them in braces. Try the following example to see how this works:

```
define msg echo \%0: \%1
msg this is another test of macro argument passing
msg {this is yet another test of macro argument passing}
```

You can have a macro list its arguments on your screen by including the command SHOW ARGUMENTS within the macro definition.

You can list the names and definitions of all existing macros with the SHOW MACROS command. If you include a name, as in "SHOW MACROS FOO", it will list the definition(s) of only those macros whose names start with the character(s) you have specified.

If you define a macro named ON_EXIT, it will be executed automatically when the Kermit program exits.

1.13.3. Named Variables

Built-in named variables are read-only, you cannot change them. Their names are of the form `\v(name)`, in which the v can be lower or uppercase, and the parentheses are required around the variable name. If you refer to a named variable that does not exist, it will be evaluated as the empty (zero-length) string. C-Kermit's variables are:

`\v(argc)`
: number of arguments passed to currently active macro.

`\v(args)`
: number of arguments passed to the program on the command line. The program argument vector is assigned to the array `\&@[]`.

`\v(count)`
: current value of COUNT (loop control via SET COUNT, IF COUNT).

[3]which, for compatibility with MS-DOS Kermit, may also be referred to simply as ARGC, but only within IF conditions.

`\v(cpu)`
: CPU hardware type, if known, otherwise "unknown".

`\v(date)`
: current date in *dd mmm yyyy* format (e.g. 8 Feb 1990 or 10 Nov 1990)

`\v(directory)`
: current device and/or directory.

`\v(filespec)`
: file specification from most recent SEND, MSEND, or GET command, or the name of the file most recently received.

`\v(fsize)`
: size of last file transferred.

`\v(home)`
: user’s home (login) directory name.

`\v(host)`
: computer hostname.

`\v(input)`
: current INPUT buffer contents.

`\v(line)`
: current communication device or network host.

`\v(ndate)`
: current date in numeric format, e.g. 19901225.

`\v(ntime)`
: current time in seconds since midnight, 0 through 86399.

`\v(platform)`
: name specific machine or environment C-Kermit was built for.

`\v(return)`
: Value of most recent RETURN command.

`\v(speed)`
: Transmission speed of current communication device, if known. (A spurious value of 38400 might be reported for pseudoterminals.)

`\v(status)`
: 0 if the previous command succeeded, nonzero if it failed.

`\v(system)`
: name of generic operating system C-Kermit was built for, such as UNIX or VMS.

`\v(tfsize)`
: total size of all files in the last group of files that was transferred.

`\v(time)`
: current time in hh:mm:ss 24-hour clock format (e.g. 13:45:23).

`\v(ttyfd)`
: file descriptor of communication device (UNIX only).

`\v(version)`
: numeric version number of C-Kermit.

You can use these variables in any Kermit command where their values would make sense:

```
    echo It is \v(time) o’clock on \v(date)
```

A `\v`-variable’s name can be abbreviated, as long as the abbreviation is enough to distinguish it from all the other built-in variable names: `\v(dir)`, `\v(ver)`, etc.

You can get a listing of Kermit’s built-in variables and their values with the SHOW VARIABLES command.

1.13.4. Built-in Functions

Built-in functions are of the form `\f`*name*(*args*). The F and the name can be upper or lower case. The *args* are a comma-separated list of arguments. The function reference is replaced by its value. For example:

```
    define \%a ABC123XYZ
    define \%b ...\Flower(\%a)...
```

results in a value of `...abc123xyz...` for `\%b`.

The names of built-in functions and variables can be abbreviated to their minimum unique length, for example `\feval(1+1)`, `\v(dir)`, etc. C-Kermit’s functions include string-oriented functions that return a string:

`\Fliteral(`*arg*`)`
: Copies its argument literally, without any evaluation.

`\Fcharacter(`*arg*`)`
: Returns the single character corresponding to its argument, which must be numeric. For example, `\fchar(65)` is ’A’, `\fchar(193)` is A-acute (in the Latin-1). If you give a negative number or a number larger than 255, only the low-order 8 bits are used.

`\Fsubstr(`*arg1*`,`*arg2*`,arg3)`
: Substring of the string *arg1* starting at position *arg2*, of length *arg3*. *arg2* and *arg3* must be numbers or variables that have numeric values. Example:

```
        echo \fsubst(hello there,7,5)
```

extracts the word "there". The following is equivalent:

```
        define \%a hello there
        define \%b 7
        define \%c 5
        echo \fsubst(\%a,\%b,\%c)
```

`\Flower(`*arg*`)`
: Converts all uppercase letters in its argument to lowercase, for example:

```
        define \%a FINE
        echo This is a \flower(\%a Mess).
```

prints "This is a fine mess."

`\Fupper(`*arg*`)`
: Converts all lowercase letters in its argument to uppercase.

`\Freverse(`*arg*`)`
: Reverses the order of the characters in its argument, for example `\frev(mupeen)` is `neepum`.

`\Frepeat(`*arg1*`,`*arg2*`)`
Repeats the first argument the number of times given by the second argument, for example:

```
echo +\frep(-+,10)
```

produces `+-+-+-+-+-+-+-+-+-+-+`.

`\Flpad(`*text*`,`*n*`,`*c*`)`
Left-pads the *text* out to length *n* with character *c*. If *c* is omitted, blank (space) is used.

`\Frpad(`*text*`,`*n*`,`*c*`)`
Right-pads the *text* out to length *n* with character *c*. If *c* is omitted, blank (space) is used.

`\Fexecute(`*macroname macro-args*`)`
Execute the named macro with the given parameters (if any), return the macro's RETURN value if any (see RETURN).

`\Fcontents(`*variable-name*`)`
Returns the current definition (contents) of a variable. If the definition includes variable names or function references, these are copied literally, without evaluation.

`\Fdefinition(`*macro-name*`)`
Returns the literal definition of the named macro.

Here are the string functions that return a number:

`\Flength(`*arg*`)`
Returns the length of the argument string.

`\Findex(`*arg1*`,`*arg2*`,`*arg3*`)`
Returns the position of the first occurrence string *arg1* in string *arg2*, starting at position *arg3*. If *arg3* is omitted, then starting at the beginning.

File functions:

`\Ffiles(`*filespec*`)`
Returns the number of files that match the given file specification, for example `\ffiles(ck*.c)`.

`\Fnextfile()`
Returns the next filename that matches the `\Ffiles`() file specification. Use this in a counted loop (see below) after executing `\Ffiles()`.

Integer arithmetic functions:

`\Fmax(`*arg1*`,`*arg2*`)`
Returns the maximum of its two numeric arguments.

`\Fmin(`*arg1*`,`*arg2*`)`
Returns the minimum of its two numeric arguments.

`\Feval(`*expression*`)`
Evaluates the given arithmetic expression. The operands of the expression can be numeric strings or variables, or functions that evaluate to numeric strings. The precedence is the normal, intuitive algebraic (or programming) precedence, and can be altered by the use of parentheses, which have higher precedence than any other operator. Spaces may be used to separate operators from operands, but they are not required.

Table 1-1 shows the types of expressions accepted by `\feval()`.

Operator	Fix	Precedence	Operation	Example
( )		1	Group	`(\%a + 3) * (\%1-5)`
!	Post	2	Factorial	`\%x! - (\%x-2)!`
~	Pre	3	Logical NOT	`~\%n`
-	Pre	3	Negative	`-\%n`
^	In	4	Raise to power	`2^\%p`
*	In	5	Multiply	`\%c * 5`
/	In	5	Divide	`\%c / 5`
%	In	5	Modulus	`\%c % 5`
&	In	5	Logical AND	`\%c & 5`
+	In	6	Add	`\%t + \%u`
-	In	6	Subtract	`27 - \%x`
|	In	6	Logical OR	`\%z | 4`
#	In	6	Exclusive OR	`\%z # 4`
@	In	6	Greatest Common Divisor	`\%z @ 30`

Table 1-1: \feval() Arithmetic Functions

You can list the names of Kermit's built-in functions with the SHOW FUNCTIONS command.

Two Kermit commands are also available to perform simple arithmetic on variables:

INCREMENT *name* [*value*]
: Add *value* to the named variable. If *value* is omitted, add 1. If the variable does not have a numeric value, this command has no effect. Examples: `inc \%a, incr \%b 10`.

DECREMENT *name* [*value*]
: Subtract *value* from the named variable. If *value* is omitted, subtract 1. If the variable does not have a numeric value, this command has no effect.

1.13.5. Script Programming

Kermit's script programming language lets you write procedures, or *programs*, that can include any Kermit command. Variables and functions can be referenced at practically any point in any command. Control structures are provided for decision making, transfer of control, looping, scoping of variables, and so forth. A script program can be a TAKE file, a macro, or any combination of the two.

The basic idea of a script program is to automate tasks that you would normally do "by hand" when interacting with another computer: dialing the modem, negotiating through network boxes and front ends, logging in, etc etc. You can also write programs to manage local files, or to do anything that you could do manually with Kermit.

To automate interaction with another computer, you need a replacement for the CONNECT command. This comes in two parts, an OUTPUT command that "types" what you would type during CONNECT, and an INPUT command that reads the other computer's responses, plus a couple related commands:

OUTPUT *text*
: Send the text to the other computer. The text can be any combination of plain ordinary characters,

backslash codes, variables, and functions. To send a BREAK signal, include `\\B` in the OUTPUT string. To send a Long BREAK signal, include `\\L` in the OUTPUT string.

INPUT *timeout text*
: Read responses from the other computer. Wait up to *timeout* seconds for the specified text to appear. If the text appears within the timeout interval, the command succeeds immediately. Otherwise it fails. The *text* can contain any combination of ordinary characters, backslash codes, variables, and functions. If you want to include leading and/or trailing blanks in the INPUT text, surround it with braces, as in:

```
    input 10 {login: }
```

INPUT can be interrupted with Ctrl-C, in which case it fails.

REINPUT *timeout text*
: Searches previous responses from the computer for the given text. The timeout parameter is ignored. The previous responses are stored in the INPUT buffer, which is 256 characters long, and which may also be accessed via the `\v(input)` variable.

ECHO [*text*]
: Display the text on the local screen, followed by a newline. The text can contain any combination of backslash codes, variables, functions, etc.

CLEAR
: Clear any as-yet-unread characters from the communication device's input buffer.

PAUSE [*n*]
: Do nothing for the indicated number of seconds. If a number is not given, pause for 1 second. If anything is typed on the keyboard during the pause interval, "wake up". PAUSE can be interrupted by typing Ctrl-C (or any other character), in which case it fails.

WAIT [*n* [{ CD, CTS, DSR }]]
: Wait up to *n* seconds for the specified modem signals to appear on the currently selected communication device. If you don't include any modem signal names, WAIT is equivalent to PAUSE. If you include one or more signal names (separated by spaces), Kermit will wait for *all* of the specified signals to appear. If they don't appear within the alloted time, or if the device does not have modem control, then the command will fail. You may interrupt the WAIT command by typing any character at the keyboard, in which case the command will fail. The command will also fail if modem-signal support is not included in C-Kermit for your particular operating system version. The modem signals are: CD (carrier detect), CTS (Clear To Send), and DSR (Data Set Ready).

Example:

```
    wait 10 cd dsr      ; Wait 5 seconds for CD and DSR.
    if success goto ok  ; Test the result.
```

WAIT can be interrupted by typing Ctrl-C (or any other character), in which case it fails.

The behavior of the INPUT command can be controlled by SET INPUT:

SET [INPUT] CASE {IGNORE, OBSERVE}
: Tells whether alphabetic case matters when searching the remote computer's responses for the INPUT text. This setting also affects C-Kermit's other string comparison operations, including IF EQUAL, IF LGT, etc (IF command described later).

SET INPUT ECHO { ON, OFF }
: Tells whether the characters read by the INPUT command should be echoed on the screen.

SET INPUT TIMEOUT-ACTION { PROCEED, QUIT }
: Tells whether the script program should continue or stop immediately if an INPUT command fails.

At this point Kermit needs a decision making mechanism to tell whether the INPUT command succeeded or failed. This is the IF command. Note that IF can be used after INPUT only if you SET INPUT TIMEOUT PROCEED.

IF [NOT] *condition command*
: If the *condition* is satisfied, execute the following Kermit *command*. If the word NOT is included, execute the command if the condition is *not* satisfied. Only one command may be given, and it must appear on the same line as the IF.

The IF command supports a wide variety of conditions. These are:

IF SUCCESS
: The previous command succeeded.

IF FAILURE
: The previous command failed.

IF DEFINED *name*
: The named variable or macro is defined.

IF BACKGROUND
: Kermit is running in the background or with its standard input and output redirected.

IF COUNT
: Subtract one from COUNT, execute the command if the result is greater than zero (see SET COUNT).

IF EXIST *filename*
: The named file exists.

IF NUMERIC *variable*
: The variable’s value is numeric.

IF EQUAL *s1 s2*
: s1 and s2 (character strings or variables) are equal.

IF LLT *s1 s2*
: s1 is lexically (alphabetically) less than s2. Use IF NOT LGT for less-than-or-equal.

IF LGT *s1 s1*
: s1 is lexically (alphabetically) greater than s2. Use IF NOT LLT for greater-than-or-equal.

IF = *n1 n2*
: n1 and n2 (numbers or variables containing numbers) are equal.

IF < *n1 n2*
: n1 is arithmetically less than n2. Use IF NOT > for less-than-or-equal.

IF > *n1 n2*
: n1 is arithmetically greater than n2\n. Use IF NOT < for greater-than-or-equal.

String comparisons in IF EQUAL, IF LGT, and IF LLT treat alphabetic case according to SET INPUT CASE {IGNORE, OBSERVE}. You can also use `\fupper()` and `\flower()` to force caseless comparisons if INPUT CASE is set to OBSERVE.

Arithmetic comparisons work with numeric constants, variables and array elements that evaluate to a numeric string, functions that return a numeric value, and the special "MS-DOS Kermit compatibility" variables COUNT, VERSION, and ARGC, which can only appear in this context. Numeric strings are converted to binary integers before comparison, and may be positive, zero, or negative. Floating point ("real") numbers are not supported.

The IF command may be followed on the next line by an ELSE command, which is executed if the IF condition is not true, and which is not executed if the IF condition is true. Example:

```
IF < \%x 10 ECHO It's less
ELSE echo It's not less
```

As in any programming language, it is desirable to be able to go to different places in the program based on the decisions made by IF commands. The command for going-to is:

GOTO *label*
: Go to the command which follows the named label.

A label is a word beginning with a colon (:) on the left margin. Example:

```
IF < \%x 10 goto less
echo It's not less
goto fin
:less
ECHO It's less
:fin
```

Several other commands are especially useful in conditional contexts, i.e. as objects of IF commands:

STOP
: Stop executing the current script program (TAKE file or macro) and return immediately to C-Kermit prompt level (or, if Kermit was invoked with command-line action arguments, exit C-Kermit altogether).

END [*n*]
: Go "up" one command level. Exit the currently executing macro or TAKE file, and return to the invoking TAKE file or macro. Or, if the current TAKE file or macro was invoked from `C-Kermit>` prompt level, return to the prompt. Or, if Kermit was invoked with command-line action arguments, exit C-Kermit. If a value is given, it is used to set the SUCCESS flag (if the value is zero) or the FAILURE flag (if the value is nonzero). You can use POP as a synonym for this command.

RETURN [*value*]
: For use within macros. Just like END, except that a return value may be set. This is for use with the `\fexecute()` function, which allows you to write user-defined functions. For example:

```
    def sum if = \%1 1 return 1,-
      else return \feval(\%1 + \fexecute(sum \feval(\%1 - 1)))
```

This function, called SUM, returns the sum of all the numbers from 1 to whatever number (1 or greater) it was called with, for example:

```
    echo \fexec(sum 5)
```

prints the number 15. Notice that this function calls itself; C-Kermit functions are allowed to do this.

Although the object command of an IF or ELSE statement may only be a single command, that command is allowed to be a macro invocation or a TAKE command. This provides a kind of statement grouping that can be used to avoid a lot of confusing GOTOs. A more flexible kind of statement grouping is available in the XIF ("extended IF") command:

XIF *condition* { *command-list* } [ELSE { *command-list* }]
: In this extended form, braces are required to enclose a comma-separated list of one or more commands, and the ELSE clause is on the same line as the XIF. Example:

```
    xif < \%x 10 {ech Less,def \%m 10} else {ec Not less,def \%m \%x}
```

Here the appropriate message is echoed, depending on the value of `\%x`, and the minimum of `\%x` and 10 is assigned to the variable `\%m`. Commands like this one can be broken onto multiple lines for clarity, but only if you use dash (or backslash) for line continuation:

```
xif < \%x 10 { -
    echo Less, -
    def \%m 10 -
} else { -
    echo Not less, -
    def \%m \%x -
}
```

WARNING: RETURN or END statements should not be used within an XIF command.

Other Ways of Assigning Values to Variables

Sometimes it is desirable for a script program to interact with you as well as with the remote computer. For example, you should not store passwords on disk, so you can't write script programs that contain them. However, you can write script programs that prompt you for your password. This is done using the ASK or ASKQ command:

ASK *name prompt*

Print the prompt on the screen and wait for you to type a line of text (terminated by carriage return), and assign this line of text to the named variable (but without the carriage return), for example:

```
ask \%p Password:
output \%p\13
```

The prompt may be enclosed in { curly braces } to retain leading and/or trailing spaces.

ASKQ *name prompt*

Just like ASK, but it "asks quietly". The characters you type do not echo. Especially useful for passwords.

GETOK *text*

Asks a "yes or no" question. The *text* is the question. Enclose it in curly braces to preserve leading or trailing spaces. If the user replies "Yes" or "OK" (or any abbreviation of these), the SUCCESS flag is set (for IF SUCCESS). If she replies "N" or "No", the FAILURE flag is set.

Now, with IF, branching, ASK, and so forth, we can add to our login script:

```
define error echo Error: \%1, end  ; Error-handling macro.
set input timeout proceed          ; Don't quit if INPUT fails.
set input case ignore              ; Alphabetic case doesn't matter.
set input echo off                 ; Don't need to watch.
output \13                         ; Send carriage return (ASCII 13).
input 5 login:                     ; Wait 5 secs for "login:" prompt.
if fail error {No login prompt}    ; Give up if it doesn't come.
output myuserid\13                 ; Send user ID followed by CR.
input 5 password:                  ; Wait for password prompt.
if fail error {No password prompt} ; Give up if it doesn't come.
ask \%p Password:                  ; Prompt user for password
output \%p\13                      ; Send it, followed by CR
input 30 $\32                      ; Wait for system prompt "$ "
if fail error {No system prompt}   ; Give up if it doesn't come.
connect                            ; Got it, connect.
```

Reading and Writing Local Files

It is also possible to assign data from local files to variables. In fact, C-Kermit includes an entire input and output system for local files and commands:

OPEN *mode filename*

For use with READ and WRITE commands. Open the local file in the specified mode: READ, WRITE, or APPEND. !READ and !WRITE mean to read from or write to a system command rather than a file. Examples:

```
OPEN READ oofa.txt
OPEN !READ ls ck[uwc].[cwh]
```

The first example reads lines from the file `oofa.txt`. The second reads filenames from a UNIX directory listing; this gives you a way to select files using your UNIX shell's filename-matching metacharacters, which are more powerful than Kermit's built-in ones.

READ *name*

The next line from the current OPEN READ or OPEN !READ file is assigned to the named variable, for example `READ \%A`. If no more lines remain in the file, the command fails.

WRITE *name text*

Writes the given text to the named log or file. The text text may include backslash codes, and is not terminated by a newline unless you include the appropriate code. The name parameter can be any of the following:

DEBUG-LOG
FILE (the OPEN WRITE, OPEN !WRITE, or OPEN APPEND file).
PACKET-LOG
SCREEN (just like ECHO except that WRITE does not supply a line terminator)
SESSION-LOG
TRANSACTION-LOG

CLOSE *name*

Closes the named file. A READ or !READ file is closed automatically when end-of-file (EOF) is encountered during a READ operation, but it does no harm to close it again. WRITE, !WRITE, and APPEND files must be closed explicitly. Examples: CLOSE READ, CLOSE WRITE.

A typical script program for reading and writing files might look like this:

```
set take error off               ; So EOF can be handled.
open read foo                    ; Open input file "foo".
open write bar                   ; Open output file "bar".
def \%c 0                        ; Make a line counter.
:loop                            ; Loop to read all lines.
read \%a                         ; Read one line into \%a.
if fail goto done                ; Catch EOF this way.
increment \%c                    ; Count the line.
write file \%c. \fupp(\%a)\10    ; Format and write it.
goto loop                        ; Go back and get more.
:done
close read                       ; Finished,
close write                      ; close the files.
echo All done.                   ; Print a message.
```

The previous example shows a loop, which is a common programming construct. It is always desirable to avoid infinite loops. The READ example does this by testing for end of file (EOF) with IF FAILURE.

Counted Loops

Kermit also provides for "finite", or counted, loops, in several varieties:

SET COUNT / IF COUNT
: Here a special variable, COUNT, is set. The IF COUNT command decrements the count variable, and if the result is greater than zero, executes the object command:

```
set count 5
:loop
echo \v(count)
if count goto loop
else echo Zero!
```

FOR *variable iv fv step* { *command*, *command*, ... }
: The FOR loop is controlled by a loop *variable* which is assigned an initial value (*iv*), gets tested against a final value (*fv*), and gets incremented by a *step* value. If the step value is positive, the loop exits when the loop variable is greater than the final value. If the step is negative, the loop exits when the variable becomes less than the final value. Example (cute way to print the 9's times table):

```
for \%i 0 9 1 { assign \%j \feval(9-\%i), echo \%i\%j }
```

FOR loops can be nested:

```
for \%i 1 5 1 { for \%j 5 \%i -1 { echo \%i:\%j } }
```

WHILE *condition* { *command*, *command*, ... }
: The commands are executed as long as the *condition* is true. The conditions are the same as for the IF command. Example:

```
while not exist foo.bar { sleep 1 }
```

This loop waits until file named `foo.bar` appears.

FOR and WHILE loops can be nested in any combination. Both FOR and WHILE loops may be exited or continued prematurely using the following commands:

BREAK
: Exit from the closest enclosing FOR or WHILE loop.

CONTINUE
: Begin the next iteration (if any) of the closest enclosing FOR or WHILE loop immediately.

WARNING: RETURN and END commands should not be used in the object commands of FOR, WHILE, or XIF.

Here's a script program example that has nothing to do with data communication or file transfer, but which illustrates the control structures of the script language, the use of arrays, statement grouping, file i/o, etc:

```
; Kermit script program to read a file of numbers into an array,
; sort the array numerically, and display the sorted result.
;
def \%d 100                  ; Allow up to 100 elements.
dcl &a[\%d]                  ; Declare a 100-element array.
set take error off           ; To handle EOF on READ.
open read numbers.dat        ; Open this file for reading.
```

```
    ; Read each line into a separate array element
    ;
    echo Reading...
    for \%n 1 \%d 1 { -
        read \&a[\%n], -
        if fail break -
    }
    decr \%n                        ; Subtract one for read that failed.
                                    ; Check for too many lines.
    xif > \%n \%d { -
        echo Too many lines in file, -
        stop -
    }
    echo Array initialized: \%n elements, sorting...

    ; Sort the array using the (slow) "bubble sort" algorithm.
    ;
    assign \%m \feval(\%n-1)
    for \%i 1 \%m 1 { -
        for \%j \%i \%n 1 { -
            xif > \&a[\%i] \&a[\%j] { -
                assign \%t \&a[\%i], -
                assign \&a[\%i] \&a[\%j], -
                assign \&a[\%j] \%t -
            } -
        } -
    }

    ; Display the sorted array.
    ;
    echo sorted.
    for \%i 1 \%n 1 { echo \flpad(\%i,3). \flpad(\&a[\%i],5) }
```

1.14. The SCRIPT Command

The SCRIPT command allows connection and login sequences to be specified in a single, cryptic line in the style of the UNIX uucp `L.sys` file.

Syntax: `SCRIPT` *expect send* [*expect send*] . . .

"expect" has the syntax: *expect*[-*send*-*expect*[-*send*-*expect*[...]]]

The SCRIPT command carries on a "canned dialog" with a remote system, in which data is sent according to the remote system's responses. The typical use is for logging in to a remote system automatically.

A login script is a sequence of the form:

```
    expect send [expect send] . . .
```

where *expect* is a prompt or message to be issued by the remote site, and *send* is the string (names, numbers, etc) to return, and expects are separated from sends by spaces. The send may also be the keyword EOT, to send Control-D, or BREAK, to send a break signal. Letters in sends may be prefixed by '~' to send special characters, including:

`~b`	backspace
`~s`	space
`~q`	'?'(trapped by Kermit's command interpreter)
`~n`	linefeed
`~r`	carriage return
`~t`	tab

`~'`	single quote
`~-`	dash (hyphen)
`~~`	tilde itself
`~"`	double quote
`~x`	XON (Control-Q)
`~c`	don't append a carriage return
~*o[o[o]]*	octal representation of an ASCII character code
`~d`	delay approx 1/3 second during send
`~w`*[d[d]]*	wait specified interval during expect, then time out
`\\b`	send a BREAK signal

As with some UUCP systems, sent strings are followed by `~r` unless they have a `~c`.

Only the last 7 characters in each expect are matched. A null *expect*, e.g. `~0` or two adjacent dashes, causes a short delay before proceeding to the next send sequence. A null expect always succeeds.

As with UUCP, if the expect string does not arrive, the script attempt fails. If you expect that a sequence might not arrive, as with UUCP, conditional sequences may be expressed in the form:

-send-expect[-send-expect[...]]

where dashed sequences are followed as long as previous expects fail. Timeouts for expects can be specified using `~w`; `~w` with no arguments waits 15 seconds.

expect-send transactions can be easily be debugged by logging transactions. This records all exchanges, both expected and actual. The script execution will also be logged in the session log, if that is activated.

Note that '\' characters in login scripts, as in any other C-Kermit interactive commands, must be doubled up. A line may be ended with a single '\' for continuation.

Example one:

Using a modem, dial a UNIX host site. Expect "login" (...gin), and if it doesn't come, simply send a null string with a `~r`. (Some UNIXes require either an EOT or a BREAK instead of the null sequence, depending on the particular site's "logger" program.) After providing user id and password, respond "x" to a question-mark prompt, expect the Bourne shell "$" prompt (and send return if it doesn't arrive). Then cd to directory kermit, and run the program called "wermit", entering the interactive connect state after wermit is loaded.

```
set modem ventel
set line /dev/tty77
set baud 1200
dial 9&5551212
script gin:--gin:--gin: smith ssword: mysecret ~q x $--$ \
 cd~skermit $ wermit
connect
```

Note that SET LINE is issued *after* SET MODEM, but *before* SET BAUD or other line-related parameters.

Example two:

Using a modem, dial the Telenet network. This network expects three returns with slight delays between them. These are sent following null expects. The single return is here sent as a null string, with a return appended by default. Four returns are sent to be safe before looking for the prompt. Then the Telenet id and password are entered. Then Telenet is instructed to connect to a host site (c 12345). The host has a data switch that asks "which system"; the script responds "myhost" (if the "which system" prompt doesn't appear, the Telenet connect command is reissued). The script waits for an ‘‘`@`’’ prompt from the host, then sends the user ID ("joe") and password ("secret"), looks for another ‘‘`@`’’ prompt, runs Kermit, and in response to the Kermit’s prompt (which ends in ‘‘`>`’’), gives the commands "set parity even" and "server". Files are then exchanged. The commands are in a take file; note the continuation of the ’script’ command onto several lines using the ‘`\`’ terminator.

```
set modem hayes
set line /dev/acu
set speed 1200
set parity mark
dial 9,5551212
script ~0 ~0 ~0 ~0 ~0 ~0 ~0 ~0 @--@--@ id~s\
aa001122 = 002211 @ c~s12345 ystem-c~s12345-ystem \
myhost @ joe~ssecret @ kermit > set~sparity~seven > server
send some.stuff
get some.otherstuff
bye
quit
```

Since these commands may be executed totally in the background, they can also be scheduled. A typical shell script, which might be scheduled by cron, would be as follows (csh used for this example):

```
#
# Keep trying to dial and log onto remote host and exchange
# files, wait 10 minutes before retrying if it fails.
#
cd someplace
while ( 1 )
    kermit < ./tonight.cmd >> nightly.log &
    if ( ! $status ) break
    sleep 600
end
```

File `tonight.cmd` might have two takes in it, for example, one to take a file with the set modem, set line, set baud, dial, and script, and a second take of a file with send/get commands for the remote server. The last lines of `tonight.cmd` should be BYE and QUIT.

1.15. Command Line Operation

A subset of Kermit’s commands is available using UNIX-style command line options, on operating systems that support passing of command-line options to programs through the "argv, argc" mechanism.

The C-Kermit command line syntax mostly conforms to the Proposed Syntax Standards for UNIX System Commands put forth by Kathy Hemenway and Helene Armitage of AT&T Bell Laboratories in *UNIX/World*, Vol.1, No.3, 1984. The rules that apply are:

- Command names must be between 2 and 9 characters ("kermit" is 6).
- Command names must include lower case letters and digits only.
- An option name is a single character.
- Options are delimited by ’`-`’.
- Options with no arguments may be grouped (bundled) behind one delimiter.
- Option-arguments cannot be optional.
- Arguments immediately follow options, separated by whitespace.
- The order of options does not matter.
- ’`-`’ preceded and followed by whitespace means standard input.

A group of bundled options may end with an option that has an argument.

The following notation is used in command descriptions:

fn	A file specification, possibly containing the "wildcard" characters `*` or `?` (`*` matches all character strings, `?` matches any single character).
fn1	A UNIX file specification which may not contain `*` or `?`.
rfn	A remote file specification in the remote system's own syntax, which may denote a single file or a group of files.
rfn1	A remote file specification which should denote only a single file.
n	A decimal number between 0 and 94.
c	A decimal number between 0 and 127 representing the value of an ASCII character.
cc	A decimal number between 0 and 31, or else exactly 127, representing the value of an ASCII control character.
`[ ]`	Any field in square braces is optional.
`{x,y,z}`	Alternatives are listed in curly braces.

C-Kermit command line options may specify any combination of actions and settings. If C-Kermit is invoked with a command line that specifies no actions, then it will issue a prompt and begin interactive dialog. Action options specify either protocol transactions or terminal connection.

An implicit TAKE command is executed upon your C-Kermit initialization file when C-Kermit starts up, upon either interactive or command-line invocation. This file may contain C-Kermit interactive-mode commands, which are explained later.

In addition, if the first command line argument is the name of an existing file, C-Kermit will read that file and execute Kermit interactive-mode commands from it, in addition to (and after) your initialization file (if any). Example:

```
$ kermit cmdfile -b 3 -p m
```

In UNIX, this feature allows you to invoke C-Kermit by "running" a C-Kermit command file, if its first line is of the form:

```
#!/dir/kermit
```

That is, number sign followed by exclamation point, followed by the full file specification of the Kermit program. You must also give this file execute permission:

```
$ chmod +x cmdfile
```

Then you can run this file as if it were a program, and it will feed itself to C-Kermit automatically:

```
$ cmdfile
```

You can even include C-Kermit command line options:

```
$ cmdfile -l /dev/ttyh8 -p e
```

If you want the shell prompt to return automatically at the end of this file, include an EXIT command at the end of it. Otherwise you will receive the C-Kermit prompt when the file is finished.

Here are C-Kermit's command-line options:

`=` Ignore all command-line options that follow (but make them available to the user in the array `\&@[]`).

`-Y` (uppercase) Do not execute commands from the initialization file.

`-y` *fn*
Read commands from the specified file instead of the normal initialization file.

`-C "`*command, command,* `..."`
Execute the interactive-mode commands after the initialization file (if any), the other command-line options (if any), and the command or application file (if any). The command list must be enclosed in double quotes. Example:

```
kermit -C "set file type binary, set window 2"
```

The commands in this list are assigned to a macro called `cl_commands`, so you can also execute them later during your session simply by typing the name of this macro:

```
C-Kermit>cl_commands
```

The -C option does is not considered an "action" option, even if the command list contains action commands, so if there are no other action commands among the command-line arguments, the C-Kermit prompt appears after the last command in the list is executed. To defeat this behavior, include EXIT as the last command:

```
kermit -C "echo \27[2J\27H, exit"
```

`-S` ("Stay") This option tells C-Kermit to issue its prompt and enter interactive command mode, even if the command-line options included action commands.

These are the file-transfer commands:

`-s` *fn*
Send the specified file or files. If *fn* contains wildcard (meta) characters, the UNIX shell expands it into a list. On non-UNIX systems, Kermit expands any wildcard characters. *fn* may also be a list of files, as in:

```
kermit -s ckcmai.c ckuker.h mail.txt
```

If *fn* is '`-`' then kermit sends from standard input, which may come from a file:

```
kermit -s - < foo.bar
```

or a parallel process:

```
ls -l | grep cmg | kermit -s -
```

You cannot use this mechanism to send terminal typein. If you want to send a file whose actual name is ‘‘`-`’’ you can precede it with a path name, as in

```
kermit -s ./-
```

`-r` Receive a file or files. Wait passively for files to arrive.

`-k` Receive (passively) a file or files, sending them to standard output. This option can be used in several ways:

`kermit -k`
Displays the incoming files on your screen; to be used only in "local mode" (see below).

`kermit -k >` *fn1*
Sends the incoming file or files to the named file, *fn1*. If more than one file arrives, all are concatenated together into the single file *fn1*.

`kermit -k | command`
Pipes the incoming data (single or multiple files) to the indicated command, as in

```
kermit -k | sort > sorted.stuff
```

`-a` *fn1*
If you have specified a file transfer option, you may give an alternate name for a single file with the `-a` ("as") option. For example,

```
kermit -s foo -a bar
```

sends the file `foo` telling the receiver that its name is `bar`. If more than one file arrives or is sent, only the first file is affected by the `-a` option:

```
kermit -ra baz
```

stores the first incoming file under the name `baz`.

`-x` Begin server operation. May be used in either local or remote mode.

Before proceeding, a few words about remote and local operation are necessary. C-Kermit is "local" if it is running on PC or workstation that you are using directly, or if it is running on a multiuser system and transferring files over an external communication line -- not your job's controlling terminal or console. C-Kermit is remote if it is running on a multiuser system and transferring files over its own controlling terminal's communication line (normally `/dev/tty`), connected to your PC or workstation.

If you are running C-Kermit on a PC, it is normally used in local mode, with the "back port" designated for file transfer and terminal connection, and the keyboard and screen available to control or interrupt the file transfer and to display its status. If you are running C-Kermit on a multiuser (timesharing) system, it is in remote mode unless you explicitly point it at an external line for file transfer or terminal connection. The following command determines whether C-Kermit is in local or remote mode:

`-l` *dev*
Line -- Specify a terminal line to use for file transfer and terminal connection, as in:

```
kermit -l /dev/ttyi5
```

You can also give an open file descriptor for a serial communication (tty) device.

When an external line is being used, you will also need some additional options for successful communication with the remote system:

`-b` *n*
Bits per second -- Specify the transmission speed in bits per second ("baud rate") for the line given in the `-l` option, as in:

```
kermit -l /dev/ttyi5 -b 9600
```

This option should always be included with the `-l` option, since the speed of an external line is not necessarily what you expect.

`-p` *x*
Parity -- e,o,m,s,n (even, odd, mark, space, or none). If parity is other than none, then Kermit's 8th-bit prefixing mechanism will be used for transferring 8-bit binary data, provided the opposite Kermit agrees. The default parity is none.

`-t` Specifies half duplex, line turnaround with XON as the handshake character, used mainly for communicating with IBM mainframes in linemode.

`-m`*name*
: Modem type -- hayes, penril, vadic, etc. For a complete and up-to-date list, run C-Kermit interactively and type "set modem ?". Note: name must be given in lowercase, but can be abbreviated, e.g. "hay" for "hayes".

If you wish to use a network connection rather than a terminal line for communication, use the following options rather -l:

`-j` *name*
: Host -- Specify a network host name. This option is currently supported for TCP/IP Telnet connections on Berkeley-based UNIX implementations only, and for VAX/VMS when equipped with TGV MultiNet TCP/IP. The speed (-b) parameter does not apply to network connections. The name can be an IP host name, an IP host number (containing dots), or either one of these followed by a colon and then a TCP service name or number. It can also be just a number, in which case it is assumed to be a file descriptor for an open TCP/IP TELNET connection.

`-X` *address*
: X.25 address -- Specify an X.25 network address.

`-Z` *number*
: X.25 file descriptor -- Specify a file descriptor for an X.25 connection that is already open.

The following commands may be used only with a C-Kermit which is in local mode.

`-g` *rfn*
: Actively request a remote server to send the named file or files; *rfn* is a file specification in the remote host's own syntax. If *fn* happens to contain any special shell characters, like space, '`*`', '`[`', '~', etc, these must be quoted, as in:

```
kermit -g x\*.\?
```

or:

```
kermit -g "profile exec"
```

`-f` Send a 'finish' command to a remote server.

`-c` Establish a terminal connection over the specified or default communication line, before any protocol transaction takes place. Get back to the local system by typing the escape character (normally Control-Backslash) followed by the letter 'c'.

`-n` Like `-c`, but *after* a protocol transaction takes place; `-c` and `-n` may both be used in the same command. The use of `-n` and `-c` is illustrated below.

If the other Kermit is on a remote system, the `-l` and `-b` options should also be included with the `-r`, `-k`, or `-s` options.

Several other command-line options are provided:

`-i` Specifies that files should be sent or received exactly "as is" with no conversions. This option is necessary for transmitting binary files. It may also be used in UNIX-to-UNIX transfers (it must be given to *both* UNIX Kermit programs), where it will improve performance by circumventing the normal text-file conversions, and will allow mixture of text and binary files in a single file group. But only use it if character-set translation is not needed.

`-w` Writeover -- An incoming file should write over any existing file of the same name. Changes the default behavior, which is to back up the existing file before creating the new one.

`-e` *n*
: Extended packet length -- Specify that C-Kermit is allowed to receive packets up to length *n*, where *n* may be between 10 and some large number, like 1000 or 2000, depending on the system. The default maximum length for received packets is 90. Packets longer than 94 will be used only if the other Kermit supports, and agrees to use, the "long packet" protocol extension, and if you request them.

`-v` *n*
: Window size -- Specify that C-Kermit is allowed to send and receive files using a window size of *n*. That is, up to *n* packets may be sent before acknowledgements are required. This speeds up transfers in most situations, especially long distance network connections. Default window size is 1, maximum is 32. Sizes greater than 1 work only if the other Kermit supports this option.

`-q` Quiet -- Suppress screen update during file transfer, for instance to allow a file transfer to proceed in the background.

`-z` Force foreground mode: even if Kermit thinks it is running in the background, this will make it issue its normal prompts and messages. Example:

```
kermit -z | vt100
```

`-d` Debug -- Record debugging information in the file `debug.log` in the current directory. Use this option if you believe the program is misbehaving, and show the resulting log to your local Kermit maintainer.

`-h` Help -- Display a brief synopsis of the command line options.

The command line may contain no more than one protocol action option.

During transmission, files are encoded as follows:

- Control characters are converted to prefixed printables.
- Sequences of repeated characters are collapsed via repeat counts, if the other Kermit is also capable of repeated-character compression. This feature is negotiated automatically.
- If parity is being used on the communication line, data characters with the 8th (parity) bit on are specially prefixed, provided the other Kermit is capable of 8th-bit prefixing; if not, 8-bit binary files cannot be successfully transferred. This feature is negotiated if you give the `-p` option.
- Conversion is done between UNIX newlines and carriage-return-linefeed sequences unless the `-i` option was specified.

Command Line Examples:

```
kermit -l /dev/ttyi5 -b 1200 -cn -r
```

This command connects you to the system on the other end of `ttyi5` at 1200 baud, where you presumably log in and run Kermit with a SEND command. After you escape back, C-Kermit waits for a file (or files) to arrive. When the file transfer is completed, you are reconnected to the remote system so that you can logout.

```
kermit -l /dev/ttyi4 -b 1800 -cntp m -r -a foo
```

This command is like the preceding one, except the remote system in this case uses half duplex communication with mark parity. The first file that arrives is stored under the name `foo`.

```
    kermit -l /dev/ttyi6 -b 9600 -c | tek
```

This example uses Kermit to connect your terminal to the system at the other end of `ttyi6`. The C-Kermit terminal connection does not provide any particular terminal emulation, so C-Kermit's standard i/o is piped through a (hypothetical) program called tek, which performs (say) Tektronix emulation.

```
    kermit -l /dev/ttyi6 -b 9600 -nf
```

This command would be used to shut down a remote server and then connect to the remote system, in order to log out or to make further use of it. The `-n` option is invoked *after* `-f` (`-c` would have been invoked before).

```
    kermit -l /dev/ttyi6 -b 9600 -qg foo.\* &
```

This command causes C-Kermit to be invoked in the background, getting a group of files from a remote server (note the quoting of the '`*`' character). No display occurs on the screen, and the keyboard is not sampled for interruption commands. This allows other work to be done while file transfers proceed in the background. UNIX only.

```
    kermit -l /dev/ttyi6 -b 9600 -g foo.\* > foo.log < /dev/null &
```

This command is like the previous one, except the file transfer display has been redirected to the file `foo.log`. Standard input is also redirected, to prevent C-Kermit from sampling it for interruption commands.

```
    kermit -iwx
```

This command starts up C-Kermit as a server. Files are transmitted with no newline/carriage-return-linefeed conversion; the `-i` option is necessary for binary file transfer and recommended for UNIX-to-UNIX transfers. Incoming files that have the same names as existing files are given new, unique names.

```
    kermit -l /dev/ttyi6 -b 9600
```

This command sets the communication line and speed. Since no action is specified, C-Kermit issues a prompt and enters an interactive dialog with you. Any settings given on the command line remain in force during the dialog, unless explicitly changed.

```
    kermit
```

This command starts up Kermit interactively with all default settings.

The next example shows how UNIX Kermit might be used to send an entire directory tree from one UNIX system to another, using the tar program as Kermit's standard input and output. On the originating system, in this case the remote, type (for instance):

```
    tar cf - /usr/fdc | kermit -is -
```

This causes tar to send the directory `/usr/fdc` (and all its files and all its subdirectories and all their files...) to standard output instead of to a tape; kermit receives this as standard input and sends it as a binary file. On the receiving system, in this case the local one, type (for instance):

```
    kermit -il /dev/ttyi5 -b 9600 -k | tar xf -
```

Kermit receives the tar archive, and sends it via standard output to its own copy of tar, which extracts from it a replica of the original directory tree.

A final example shows how a UNIX compression utility might be used to speed up Kermit file transfers:

```
compress file | kermit -is -     (sender)
kermit -ik | uncompress          (receiver)
```

1.16. UNIX Specifics

1.16.1. The UNIX File System

Consult your UNIX manual for details about the file system under your version of UNIX. In general, UNIX files have lowercase names, possibly containing one or more dots or other special characters. UNIX directories are tree-structured. Directory levels are separated by slash (‘‘/’’) characters. For example,

```
/usr/foo/bar
```

denotes the file `bar` in the directory `/usr/foo`. Alphabetic case is significant in UNIX file and directory names, i.e. ‘‘`a`’’ is a different file (or directory) from ‘‘`A`’’. Wildcard or "meta" characters allow groups of files to be specified. ‘‘`*`’’ matches any string; ‘‘`?`’’ matches any single character.

When C-Kermit is invoked with file arguments specified on the UNIX command line, the UNIX shell (Bourne Shell, C-Shell, K-Shell, etc) expands the meta characters itself, and in this case a wider variety is available. For example,

```
kermit -s ~/ck[uvm]*.{upd,bwr}]
```

is expanded by the Berkeley C-Shell into a list of all the files in the user’s home directory (`~/`) that start with the characters "`ck`", followed by a single character ‘‘`u`’’, ‘‘`v`’’, or ‘‘`m`’’, followed by zero or more characters, followed by a dot, followed by one of the strings ‘‘`upd`’’ or ‘‘`bwr`’’. Internally, the C-Kermit program itself expands only the ‘‘`*`’’ and ‘‘`?`’’ meta characters.

UNIX files are linear (sequential) streams of 8-bit bytes. Text files consist of 7-bit ASCII or ISO-646 NRC characters, with the high-order bit off (0), or in very recent versions of UNIX, of 8-bit characters in an ASCII-based international standard character set such as ISO 8859 Latin Alphabet 1. Text file lines are separated by the UNIX newline character, which is linefeed (LF, ASCII 10). This distinguishes UNIX text files from those on most other ASCII-based stream file systems, in which lines are separated by a carriage-return linefeed sequence (CRLF, ASCII 13, followed by linefeed, ASCII 10). Binary files are likely to contain data in the high bits of the file bytes, and have no particular line or record structure.

1.16.2. UUCP Lock Files

UNIX has no built-in way of obtaining exclusive access to an external communication line. When you issue the ’set line’ command to UNIX Kermit, UNIX would normally grant you access to the line even if some other process is making use of it. The method adopted by most UNIX systems to handle this situation is the "UUCP lock file". UUCP, the UNIX-to-UNIX Copy program, creates a file in its directory (usually `/usr/spool/uucp`, but often elsewhere) with a name like `LCK..`*name*, where *name* is the device name, for instance `LCK..tty07`.

UNIX Kermit uses UUCP lock files in order to avoid conflicts with UUCP, tip, or other programs that follow this convention, including other users running Kermit itself. Whenever you attempt to access an external line using the ’set line’ command or ‘`-l`’ on the command line, Kermit looks in the UUCP directory for a lock file corresponding to that device. For instance, if you ’set line /dev/ttyi6’ then Kermit looks for the file

```
/usr/spool/uucp/LCK..ttyi6
```

If it finds this file, it gives you an error message and a directory listing of the file so that you can see who is using it, e.g.

```
    -r--r--r--  1 fdc    8 Feb  7 13:02 /usr/spool/uucp/LCK..ttyi6
```

In this case, you would look up user fdc to find out how soon the line will become free.

This convention requires that the uucp directory be publicly readable and writable. If it is not, the program will issue an appropriate warning message.

If no lock file is found, UNIX Kermit will attempt create one, thus preventing anyone who subsequently tries to run Kermit, UUCP, cu, tip, or similar programs on the same line from gaining access until you release the line. If Kermit could not create the lock file (for instance because the uucp directory is write-protected), then you will receive an error message.

Even when the lock directory is writable and readable, the locking mechanism depends upon all users using the same name for the same device. If a device has more than one path associated with it, then a lock can be circumvented by using an alias.

When a lock-creating program abruptly terminates, e.g. because it crashes or is killed via shell command, the lock file remains in the uucp directory, spuriously indicating that the line is in use. If the lock file is owned by yourself, you may remove it. Otherwise, you'll have to get the owner or the system manager to remove it, or else wait for a system task to do so; uucp supports a function (uuclean) which removes these files after a predetermined age -- uucp sites tend to run this function periodically via crontab.

Locking is not needed, or used, if communications occur over the user's login terminal line (normally `/dev/tty`).

1.16.3. C-Kermit under Berkeley or System III/V UNIX

C-Kermit may be interrupted at command level or during file transfer by typing Control-C. The program will perform its normal exit function, restoring the terminal and releasing any lock. If a protocol transaction was in progress, an error packet will be sent to the opposite Kermit so that it can terminate cleanly.

C-Kermit may be invoked in the background ("&" on shell commmand line). If a background process is "killed", the user will have to manually remove any lock file and may need to restore the modem. This is because the kill signal (`kill(`*x*`,9)`) cannot be trapped by Kermit.

During execution of a system command ('directory', 'cd', or '`!`'), C-Kermit can often be returned to command level by typing a single Control-C. (With System III/V, the usual interrupt function (often the DEL key) is replaced by Control-C.)

Under Berkeley UNIX only: C-Kermit may also be interrupted by `^Z` to put the process in the background.

Control-C, Control-Z, and Control-\ lose their normal functions during terminal connection and also during file transfer when the controlling tty line is being used for packet i/o.

If you are running C-Kermit in "quiet mode" in the foreground, then interrupting the program with a console interrupt like Control-C will not restore the terminal to normal conversational operation. This is because the system call to enable console interrupt traps will cause the program to block if it's running in the background, and the primary reason for quiet mode is to allow the program to run in the background without blocking, so that you can do other work in the foreground.

If C-Kermit is run in the background ("&" on shell commmand line), then the interrupt signal (Control-C) (and System III/V quit signal) are ignored. This prevents an interrupt signal intended for a foreground job (say a compilation) from being trapped by a background Kermit session.

1.16.4. C-Kermit on the AT&T UNIX PC

For UNIX PC owners here are a couple of hints. The name of the phone line devices are `/dev/ph0` and `/dev/ph1`. The RS232 serial port is `/dev/tty000`.

Dialing out with the internal modem:

```
    C-Kermit>set line /dev/ph0
    C-Kermit>set speed 1200
    C-Kermit>set modem att7300
    C-Kermit>dial (123) 555-1212
```

Or use `/dev/ph1` for the second phone line. Control-C will terminate the dialer. The telephone line must be in the DATA state; C-Kermit will remind you of this if it finds the line in VOICE state.

Connecting via the RS232C serial port:

```
    $ kermit
    C-Kermit>set line /dev/tty000
    C-Kermit>set speed 9600
        (start doing work...)
        (after exiting C-Kermit you may type:)
```

C-Kermit automatically handles disabling and enabling logins on the phone lines and RS-232 port.

1.17. X.25 Support

This section applies only to UNIX C-Kermit on SUN computers that are attached to X.25 networks via the SunLink X.25 product.

To use an X.25 connection:

```
    C-Kermit>set network x.25
    C-Kermit>set host aaaaaaa
    C-Kermit>connect
```

where aaaaaaa is the X.121 address of the host you wish to connect to. An X.121 address is a many-digit number consisting of a 4-digit DNIC (Data Network Identification Code) followed by an NTN (Network Terminal Number) up to 10 digits in length, or a 3-digit DCC (Data Country Code) followed by a country-dependent NN (National Number) up to 11 digits in length.

The following connect-mode escapes apply to X.25 terminal connections:

Ctrl-\H
    Hangup - Close the X.25 network connection.

Ctrl-\I
    Interrupt - Send an X.25 interrupt packet.

Ctrl-\R
    Reset the X.25 virtual circuit.

The following special commands can be used to control your X.25 connection:

HANGUP
: Close the X.25 network connection.

PAD CLEAR
: Clear the X.25 virtual circuit.

PAD INTERRUPT
: Send an X.25 Interrupt packet.

PAD RESET
: Reset the X.25 virtual circuit.

PAD STATUS
: X.25 status request.

SET PAD BREAK-ACTION *n*
: X.3 Parameter 7. What PAD should do if it receives a BREAK signal. 0 = nothing, 1 = send Interrupt packet, 2 = reset, 4 = send Indication Of Break PAD message, 8 = escape to PAD, 16 = discard output. Default = 21 (= 16 + 4 + 1).

SET PAD BREAK-CHARACTER *n*
: Default 0.

SET PAD LINE-DELETE *n*
: X.3 Parameter 17. 0-127 = ASCII value of character to be used for erasing a line. Default = 21 (Ctrl-U).

SET PAD CHARACTER-DELETE *n*
: X.3 Parameter 16. 0-127 = ASCII value of character to be used for erasing a character. Default = 8 (Ctrl-H = Backspace).

SET PAD CR-PADDING *n*
: X.3 Parameter 9, Padding After Carriage Return (CR). 0-255. Number of padding characters PAD should send to DTE after sending a CR. Default = 0.

SET PAD DISCARD-OUTPUT { 0, 1 }
: X.3 Parameter 8. 0 = normal data delivery, 1 = discard output. Default = 0.

SET PAD LINE-DISPLAY *n*
: X.3 Parameter 18. 0-127 = ASCII value of character to redisplay an edited line. Default = 18 (Ctrl-R).

SET PAD ECHO { 0, 1 }
: X.3 Parameter 2. 0 = PAD will not echo, 1 = PAD will echo. Default = 1.

SET PAD EDITING { 0, 1 }
: X.3 Parameter 15. 0 = No editing, 1 = editing allowed. Default = 1.

SET PAD ESCAPE { 0, 1 }
: X.3 Parameter 1. 0 = Escape to PAD not possible, 1 = Ctrl-P escapes to PAD.

SET PAD FORWARD *n*
: X.3 Parameter 3, Data Forwarding Characters. 0 = none, 1 = any alphanumeric, 2 = carriage return, etc. Default = 2.

SET PAD LF-PADDING *n*
: X.3 Parameter 14. 0-255 padding characters to be sent by PAD after linefeed. Default = 0.

SET PAD LF-INSERT *n*
: X.3 Parameter 13, Linefeed (LF) insertion after carriage return (CR). 0 = no LF insertion, 1 = PAD inserts LF after each CR sent to DTE, 2 = PAD inserts LF after each CR received from DTE, 4 = PAD echoes LF as CRLF. Default = 0.

SET PAD LINE-FOLD *n*
: X.3 Parameter 10, Line Folding. 0 = none, 1-255 = number of graphic characters per line after which to insert folding characters. Default = 0.

SET PAD PAD-FLOW-CONTROL { 0, 1 }
: X.3 Parameter 5. 0 = No flow control by PAD, 1 = PAD may send Xon/Xoff flow control to user. Default = 0.

SET PAD SERVICE-SIGNALS { 0, 1 }
: X.3 parameter 6, PAD Service and Command Signals. 0 = PAD service signals are not sent to DTE, 1 = PAD service signals sent. Default = 1.

SET PAD TIMEOUT *n*
: X.3 Parameter 4, Data forwarding timeout, 0-255 (twentieths of a second). Default = 0 (no data forwarding on timeout).

SET PAD USER-FLOW-CONTROL { 0, 1 }
: X.3 Parameter 12. 0 = no flow control by user, 1 = user device may send Xon/Xoff flow control to PAD. Default = 0.

SET X.25 CALL-USER-DATA { ON [text], OFF }
: OFF = no call user data, ON *text* sends the given text during call setup.

SET X.25 CLOSED-USER-GROUP { OFF, ON *n* }
: OFF = no closed user group, ON *n* specifies the user group number, 0 to 99. OFF by default.

SET X.25 REVERSE-CHARGE { OFF, ON }
: OFF = caller pays, ON = callee pays. Default is OFF.

1.18. C-Kermit under VAX/VMS

This section contributed by Terry Kennedy, St. Peter’s College, Jersey City, NJ, USA.

1.18.1. File Transfer Issues

VMS provides a plethora of file formats, most of which have no direct equivalent on other operating systems (especially the other operating systems that C-Kermit runs under). Thus, the user is faced with the issue of file format conversion. VMS C-Kermit attempts to make this as painless as possible by mapping the various file formats to canonical form on transmission, and by creating the simplest file structure upon reception. Various C-Kermit commands may be used to give C-Kermit "hints" about what the user desires. Some of these work only when sending files, others only when receiving files. First, let’s discuss some terms:

BINARY TRANSFER MODE (SET FILE TYPE BINARY)

When sending a file, C-Kermit/VMS reads bytes from the file and sends them without any kind of interpretation, translation, or reformatting. In particular, no characters are inserted at record boundaries.

When receiving files, C-Kermit/VMS collects bytes and writes them to disk at intervals. The file is created as "Fixed" organization, with the record size specifed by the user with SET FILE

RECORD-LENGTH, or the default value of 512 if the user does not override it. For both send and receive operations, the RMS attribute "First free byte" is used. Thus, VMS C-Kermit can operate on files where the file size (in bytes) is not an exact multiple of the file record size.

TEXT TRANSFER MODE (SET FILE TYPE TEXT)

When sending files, C-Kermit/VMS attempts to convert the file to a form which has the proper delimiters for "canonical Kermit protocol" form. Specifically, CR and LF characters are inserted as needed at the end of each record. This is normally an easy task, but is complicated by some VMS file types (such as the "Fortran carriage control" type). On reception, files are created as "Sequential variable, carriage return carriage control". C-Kermit uses a CR followed immediately by a LF as a record delimiter. Lone CR's or LF's are written as part of a record if they are present.

LABELED TRANSFER MODE (SET FILE TYPE LABELED)

When in this mode, C-Kermit/VMS gathers information about the file from various sources, such as the file's directory entry, ACL's, etc. and transmits it as part of the file data. This is called a LABELED FILE. Such files are not usable on non-VMS systems, but may be stored on such systems and later restored to another (or the same) VAX. An external utility named CKVCVT reconstitutes the file as it was, with various options.

Other systems should treat labeled files as simple files of TYPE BINARY.

Now, let's discuss the actions of sending or receiving:

1.18.1.1. Sending Files from VMS C-Kermit

The requested file is opened and the VMS file characteristics are examined. Files of format "Undefined" or "Fixed" are sent as binary files, other formats are sent as text files.

SET FILE TYPE IMAGE. This forces the file to be sent as a stream of bytes, regardless of file characteristics, with a file type of binary declared in the attribute packet. This is useful for cases where other applications generate file contents with inappropriate file characteristics (for example, a stream-LF file which is actually binary data). This should only be used as a last resort, as [normally] RMS inserts bytes in the file for control purposes, and these bytes are transmitted (and misinterpreted by the receiving system).

SET FILE TYPE LABELED forces the file to be sent in labeled format, with C-Kermit reading the file from the disk as in IMAGE mode, and with a file type of binary declared in the attribute packet.

1.18.1.2. Receiving Files to VMS C-Kermit

The characteristics supplied in the file's attribute packet are used to determine the desired file type. If no attribute packet arrives, the current SET FILE TYPE setting, TEXT or BINARY, is used. If the current file type is IMAGE, BINARY is used. SET FILE TYPE UNDEFINED forces the received file to be written with RMS RECORD FORMAT UNDEFINED for compatbility with some software that uses non-RMS routines for manipulating files. This option should normally not be used, as you will generate files VMS can't do anything with.

If the current FILE TYPE is LABELED, the incoming file is handled in LABELED mode if (a) its attribute packet declares its type to be binary, OR (b) there is no attribute packet. If the attribute packet says the file type is text, C-Kermit rejects the file. At the present time, C-Kermit does not understand received labeled files, and you should use the external CKVCVT utility to convert them.

1.18.2. PRACTICAL APPLICATIONS

1.18.2.1. Between a VMS System and a Different System

The defaults should function properly, provided that the receiving system uses the "stream-of-bytes" philosophy of UNIX. Extensive testing has been done between VMS C-Kermit and UNIX C-Kermit, as well as MS-DOS Kermit. Other systems may require a re-thinking of the transfer strategy, but this author [tmk] considers it unlikely.

1.18.2.2. Between VMS Systems

The goal of the defaults is to produce useful output in the majority of cases. If you need to transfer binary files where record boundaries are important and don't occur at fixed intervals (such as .OBJ files), or where the file semantics are extensive and not propagated by C-Kermit (such as DDIF files, indexed files, or files with extended semantics stored in an ACE, such as PCSA file service contents), you have two options.

1. SET FILE TYPE LABELED. Give this command to the sending C-Kermit system, receive the file on the target VMS system, and extract the contents with CKVCVT.

2. Encapsulate these files using VMS BACKUP, transfer the BACKUP saveset, and then restore it on the target system. BACKUP produces output files with a simple fixed length record structure. You should specify an explicit record size on the BACKUP command line such as:

   ```
   BACKUP *.* FILES.BCK/SAVE_SET/RECORD=8192
   ```

   and give the record size to the receiving Kermit with the command SET FILE RECORD-LENGTH before sending your saveset.

1.18.3. Compatibility with Bliss-32 Kermit

Testing has shown that the two Kermits make similar assumptions about file characteristics. A variety of files were uploaded from MS-DOS to VMS with Bliss-32 Kermit, using a variety of transfer modes, and then returned to MS-DOS with C-Kermit and a byte-for-byte comparison was performed. The reverse operation was then tried and tested. No discrepancies were found. Although this does not guarantee 100% compatibility, it is very close. Equivalences:

C-Kermit/VMS	Bliss Kermit-32
SET FILE TYPE TEXT	SET FILE TYPE ASCII
SET FILE TYPE BINARY [FIXED]	SET FILE TYPE BINARY, SET FILE TYPE FIXED
SET FILE TYPE BIN UNDEF	(none)
SET FILE TYPE IMAGE	SET FILE TYPE BLOCK
SET FILE TYPE LABELED	(none)

Note: There is no distinction between SET FILE TYPE BINARY and FIXED in C-Kermit/VMS, and none is needed. C-Kermit determines the file type and length automatically when sending, and when receiving binary files, creates a fixed-format file and marks the true end using the "first free byte" mechanism of RMS. EXCEPT if you have given the command SET FILE TYPE BINARY UNDEFINED, in which case the file is stored with the RMS record format of UNDEFINED rather than fixed (but in the same format as a fixed file).

1.18.4. Date and Time Attributes

The VMS C-Kermit server sends the file's creation date in the attribute packet when a file is sent. When receiving files, if there was valid date/time information in the attribute packet, C-Kermit will set the new file's creation date/time to the value provided. The file will be marked as being revised once (initial creation), on the current date/time. This is necessary to ensure that the file will be detected as "new" by VMS BACKUP. You should also note that if a file with a creation date in the future (as perceived by VMS) is received from a remote Kermit, the system will report "File has creation date in the future" to your system management if they use the ANALYZE/DISK command before the file's creation date has occured.

VMS C-Kermit has the command SET FILE COLLISION UPDATE, which can be used to reject files when there is a newer file than the incoming one with the same name. If you use this option, be aware that due to system clock skew, you may be able to send the file several times in succession before VMS C-kermit rejects it as [now] being older than the one on the VAX.

1.18.5. File Size Attributes

When sending files, VMS C-Kermit fills in the file size information in the attribute packet. Note that the reported sizes may differ from the actual size due to insertion of CR/LF characters on transmission.

For received files, VMS C-Kermit checks checks for sufficient free disk space if the remote Kermit supplies file size information. If there isn't enough space on the disk, VMS C-Kermit will reject the file. Please note that user quotas are not checked. The logic to do so is quite complicated, and would lead to VMS C-Kermit rejecting files that could be received. As an example, consider a user with EXQUOTA privilege. In general, a file size rejection by VMS C-Kermit means the file will not fit; the lack of a rejection does not guarantee that the file will fit. Consider a small file being received onto an RX50 floppy. If debugging is turned on, the debug output will fill the floppy before the file is completely received.

If the device for the received file is not a disk (for example, the system lineprinter) the file is assumed to always fit.

1.18.6. SET FILE TYPE LABELED

Labeled files contain various pieces of information needed to reconstruct the file. At the moment, this is handled by an external utility known as CKVCVT. Some items are always restored and other are not restored unless the user explicitly asks for them. File characteristics are always restored. The file name is restored (de-truncated). However, the complete directory specification is not used to recreate the file unless explicitly requested. This means that converted files will normally be restored to your current directory. Similarly, file ownership is assigned to you unless you specify that the original owner be retained. Note that the two previous options will require privileges. The file backup date is normally cleared so that the file will be backed up. This action may be overridden.

File ACE's are normally not restored. This is an all-or-nothing operation, as there are many types of ACE's under VMS. The most common usage of an ACE is to grant/deny file access to other users. However, various other uses exist. Some examples are: RMS Journaling, RMS Statistics, DDIF, and Pathworks. Note that ACE's are transported in binary mode, which means that protection ACE's may wind up applying to incorrect users if you move files between different VMS systems with ACE's enabled. Also, note that you can create ACE's which you cannot see with DIR/FULL unless you have SECURITY privilege. However, you should be able to view (but not modify) them with EDIT/ACL. This also means that you will not be able to delete these ACE's without deleting the file, which is another reason they are normally not restored.

RMS Journaling files are a special case. Journaling will not be enabled for a received labeled file. A DIR/FULL will show the journaling ACE, which you can use to determine the correct journaling paramters if you wish to journal the new file. Note that if you restore the file to another disk (even if it is on the same system) the journaling information will display incorrectly. This is a limit of the DEC RMS Journaling product.

1.18.7. Reporting Problems

If you have a file that does not transfer correctly, or that exhibits a problem when moved one way with Bliss-32 Kermit and the back with C-Kermit, we would be interested in seeing it (especially if it's small 8-). Please make a BACKUP saveset of it under VMS, and send the saveset along to the BUGS address (use the command BUG in C-Kermit for up-to-date information). Please include any specific information you think would be useful. As a bare minimum, please give the Kermit version number(s), the VMS version, and the compiler version (if you re-compiled Kermit).

1.18.8. Sections that need to be added

Hints for users and system managers on configuring their VMS system and terminal devices for best use with Kermit -- BYTLM quotas, Alt-typahead buffers, etc etc. What are the sysgen options, what SET TERMINAL commands (if any) should be used before running C-Kermit, etc...

1.19. C-Kermit under OS/2

This section describes special considerations for using C-Kermit under OS/2.

C-Kermit 4E(72) was adapted to OS/2 by Chris Adie of Edinburgh University, Scotland, UK. The OS/2-specific code was adapted to version 5A(179) and parts of it were rewritten by Kai Uwe Rommel of the Technical University of Munich, Germany.

C-Kermit is a protected-mode program. It will not run in the DOS compatability environment. This means that it will continue running (eg transferring files) even when it is not the foreground session.

The OS/2 C-Kermit initialization file is called CKERMIT.INI.

All numbers in the C-Kermit documentation are decimal unless noted otherwise.

1.19.1. Requirements

C-Kermit will run on a computer with an 80286 or 80386 processor running OS/2 version 1.0 or higher. It runs in character mode - in other words it is not a Presentation Manager application. However, it will run in a Presentation Manager window as a character application. Normally, though, you would run it from the command processor (CMD.EXE) prompt.

1.19.2. The Serial Port

Naturally, a serial port (COM1 through COM4) is required. The OS/2 serial port device driver must be loaded using a line like one of the following in the CONFIG.SYS file:

```
                      DEVICE=COM01.SYS
   or                 DEVICE=COM02.SYS
   or                 DEVICE=COM.SYS
```

COM01.SYS is used for PC/AT - type machines, while for PS/2s COM02.SYS must be used. C-Kermit *will not work* if this device driver is not loaded. (It provides the Category 1 IOCTLs which are used extensively within the program.) COM01.SYS and COM02.SYS are used for version 1.x of OS/2 only. For the version 2.0 of OS/2 (soon to be released), only one driver COM.SYS exists for both types of machines.

The connecting cable and the modem (or other computer, PAD etc to which your computer is connected) must satisfy the requirements of your computer’s RS232 interface. In particular, the computer will provide two output control signals (RTS and DTR), and may expect to see signals on four input lines (DCD, DSR, CTS, RI). The precise behaviour of these lines is software configurable (for instance by using the OS/2 ’`MODE`’ command), and C-Kermit makes no attempt to impose a particular method of using them.

By default, the DTR and RTS line will both go ON when C-Kermit opens the comms port, and they will go OFF when it is closed.

The default behaviour for the input lines is that DSR and CTS must be ON to enable the port to work. If the modem you are connected to does not provide these signals, you can ’loop back’ the RTS output signal from the computer to DSR and CTS, using a suitably modified cable. An alternative is to use the MODE command to disable the DSR and CTS inputs. To do this, type a command similar to the following at the OS/2 CMD prompt:

```
    MODE COM1:9600,N,8,1,OCTS=OFF,ODSR=OFF,IDSR=OFF
```

You can check the effect using:

```
    MODE COM1
```

which reports the current settings of COM1. Note that on some machines, C-Kermit may appear to work even although DSR and CTS are not connected to anything, nor disabled using ’`MODE`’. This is because unconnected input lines tend to ’float high’. Although this situation may not cause any problems, it is not good practice - you should explicitly disable the inputs as above.

The ’`MODE`’ utility also allows you to change the baud rate, parity, number of data bits and number of stop bits. C-Kermit provides facilities for changing the baud rate and parity too (see later in this manual), but when it starts up, it resets the parity to none and the number of data bits to 8. Any changes to baud rate and parity will remain in effect after C-Kermit terminates.

If you change the parity within C-Kermit, it will ajust the number of data bits to cope. There is no way of changing the number of stop bits within C-Kermit: use ’`MODE`’ to do this.

There is also no way to change the hardware flow control settings from within C-Kermit. There are too many possible settings for the OS/2 serial driver to duplicate all ’`MODE`’ options into C-Kermit. You can use ’`MODE`’ to adjust the hardware flow control and the settings are used by C-Kermit without modification. You can, however, change the software flow control from within C-Kermit.

1.19.3. Emergency Exit

EMERGENCY EXIT: The Control-C and Control-Break keys cannot be used to terminate C-Kermit. To terminate C-Kermit unconditionally, you can select it in the task list and choose ’Terminate’ (OS/2 1.x) or select it in the window list and choose the ’Close’ option from the menu (OS/2 2.0).

C-Kermit returns an exit status of zero, except when a fatal error is encountered, when the exit status is set to one. This can be used in a batch file, to take some action depending on whether the operation was successful. For instance, suppose the file `SEND.CMD` contains the following:

```
echo Sending %1 out port %2
kermit -ql COM%2 -b 9600 -s %1
if ERRORLEVEL 1 goto badend
echo Transferred succcessfully!
goto end
:badend
echo Transfer problems!
:end
```

To send a file `FOO.BAS`, you could type:

```
send foo.bas 2
```

to send it to another computer running Kermit, connected to port COM2. If the transfer completed OK, you would get the message '`Transferred successfully!`'.

1.19.4. The OS/2 File System

The features of the OS/2 file system of greatest interest to Kermit users are the form of the file specifications, and the formats of the files themselves. Note that the following discussion refers to the MS-DOS compatible file system supported by initial versions of OS/2. Installable file systems are not covered here - they are significantly different, and the extent to which C-Kermit will work under such file systems is unknown (because no installable file system has been released at the time of writing).

1.19.4.1. File Specifications

OS/2 file specifications are of the form

```
DEVICE:\PATHNAME\NAME.TYPE
```

where DEVICE stands for a single character identifier (for instance, `A` for the first floppy disk, `C` for the first fixed disk, `D` for a RAM disk emulator) followed by a colon ('`:`'), PATHNAME is up to 63 characters of identifier(s) (up to 8 characters each) surrounded by backslashes ('`\`'), NAME is an identifier of up to 8 characters, and TYPE is an identifier of up to 3 characters in length. Device and pathname may be omitted. The first backslash in the pathname may be omitted if the specified path is relative to the current directory. In the path field, '`.`' means the current directory, '`..`' means the parent directory.

Note that the name, type and path length restrictions apply to the original FAT file system of OS/2. The HPFS file system added in later revisions does not have such hard restrictions. However, C-Kermit will not take much advantage of the long HPFS file names except that it recognizes them when sending/receiving files.

The device and directory specification is normally omitted, but can be specified in all C-Kermit commands. Device and directory pathnames, when omitted, default to either the user's current disk and directory, or to the current directory search path as specified in the PATH environment variable, depending on the context in which the file name appears.

The C-Kermit command line parser treats backslash characters specially and thus requires you either to enter two backslashs when you want to enter one in a file specification or to enter a forward slash instead. The following two commands are equivalent.

```
C-Kermit>send c:\\autoexec.bat
C-Kermit>send c:/autoexec.bat
```

NAME.TYPE is sufficient to specify a file on the current disk and directory, and only this information is sent along by C-Kermit with an outgoing file (by default).

The device, path, name, and type fields may contain uppercase letters, digits, and the special characters '`-`' (dash), '`_`' (underscore), '`$`' (dollar sign), '`&`' (ampersand), '`#`' (number sign), '`@`' (at sign), '`!`' (exclamation mark), '`'`' (single quote), '`()`' (parentheses), '`{}`' (curly braces), '`^`' (caret or circumflex), '`~`' (tilde), and '`` ` ``' (accent grave). Normally, you should confine your filenames to letters and digits for maximum transportability to non-OS/2 systems (by default, C-Kermit will translate filenames being sent by converting non-alphanumeric characters to '`X`'). When you type lowercase letters in filenames, they are converted automatically to uppercase. There are no imbedded or trailing spaces. Other characters may not be included; there is no mechanism for "quoting" otherwise illegal characters in filenames. The fields of the file specification are set off from one another by the punctuation indicated above (ie colon, backslash and dot).

The name field is the primary identifier for the file. The type, also called the extension or suffix, is an indicator which, by convention, tells what kind of file we have. For instance `FOO.BAS` is the source of a BASIC program named FOO; `FOO.OBJ` might be the relocatable object module produced by compiling `FOO.BAS`; `FOO.EXE` could be an executable program produced by loading `FOO.OBJ`, and so forth. `.EXE` is the normal suffix for executable programs.

OS/2 allows a group of files to be specified in a single file specification by including the special "wildcard" characters, '`*`' and '`?`'. A '`*`' matches any string of characters from the current position to the end of the field, including no characters at all; a '`?`' matches any single character. Here are some examples:

`*.BAS`	All files of type BAS (BASIC source files) in the current directory.
`FOO.*`	Files of all types with name FOO.
`F*.*`	All files whose names start with F.
`*.?`	All files with types exactly one character long, or with no type at all.

Wildcard notation is used on many computer systems in similar ways, and it is the mechanism most commonly used to instruct Kermit to send a group of files.

You should bear in mind that other (non-OS/2) systems may use different wildcard characters. For instance VMS and the DEC-20 use '`%`' instead of '`?`' as the single character wildcard; when using C-Kermit to request a wildcard file group from a Kermit-20 server, the OS/2 '`?`' must be replaced by the DEC-20 '`%`'.

1.19.4.2. File Formats

OS/2 systems store files as streams of 8-bit bytes, with no particular distinction among text, program code, and binary files. ASCII text files consist of lines separated by carriage-return-linefeed sequences (CRLFs), and this conforms exactly to the way Kermit represents text files during transmission.

OS/2 (unlike CP/M) knows the exact end of a file because it keeps a byte count in the directory, so one would expect no particular confusion in this regard. However, certain MS-DOS and OS/2 programs continue to use the CP/M convention of terminating a text file with a Control-Z character. This may cause problems when the file is transferred elsewhere, since other systems may object to the Control-Z. By default, therefore, C-Kermit treats the first Control-Z it finds in the file as being equivalent to end-of-file. The Control-Z is not transmitted to the other system. Of course, this leads to problems when transferring non-text files, when we *do* want any Control-Zs in the file to be sent. To achieve this, the C-Kermit '`set file type binary`' command may be used. The opposite, '`set file type text`', is the default.

1.19.5. Terminal Emulation

The CONNECT-mode escape character is Ctrl-] (Ctrl-Rightbracket) by default, but you can change it to any other control character with the SET ESCAPE command.

In connect mode, C-Kermit emulates a DEC VT102 terminal. See the section "Terminal Emulation" for details of how the emulation works.

In CONNECT mode, keyboard input is obtained through the KBD subsystem, and screen output is through the VIO subsystem. It is therefore impossible to redirect terminal I/O.

When you issue a '`connect`' command the first time after starting Kermit, the screen clears and the cursor is positioned at the top left-hand corner. You can log into the remote host computer as normal. In this mode, the PC emulates a DEC VT102 terminal, so any control codes or escape sequences received from the host will be actioned appropriately.

The 25th line on the screen is used as a status line, giving the name of the communications port, the current transmission speed, and how to obtain help.

Some keys on the VT102 keyboard have no direct equivalent on the PC keyboard. The following table shows the mapping which obtains between VT102 keys and PC keys. Note that the `Alt` *n* combinations use the number keys along the top row of the keyboard, not the numeric keypad.

VT102	IBMPC
Delete	Del
PF1	F1
PF2	F2
PF3	F3
PF4	F4
Keypad 0	Alt 0
Keypad 1	Alt 1
Keypad 2	Alt 2
Keypad 3	Alt 3
Keypad 4	Alt 4
Keypad 5	Alt 5
Keypad 6	Alt 6
Keypad 7	Alt 7
Keypad 8	Alt 8
Keypad 9	Alt 9
Keypad minus	F5 or F6
Keypad comma	F7 or F8
Keypad dot	F9
Keypad enter	F10
No Scroll	Scroll-Lock

The PC's '`Scroll-Lock`' key (equivalent to the VT102 '`No Scroll`' key) freezes the data on the screen. It is typically used when listing a long file, to prevent information being scrolled off the top of the screen. Note that the Control-S and Control-Q (Xon/Xoff) keys should not be used for this purpose if '`flow`' is set to '`xon/xoff`', because they interfere with the correct operation of the comms device driver flow control. When the '`Scroll-Lock`' key is pressed, an '`xoff`' will be sent automatically when the device driver's receive buffer fills up, and an '`xon`' will be sent as it empties after the '`Scroll-Lock`' key has been pressed a second time to unfreeze the screen. All other keys are ignored when the screen is frozen. The status line indicates when the emulator is in this state.

Information which scrolls off the top of the screen is not in fact lost, but is stored in an "extended display buffer", which can be examined by pressing the 'PgUp' key. The extended display buffer can contain a number of screenfulls of data, and the 'PgUp' and 'PgDn' keys can be used to range freely through this data. If any other key is pressed while the extended display buffer is visible, the current screen contents are redisplayed and the keystroke is sent to the host. The 'PgUp' and 'PgDn' keys may be used even when the host is still sending data. If Xon/Xoff flow control is in effect, no data will be lost.

The following VT102 features are not implemented:

- Smooth scrolling
- 132-column mode
- Alternate character ROM
- LED lamps

The VT102 keyboard autorepeat mode is always enabled.

When in connect mode, typing the escape character (Control-]) followed by a ? for help will display a "pop-up" help window, indicating the options available. If ^]c is typed to close the connection, the screen is restored to its state when the 'connect' command was issued. A subsequent 'connect' will re-display the VT102 screen.

The control codes and escape sequences recognised by the VT102 emulation are listed below. For full details of the effects of these codes, please consult the VT102 manual.

ENQ	5	Send answerback message "OS/2 Kermit"
BEL	7	Sound beep
BS	8	Cursor left
TAB	9	Cursor to next tab stop
LF	10	Cursor down
VT	11	As LF
FF	12	As LF
CR	13	Cursor to left margin
SO	14	Select G1 character set
SI	15	Select G0 character set
CAN	24	Cancel escape sequence
SUB	26	As CAN
ESC	26	See below
Others		Ignored

ESC 7	Save cursor position
ESC 8	Restore cursor position
ESC D	Index
ESC E	Next line
ESC H	Set tab at current column
ESC M	Reverse index
ESC Z	Identify terminal
ESC c	Reset
ESC =	Enter application keypad mode
ESC >	Exit application keypad mode
ESC # 3	Double height and width emulation, top half line
ESC # 4	Double height and width emulation, bottom half line
ESC # 5	Single height and width
ESC # 6	Single height and double width emulation
ESC # 8	Screen alignment display
ESC (*g*	G0 designator - *g* = A,B or 0 only
ESC) *g*	G1 designator - *g* = A,B or 0 only
ESC [*Pn* A	Cursor up
ESC [*Pn* B	Cursor down
ESC [*Pn* C	Cursor right
ESC [*Pn* D	Cursor left
ESC [*Pl* ;*Pc* H	Direct cursor address
ESC [*Pl* ;*Pc* f	Direct cursor address
ESC [*Pn* c	Identify report - response is ESC [? 6 ; 2 c
ESC [3 g	Clear all tabs
ESC [0 g	Clear tabs at current column
ESC [? *Pn* h	Set DEC private mode, shown below
ESC [? *Pn* l	Reset DEC private mode, shown below

mode no.	mode	set	reset
1	Cursor key	Application	Cursor
2	ANSI/VT52	N/A	VT52
5	Screen	Reverse	Normal
6	Origin	Relative	Absolute
7	Wraparound	On	Off

ESC [*Pn* h	Set mode - modes supported as shown below
ESC [*Pn* l	Reset mode - modes supported as shown below

mode no.	mode	set	reset
2	Keyboard lock	On	Off
4	Insert	Insert	Replace
20	Newline	CR LF	CR

`ESC` *Pn* `i`	`Printer/screen on/off - 4 to 7 supported`
`ESC [ 5 n`	`Status report`
`ESC [ 6 n`	`Cursor position report`
`ESC [` *Pn* `x`	`Request terminal parameter`
`ESC [` *Pn* `;`*Pn* `r`	`Set top and bottom margins`
`ESC [ 0 J`	`Erase to end of screen`
`ESC [ 1 J`	`Erase from beginning of screen`
`ESC [ 2 J`	`Erase all of screen`
`ESC [ 0 K`	`Erase to end of line`
`ESC [ 1 K`	`Erase from beginning of line`
`ESC [ 2 K`	`Erase all of line`
`ESC [` *Pn* `L`	`Insert blank lines`
`ESC [` *Pn* `M`	`Delete lines`
`ESC [` *Pn* `@`	`Insert blank characters`
`ESC [` *Pn* `P`	`Delete characters`
`ESC [` *Ps* `;`*Ps* `; ..;`*Ps* `m`	`Character attributes or`
`ESC [` *Ps* `;`*Ps* `; ..;`*Ps* `}`	`Character attributes, as below:`

`0`	`Default settings`	
`1`	`High intensity`	
`4`	`Underline`	
`5`	`Blink`	
`7`	`Reverse`	
`8`	`Invisible`	
`30-37`	`sets foreground colour to be as shown`	
	`30`	`black`
	`31`	`red`
	`32`	`green`
	`33`	`yellow`
	`34`	`blue`
	`35`	`magenta`
	`36`	`cyan`
	`37`	`white`
`40-47`	`sets background colour to be as shown`	
	`40`	`black`
	`41`	`red`
	`42`	`green`
	`43`	`yellow`
	`44`	`blue`
	`45`	`magenta`
	`46`	`cyan`
	`47`	`white`

Note that the default character set for both G0 and G1 is ’`A`’, ie the UK character set.

The following escape sequences are recognised when the emulator is put into VT52 mode by receiving the sequence `ESC [ ? 2 l`.

ESC A	Cursor up
ESC B	Cursor down
ESC C	Cursor right
ESC D	Cursor leftup
ESC F	Enter graphics mode
ESC G	Exit graphics mode
ESC H	Cursor to home
ESC I	Reverse line feed
ESC J	Erase to end of screen
ESC K	Erase to end of line
ESC Y *l c*	Direct cursor address
ESC Z	Identify
ESC =	Enter application keypad mode
ESC >	Exit application keypad mode
ESC <	Enter ANSI mode

The escape sequences below are accepted but ignored.

ESC O *x*	where *x* is any character
ESC ? *x*	where *x* is any character
ESC [*Pn* q	Load LEDs

1.19.6. Keyboard mapping

The OS/2 version of C-Kermit provides the same keyboard mapping as the UNIX version. A particular key can be mapped to itself (default for all keys), to another key (single key stroke) or to a sequence of key strokes (a macro text). Use the SET KEY command for this purpose.

An extension was made to the standard SET KEY command to allow mapping of the additional PC keyboard keys. Unlike the UNIX version, the OS/2 version knows of 768 rather than only 256 keys. The keys 0..255 are the usual ASCII and extended ASCII (8-bit PC-specific) characters while all extended keys (such as cursor and function keys) in various combinations with SHIFT, CONTROL and ALT are known as 256..767 to C-Kermit. To find out what the number of some key is, enter the SHOW KEY command and press that key. The SHOW KEY command will then report the key number and the current assignment.

Note that a few extended keys have a special meaning to the VT-102 emulator in CONNECT mode. The cursor keys, for example, send the VT-102 cursor key escape sequences while Page-Up and Page-Down are used to control the emulator's scrollback buffer.

That means, the CONNECT mode knows a few key numbers and treats them specially. If you map another key to the code of one of this special known keys, that other key will also function like this special key.

1.19.7. OS/2 C-Kermit Restrictions and Known Bugs

1. Server breakout: There is no way of stopping server operation from the keyboard, short of Control-C.

2. Debugging log: There is very little debugging information logged from the OS/2-specific parts of the program (it was developed using Codeview).

3. Terminal emulation: If the host sends the escape sequence to put the terminal into 132-column mode, and subsequently sends data which would appear in the rightmost 52 columns, this may mess up the existing data on the screen. Really the emulator should ignore any data for these columns.

4. File type: The way Control-Z is handled could be better. We need a '`set ctrlz {on, off}`' like MS-DOS Kermit.

5. A better display of progress of a transfer is needed.

1.19.8. Invoking OS/2 C-Kermit from Another Program

If you are writing a communications program and wish to incorporate the Kermit protocol within it, one way is to use the OS/2 function call `DosExecPgm` to call up C-Kermit. You would supply the instructions for Kermit using command-line options, and Kermit would do the transfer, returning back to your program when it had finished.

The only problem with this scenario is that you might already have opened up the COM port within your program, so that when Kermit tries to do the same it gets an error code back from `DosOpen`. The `-u` command line option gets round this problem. It uses the fact that a child process inherits the open file handles of its parent. `-u` takes one numeric parameter which is the handle of the COM port in question, and it must occur in front of any other command-line parameter which accesses the COM port. The following is a complete C program written using the Microsoft C compiler version 5.1 and the Microsoft OS/2 Software Development Toolkit, which illustrates how to use the `-u` command-line option (NOTE: the UNIX version uses `-l` for this).

```
#define INCL_BASE
#include <os2.h>
/*
 *      Example of how to use the C-Kermit -u option to invoke
 *      Kermit from another program under OS/2.
 */
main(int argc, char *argv[]) {
HFILE   ttyfd;
USHORT  action;
int     err,i;
char    failname[80];
char    args[80];
RESULTCODES     res;
struct dcb {                    /* Device control block */
        USHORT write_timeout;
        USHORT read_timeout;
        BYTE flags1, flags2, flags3;
        BYTE error_replacement;
        BYTE break_replacement;
        BYTE xon_char;
        BYTE xoff_char;
} ttydcb;

        /*** Open a file ***/
        if (err=DosOpen(argv[1],&ttyfd,&action,0L,0,1,0x0012,0L)) {
                printf("Error %d opening %s\n",err,argv[1]);
                exit(1);
        }
        if (err=DosDevIOCtl(&ttydcb,NULL,0x0073,1,ttyfd)) {
                printf("Error %d from IOCTL on %s\n",err,argv[1]);
                exit(1);
        }
        ttydcb.flags3 &= 0xF9;
        ttydcb.flags3 |= 0x04;  /* Read "some" data from line */
        DosDevIOCtl(NULL,&ttydcb,0x0053,1,ttyfd);
        /*** Call kermit ***/
        strcpy(args,"ckoker");
        i = strlen(args);
        args[i++]=0;
        sprintf(&args[i],"-u %d -q -s test.c",ttyfd);
        i += strlen(&args[i]);
        args[i++]=0;
        args[i++]=0;
        if (err=DosExecPgm(failname,80,EXEC_SYNC,args,NULL,&res,
                                                "KERMIT.EXE")) {
                printf("Error %d executing Kermit\n",err);
                exit(1);
        }

        /*** Print out return code ***/
        printf("Termination code %d\n",res.codeTerminate);
        printf("Result code %d\n",res.codeResult);

        /*** Close the file ***/
        if (err=DosClose(ttyfd)) {
                printf("Error %d closing %s\n",err,argv[1]);
        }
}
```

1.20. C-Kermit on the Commodore Amiga

This section written mostly by Jack Rouse and Stephen Walton.

This version of Amiga Kermit is a port of the UNIX C-Kermit which attempts to reproduce as much of the functionality of UNIX version as possible. I had two main goals in porting C-Kermit: I wanted a reliable remote file transfer utility, and I wanted to investigate the use of the AmigaDOS and Exec environments.

Amiga Kermit currently provides a line oriented user interface. The DIAL and SCRIPT commands are as yet unimplemented. Therefore, only Amiga specific features are noted below.

1.20.1. Invoking C-Kermit

Amiga Kermit is usually invoked from a CLI process. However, you can also start it from a Tool type icon, at least for the version of Kermit compiled with Manx Aztec C. Create a Tool icon, called `Kermit.info`, and make sure that it contains a ToolType entry for WINDOW= followed by some window specification. This is actually a small dummy window, as C Kermit will open and use its own window.

From a CLI window, you generally enter:

```
KERMIT
```

or

```
RUN KERMIT
```

to execute Kermit and start up the Kermit command interpreter. Make sure you set your stack to at least 10000 with the AmigaDOS Stack command first. Kermit will create its own window and greet you with:

```
C-Kermit S/W, 5A(180) BETA 8 Feb 92, Commodore Amiga
Type ? or 'help' for help
C-Kermit>
```

The cursor will appear following the C-Kermit> prompt. Typing ? will produce a list of the items that can be entered at any point.

Typing ? at various points during command entry will help you navigate through the command processor. The 'help' command is also quite useful.

The Kermit command processor is normally exited with the 'QUIT' command. During Kermit protocol, you can type CTRL-C or CTRL-D to interrupt and exit Kermit. Depending on the version of the C runtime libraries used to link Kermit, the interrupt may also be active during command input, but it is disabled during connect mode. You will get a requestor when the interrupt is activated to allow you to choose to continue Kermit. However, any serial read or write that was interrupted will still be aborted.

Kermit can also be used without the command processor by specifying an action on the command line. You can enter 'kermit -h' at the CLI prompt to get a list of command line options. However, unless input is redirected, or the -q (quiet) option is specified, Kermit will still create a window for protocol monitoring and interruption. 'KERMIT <*' can be used to run Kermit completely within the CLI window; however, this does not allow you to enter control characters, and no console input is seen until you enter return. Input and output can be redirected to files to take advantage of C-Kermit command line file transfer options. Unfortunately, AmigaDOS does not implement pipes (yet).

1.20.2. Kermit Serial Initialization and Settings

Amiga Kermit uses the serial device or a reasonable emulation thereof. Three sources are used to initialize the serial parameters. First, the default serial configuration, as set by Preferences, is copied. This includes baud rate, and under version 1.2 of the Workbench, parity, modem control (7-wire vs. 3-wire), and flow control. Second, command line parameters can be used to override these settings. For example:

```
kermit -b 1200 -p e
```

can be used to select 1200 baud and even parity independently of the Preferences settings. Finally, if the command processor is used, Kermit looks for a `.kermrc` initialization file, first in the `s:` directory, then in the current directory, providing that the disk containing each directory is present in the Amiga. The `.kermrc` file contains C-Kermit commands which can be used to initialize the C-Kermit environment as desired.

The Kermit SET LINE command has a special form in Amiga Kermit, namely:

```
SET LINE device/unit
```

where *device* is the name of a device (including the `.device` extension) and *unit* is the unit number. The default setting is equivalent to:

```
SET LINE serial.device/0
```

The modem control mode is currently selected by 'SET MODEM type', which has two choices: 'DIRECT', for 3-wire control, and 'GENERIC', for 7-wire control. Because of the way the serial device operates, this setting only has an effect when the serial device is opened after previously being closed, which occurs only when the serial line is used after Kermit starts or after '<escape>H' is used to hang up and exit connect mode.

Kermit allows you to set any baud rate between 110 and 292000 baud; however, it will complain if the baud rate is nonstandard. Rates of 110 and 111 baud are implemented as 112 baud. Rates above 38400 baud can be used for connect mode, but they are not very useful for file transfer. The file transfer rate is limited by packet retries due to transfer errors, and the overhead time spent constructing packets.

Amiga Kermit uses the serial device in shared mode. This allows other programs, like dialers, to use the serial line at the same time, without exiting Kermit. This could also allow in theory a terminal emulator to be used simultaneously with Kermit. However, if two programs are reading from the serial line at the same time, the results are unpredictable. Any such program, therefore, would have to be disabled from reading while Kermit is performing file transfer or is in connect mode. Note that Kermit since does its own parity generation and stripping, so it always sets the serial device to use eight bit characters with no parity.

1.20.3. Amiga Wildcards

Both the SEND command invoked from the Amiga and a GET sent to an Amiga in server mode use allow wildcarding, in the UNIX style. Thus, ’*’ wildcard matches an arbitrary string while ’?’ matches an arbitrary character. Therefore, to get all the C source files which begin with ’cki’ from the Amiga server, you could use the command:

```
        C-Kermit>get cki*.c
```

Multiple ’*’ wildcards can be used in a pattern. Remember, the wildcarding that is used in local and remote server commands that invoke AmigaDOS commands is the AmigaDOS form.

1.20.4. Local and Remote Commands

Amiga Kermit provides several ways to invoke AmigaDOS commands from within Kermit. Entering ’!’ at the Kermit prompt will create a CLI process running in its own window, and wait for it to terminate. The form ’! command’ will invoke the given command command with its output going to Kermit’s window. There is currently no way to pause the output of commands invoked his way, other than the stopgap use of the right mouse button. The form ’REMOTE HOST command’ can be sent to the Amiga server to execute the given command remotely on the Amiga. Because of the way AmigaDOS Execute() function works, commands invoked in either the ’! command’ or ’REMOTE HOST command’ forms have NIL: as their standard input. Some AmigaDOS commands that require input, such as DiskCopy and Format, do not recognize the immediate end of file that they receive under this condition, causing them to hang.

In addition to the methods given above, various AmigaDOS commands are invoked by local Kermit commands, and generic remote commands. These are listed below:

AmigaDOS command	Local command	Remote generic command
DELETE files	--none--	REMOTE DELETE files
TYPE files	--none--	REMOTE TYPE files
INFO	SPACE	REMOTE SPACE
LIST obj	DIRECTORY obj	REMOTE DIRECTORY
STATUS	--none--	REMOTE WHO

Any parameters to these commands are expected to use AmigaDOS conventions, including AmigaDOS wildcarding. Note that in order to pass a ’?’ through the C-Kermit command processor, it must be prefixed with a ’\’.

You can change the current directory of the Kermit process locally with the CWD command and remotely with REMOTE CWD. The local CWD command prints out the name of the current directory afterwards. If no new directory is given, the current directory is not changed, so CWD alone can be used to determine where the current directory is.

1.20.5. Server Mode

Amiga Kermit completely implements server mode, including the ability to execute CLI commands remotely. Currently CLI commands are executed with their standard output directed to RAM:PIPE-HOLDER, which is then written back to the commanding Kermit after the command completes.

There are a few limitations on the commands that can be executed remotely. First of all, if they produce voluminous output, the output should be redirected (redirection is supported on the REMOTE HOST command line) to avoid using all free memory for the output file. However, the commanding Kermit will probably timeout in the middle of the execution of any such command. The best way to use these commands is to

```
        REMOTE HOST RUN command >outfile parameters
```

then use REMOTE WHO (which invokes STATUS) to monitor the command for completion.

The input stream for remote commands is NIL:, which is not handled intelligently by all Amiga commands. For example, ’REMOTE HOST diskcopy df0: to df1:’ hangs indefinitely while waiting for NIL: to press return. Finally, since each command is executed in a separate CLI, commands that set unshared process parameters, like ’cd’, will have null effect (but ’REMOTE CWD dir’ can be used instead).

While server mode is active, AmigaDOS requestors are disabled. This avoids requiring operator intervention to reset a requestore when the Amiga server is told to use a file on a disk that does not exist or is write protected. However, disabled requestors are currently not inherited by the CLI processes that the server creates to execute remote commands. Therefore, a remote AmigaDOS command can still cause the server to become hung.

To shut down the Amiga server, enter BYE or FINISH at the commanding Kermit. FINISH exits to whatever level the server was invoked from, while BYE exits Amiga Kermit altogether.

1.20.6. Connect Mode

Connect mode on Amiga Kermit currently provides you with a standard AmigaDOS console device window. Using the default Preferences setting, this gives a 23 row by 77 column screen. However, the MoreRows program allows you to increase the size of the Workbench window beyond the 640 by 200 default size; increasing the number of rows by 8 and the number of columns by 16 will allow a 24 row by 80 column Kermit window. The Amiga console device is used to provide ANSI terminal emulation. While you are in connect mode, you can give single character commands which are prefixed by an escape character which can be set from within C-Kermit. By default, the escape character is CTRL-\. You can use ’<escape>H’ to close the serial device and exit connect mode, which makes the DTR line drop causing most modems to hang up the phone line.

You can currently get a 25 by 80 screen in Kermit by means of a kludge. Entering the Kermit command line (backslashes will be echoed only once):

```
     ECHO \\033[25t\\033[80u\\033[0x\\033[0y\\014
```

activates console device private escape sequences that cause the console to use a 25 by 80 region, overwriting the borders of the Kermit window. Using window gadgets will cause the borders to be redisplayed, but the display can be cleaned up by typing ctrl-L in command mode. To reset the window to its normal condition, allowing resizing, use:

```
     ECHO \\033[t\\033[u\\033[x\\033[y\\014
```

and then activate a window gadget to refresh the borders. These commands can be placed into Kermit

TAKE files.

In addition to the standard connect mode commands, extra logging control has been added. If a session log file is open, the ’<escape>Q’ sequence allows you to temporarily suspend logging. The ’<escape>R’ sequence resumes logging if it has been suspended.

Features have also been added to prevent deadlocks while in connect mode due to spurious XOFF’s or bad modem control line states. When connect mode is unable to send serial output, keyboard characters are queued until they can be transmitted. Queuing continues as long as space is available in the output buffer. If the buffer, which is 64 characters long, fills up, the next keyboard input is discarded and the display ’beeps’. To get out of a deadlock situation, you can either exit connect mode, or send a break. In either case, the output queue is flushed, and current serial output character is given one second to finish transmitting. If it does not complete, the output is aborted, and XOFF mode reset as appropriate. Then connect mode is exited or a break is sent, as specified. When output characters are queued, connect status (accessed by ’<escape>S’) will indicate the number of queued output characters.

1.21. C-Kermit under OS-9

C-Kermit was adapted to OS-9 by Christian Hemsing, RWTH, Aachen, Germany, and Bob Larson of the University of Southern California. This section of the manual was written by Christian Hemsing.

The commands and operation of OS-9 C-Kermit should be identical to those of UNIX C-Kermit, with the exceptions noted here and in the "beware file".

The initialization file is `.kermrc`.

1.21.1. OS-9/68K Background

OS-9/68k is a multiuser, multitasking operating system designed to run on all Motorola 680x0 family processors from Microware Systems Corporation, 1900 N.W. 114th Street, Des Moines, Iowa 50332 (Trademarks: Microware, OS-9, OS-9/68000, OS-9000).

Due to its modular design, most of the code is completely hardware independent, so it can be easily be ported to a different hardware by writing new device drivers.

The original (1980) OS-9/6809 was designed for the Motorola 6809 processor. Later (1983) they switched to the 680x0 family and released OS-9/68000. For speed and compactness reasons most of the OS-9/68000 kernel is written in 680x0 assembler language. Now there is a so-called OS-9000 by Microware. Its kernel is written in C and thus it is portable. It is presently available for 680x0 and Intel’s 80386/486, and Microware plans to add further support for RISC and CISC processors. (C-Kermit has not yet been tested under OS-9000.)

The 100% ROM-able, fast, compact code in conjunction with real-time capabilities make OS-9/68k ideal for ROM-based systems used in measuring, controlling, etc. It has found a wide acceptance within the scientific and industrial world.

Yet, a full disk based OS-9/68k offers a program development environment similar to UNIX. This includes (of course, limited) UNIX sofware compatibility at C source code level, source code level debugging, UNIX I/O model, UNIX task model, UNIX-like shell and networking.

A number of UNIX utilities like lex, yacc, lint, etc, have been ported to OS-9/68k.

The basic commands of OS-9 are:

DEL	delete a file
DELDIR	delete a directory
MAKDIR	create a directory
DIR	directory listing
PROCS	show currently running processes
LIST	type contents of a text file
CHD	change working directory
PD	print working directory

All commands can be given a `"-?"` as a switch, which will display a brief (usually sufficient) help message.

All command references (like all references to names on OS-9/68k) are NOT case sensitive (switches, though, may be case sensitive since they are interpreted by the running program).

1.21.2. OS-9/68K Devices

All devices (terminal lines, networks, disks) can have arbitrary names but the usual convention is:

Terminal lines:

`term`	the console terminal
`t1`	terminal line #1
:	
`t`*n*	terminal line #*n*

Hard disks:

`h0`	hard disk #0
`h1`	hard disk #1
:	
`h`*n*	harddisk #*n*

Floppy disks (diskettes):

`d0`	floppy disk drive #0
`d1`	floppy disk drive #1
:	
`d`*n*	floppy disk drive #*n*

A path name starting with a slash ("/") must always include a device name as the first field. For example, the C-Kermit command SET LINE /T3 would select the terminal line `/t3`.

The console terminal is either a real terminal, or the screen and keyboard of a workstation such as a Macintosh, Amiga, or Atari ST that is running OS-9. Terminal emulation is not done by OS-9 C-Kermit, but rather by the real terminal or the workstation console driver. This includes the capability to display national and international characters.

1.21.3. The OS-9/68K File System

The file system is tree-structured just like the UNIX file system.

```
/h0/chris
```

means the directory or file "chris" on hard disk #0.

```
chris/rubbish
```

means the subdirectory or file "rubbish" in the subdirectory "chris" of the current directory.

The command "chd" without any parameters will always take you to your home directory.

Names of files, directories, devices, and commands are case-independent. Filenames may contain letters, digits, period, underline, and dollar sign. They are stored with upper and lower case preserved, but case is not significant when referring to them.

Wildcard expansion is performed by the shell with two metacharacters:

`*` stands for an arbitrary string of arbitrary length `?`
denotes a single character

C-Kermit/OS-9 also expands wildcards itself, using the same notation, for example:

```
C-Kermit>send ck*.\?
```

(Note: the question mark must be prefixed by "\" to override its normal function of giving a help message.)

OS-9/68k files are sequential streams of 8-bit bytes, just like in UNIX, except that carriage return (CR, ASCII 13) is the line terminator, rather than linefeed (LF, ASCII 10). Binary files are simply streams of arbitrary 8-bit bytes. The OS-9 operating system and utilities are "8-bit clean", so text files can use any ASCII-based character set that is compatible with your display and data entry devices, for example ISO 8859-1 Latin Alphabet 1.

Unlike UNIX, OS-9/68k has a built-in method to gain exclusive access to devices, so no lock files are needed. The user will be told if the device is already in use.

1.21.4. To Build C-Kermit for OS-9

Collect all the C-Kermit source files into a directory:

```
ckc*.c, ckc*.h, cku*.c, cku*.h, ckwart.c, ckcpro.w, ck9*.*
```

There are two makefiles: `ck9ker.mak` and `ck9ker.gcc`. If you have a running version of the GNU C compiler, use `ck9ker.gcc` (it produces smaller, more efficient code); otherwise use `ck9ker.mak` which uses the standard OS-9/68k C compiler. Read the approppropriate makefile, edit the necessary changes mentioned there, create the subdirectories, and make the new Kermit by typing:

```
make -f=ck9ker.mak
```

or:

```
make -f=ck9ker.gcc
```

Read the "beware file" `ck9ker.bwr` for hints relating to the OS-9 terminal driver.

Index

Table of Contents

List of Tables