C-Kermit for OS/2

Version 5A(179

)

by Chris Adie, Kai Uwe Rommel, et al.

1. Introduction

1.1. Acknowledgements

C-Kermit was written by Frank da Cruz, Bill Catchings, Jeff Damens and Herm Fischer, with contributions by many others. It was originally written for computers running the Unix operating system, but it has been ported to the VAX running VMS, the Apple Macintosh and the Commodore Amiga, among others.

The program was ported at version 4E(79) to OS/2 by Chris Adie (Edinburgh University Computing Service). The port was adapted to version 5A(179) and parts of it were rewritten by Kai Uwe Rommel.

This documentation is based closely on the C-Kermit manual by Frank da Cruz and Herm Fischer. The section on the OS/2 file system is based on the MS-DOS Kermit manual, by Christine Gianone, Frank da Cruz, and Joe Doupnik. It should be read in conjunction with a more general description of Kermit such as "The Kermit File Transfer Protocol" by Frank da Cruz (Digital Press, ISBN 0-932376-88-6).

1.2. OS/2 Kermit Capabilities

C-Kermit provides traditional (Unix-style) command line operation as well as interactive command prompting and execution. The command line options provide access to a basic subset of C-Kermit's capabilities; the interactive command set is far richer.

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.

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

Summary of capabilities:

Local operation:	Yes
Remote operation:	No
Login scripts:	Yes (UUCP style)
Transfer text files:	Yes
Transfer binary files:	Yes
Wildcard send:	Yes

File transfer interruption:	Yes
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	Yes
Alternate block checks:	Yes
Terminal emulation:	Yes
Communication settings:	Yes
Transmit BREAK:	Yes
Support for dialout modems:	Yes
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
Local file management:	Yes
Command/Init files:	Yes
Long packets:	Yes
Sliding Windows:	Yes
File attributes packets:	Yes
Command macros:	Yes
Raw file transmit:	Yes

1.3. 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.

1.4. 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.

1.5. 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).

2. 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).

2.1. 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.

Pathname 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.

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 '%'.

2.2. 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.

Non-OS/2 systems may be confused by nonstandard ASCII files sent by C-Kermit:

- Files containing any of the 8-bit "extended ASCII" characters may need conversion (or translation) to 7-bit ASCII.
- Files produced by word processing programs like Word Perfect or Wordstar may contain special binary formatting codes, and could need conversion to conventional 7-bit ASCII format prior to transmission, using commonly available "exporter" programs.
- Spreadsheet or database files usually need special formatting to be meaningful to non-OS/2 recipients (though they can be transmitted between OS/2 and MS-DOS systems with Kermit).
- BASIC programs are normally saved in a binary "tokenized" form. Use BASIC's "`,a`" option to save them as regular ASCII text, as in

```
save"foofa",a
```

In general, when attempting to transfer non-text files between OS/2 and a different kind of system, consult the Kermit manual for that system.

3. File Transfer

When C-Kermit is transferring a file, the screen (stdout) 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 Initiate
S Send Initiate
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 certain "interrupt" 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.

CAUTION: If F or X is used to cancel an incoming file, and a file of the same name previously existed, *and* the "file warning" feature is not enabled, then the previous copy of the file will disappear.

The Emergency Exit key (Control-C) can be used to terminate a file transfer. This will not terminate the transfer gracefully (it is a ’brute force’ method), so the remote Kermit system may be left in an undefined state.

4. Command Line Operation

The C-Kermit command line syntax 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 implications of this for specifying command-line options are:

- An option name is a single character.
- Options are delimited by '`-`'.
- Options with no arguments may be grouped (bundled) behind one delimiter.
- Options which take an argument must not have the argument omitted.
- Arguments immediately follow options, separated by whitespace.
- The order of options does not matter.
- A '`-`' 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 An OS/2 file specification, possibly containing the "wildcard" characters '`*`' or '`?`'.

fn1 An OS/2 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.

4.1. Command Line Options

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

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

or a parallel process:

```
dir *.txt | kermit -s -
```

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

```
kermit -s .\-
```

The argument *fn* may contain wildcards. For instance,

```
kermit -s ck*.h
```

will send all the files matching the specification '`ck*.h`'.

`-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.

`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.txt
```

`-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.

Before proceeding, a few words about remote and local operation are necessary. Kermit (in general, not just 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. Kermit is remote if it is running on a multiuser system and transferring files over its own controlling terminal's communication line, connected to your PC or workstation.

C-Kermit running under OS/2 is always used in local mode, with the serial port at the back of the machine designated for file transfer and terminal connection. Which serial port to use is determined by the `-l` command-line option:

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

```
kermit -l com2
```

The default line is COM1.

There are a number of other options associated with the use of the serial port:

`-b` *n* Baud -- Specify the baud rate for the line given in the `-l` option, as in

```
kermit -l com2 -b 9600
```

It is good practice to include this with the `-l` option, since the current speed of the serial port might not be what you expect.

`-p` *x* Parity -- *x* may be one of `e`,`o`,`m`,`s`,`n` (even, odd, mark, space, or none). If parity is other than none, then the 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.

The following command line options access a remote Kermit server:

`-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.

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

The following command line options are to do with connecting to the remote system as a terminal:

`-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-`]`) 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.

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, and is equivalent to the '`set file type binary`' interactive command.

`-w` Write-Protect -- Avoid filename collisions for incoming files.

`-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, 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.

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

`-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. (debug log disabled)

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

`-u` *d* File handle -- The argument *d* is the numerical value of an open file handle which Kermit will use for its communication line. This option is for use only by other programs which interface with Kermit, not for typing in the command line (so it's not described in the help information from `-h`). Further details are given in the Appendix.

The command line may contain no more than one protocol action option (ie only one of: `-s`, `-a`, `-r`, `-k`, `-x`, `-g`, `-f`).

See the Unix C-Kermit manual for a full description of all available options. Run C-Kermit for OS/2 with option `-h` to get a short description of available options online.

Files are sent with their own names, except that lowercase letters are raised to upper, drive specifiers and pathnames are stripped off, and non-alphanumeric characters (excepting the dot) are changed to '`X`'. Incoming files are stored under their own names. If `-w` was specified, a "generation number" is appended to the name if it has the same name as an existing file which would otherwise be overwritten. If the `-a` option is included, then the same rules apply to its argument. The file transfer display shows any transformations performed upon filenames.

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.
- 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).

4.2. Command Line Examples

```
kermit -l com1 -b 1200 -cn -r
```

This command connects you to the system on the other end of COM1 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 com2 -b 9600 -cntp m -r -a foo.bar
```

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.bar`.

```
kermit -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). This example assumes that the remote server is connected to COM1 (Kermit's default comms port), and that the current baud rate of the port is acceptable to the remote server.

```
kermit -wx
```

This command starts up C-Kermit as a server. Incoming files that have the same names as existing files are given new, unique names.

```
kermit -l com2 -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 (eg in this case by '`set line`' or '`set speed`' commands).

```
kermit
```

This command starts up Kermit interactively with all default settings. The serial line used will be COM1, and the speed used will be the current speed of COM1.

A final example shows how an OS/2 compress utility might be used to speed up Kermit file transfers:

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

4.3. Exit Status Codes

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!`'.

5. Interactive Operation

C-Kermit's interactive command prompt is "`C-Kermit>`". In response to this prompt, you may 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, so 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`').

Certain characters have special functions during typein of 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.

`ESC` (The Escape 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.

`BS` (Backspace) -- Delete the previous character from the command.

`^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.

`SP` (Space) -- Delimits fields (keywords, filenames, numbers) within a command. The tab key may also be used for this purpose.

`CR` (Carriage Return or Enter) -- Enters the command for execution. LF (Linefeed or Control-J) or FF (formfeed or Control-L) may also be used for this purpose.

`^` (Circumflex) -- Enter any of the above characters into the command, literally. To enter a `^`, type two of them in a row (`^^`). A circumflex 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, especially in the 'script' command.

You may type the editing characters (`BS`, `^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.

A command line beginning with a percent sign '`%`' is ignored. Such lines may be used to include illustrative commentary in Kermit command dialogs.

Interactive C-Kermit accepts commands from files as well as from the keyboard. When you start C-Kermit, the program looks for a file `CKERMIT.INI` in the current PATH, and executes any commands it finds there. The commands in `CKERMIT.INI` must be in interactive format, not in the command-line format. A '`take`' command is also provided for use at any time during an interactive session, to allow interactive-format commands to be executed from a file; command files may be nested to any reasonable depth.

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

`%`	Comment
`!`	Execute an OS/2 command, or start up CMD.EXE.
`bye`	Terminate and log out a remote Kermit server.
`close`	Close a log file.
`connect`	Establish a terminal connection to a remote system.
`cwd`	Change Working Directory.
`dial`	Dial a telephone number.
`directory`	Display a directory listing.
`echo`	Display arguments literally.
`exit`	Exit from the program, closing any open files.
`finish`	Tell remote Kermit server to exit, but not log out.
`get`	Get files from a remote Kermit server.
`help`	Display a help message for a given command.
`log`	Open log file: debugging, packet, session, transaction.
`quit`	Same as 'exit'.
`receive`	Passively wait for files to arrive.
`remote`	Do some file management on a remote Kermit server.
`script`	Execute a login script with a remote system.
`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.
`take`	Execute commands from a file.

The 'set' parameters are:

`block-check`	Level of packet error detection.
`delay`	How long to wait before sending first packet.
`duplex`	Specify which side echoes during '`connect`'.
`escape-character`	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.
`incomplete`	Disposition for incompletely received files.
`line`	Communication line device name.
`modem-dialer`	Type of modem-dialer on communication line.
`parity`	Communication line character parity.
`prompt`	The C-Kermit program's interactive command prompt.
`receive`	Parameters for inbound packets.
`retry`	Packet retransmission limit.
`send`	Parameters for outbound packets.
`speed`	Communication line speed.
`terminal`	Terminal parameters.

The 'remote' commands are:

`cwd`	Change remote working directory.
`delete`	Delete remote files.
`directory`	Display a listing of remote file names.
`help`	Request help from a remote server.
`host`	A command to the remote host in its command language.
`space`	Display current disk space usage on remote system.

`type` Display a remote file on your screen.
`who` Show who's logged in, or get information about a user.

Most of these commands are described adequately in the Kermit User Guide or the Kermit book. Special aspects of certain C-Kermit commands are described below. There are more (special) commands available. See the Unix C-Kermit manual for a full description.

5.1. The 'send' command

Syntax: `send` *fn*
or `send` *fn1 rfn1*

Send the file or files denoted by *fn* to the other Kermit, which should be running as a server, or which should have been given the '`receive`' 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 OS/2 file, *rfn1* may be specified as a name to send it under. The '`send`' command may be abbreviated to '`s`', even though '`s`' is not a unique abbreviation for a top-level C-Kermit command.

The wildcard characters '`*`' and '`?`' are accepted in *fn*. If '`?`' is to be included, it must be prefixed by '`^`' to override its normal function of providing help. '`*`' matches any string, '`?`' matches any single character. When *fn* contains '`*`' or '`?`' characters, there is a limit to the number of files that can be matched, which varies depending on the length of the file names involved. If you get the message '`Too many files match`' then you'll have to make a more judicious selection.

5.2. The 'receive' command

Syntax: `receive`
or `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`'. If the second form of the command is given, and *fn1* contains a path specification, then that path must exist -- C-Kermit will not create a directory for the file.

5.3. The 'get' command

Syntax: `get` *rfn*
or `get`
rfn
fn1

Request a remote Kermit server to send the named file or files. 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.ibm
```

As with '`receive`', if more than one file arrives as a result of the '`get`' command, only the first will be stored under the alternate name given by *fn1*; the remaining files will be stored under their own names if possible. 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>
```

5.4. The 'server' command

The 'server' command places C-Kermit in "server mode" on the currently selected communication line. All further commands must arrive as valid Kermit packets from the Kermit on the other end of the line. The OS/2 C-Kermit server can respond to the following commands:

Command	Server Response
get	Sends files
send	Receives files
bye	Attempts to log itself out
finish	Exits to level from which it was invoked
remote directory	Sends directory lising
remote delete	Removes files
remote cwd	Changes working directory
remote type	Sends files to your screen
remote space	Reports about its disk usage
remote help	Lists these capabilities
remote host	Execute an OS/2 command

Note that the 'remote host' command should be used with great care. It should only be used to invoke OS/2 commands which produce their output through stdio, and which require no keyboard interaction. Commands such as 'copy' and 'rename' are OK (although they may sometimes produce output on stderr, which will appear on the screen of the OS/2 system). It is not possible to use the 'remote host' command to run a word proccessor, for instance.

5.5. The 'remote', 'bye', and 'finish' commands

C-Kermit may itself request services from a remote Kermit server. In addition to 'send' and 'get', the following commands may also be sent from C-Kermit to a Kermit server:

remote cwd [*directory*]
: If the optional remote directory specification is included, you will be prompted on a separate line for a password, which will not echo as you type it. If the remote system does not require a password for this operation, just type a carriage return.

remote delete *rfn*
: delete remote file or files.

remote directory [*rfn*]
: directory listing of remote files.

remote host *command*
: issue a command in the remote host's own command language.

remote space
: disk usage report from the remote host.

remote type *rfn*

display remote file or files on the screen.

`remote who [`*user*` ]`
display information about who's logged in.

`remote help`
display remote server's capabilities.

`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.

5.6. The 'log' and 'close' commands

Syntax: `log {debugging, packets, session, transactions} [`*fn1*` ]`
`close {debugging, packets, session, transactions}`

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.

`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 dramatically slows down the Kermit protocol. Default name: `debug.log`. (debug log disabled)

`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 explicitly closes a log, e.g. '`close debug`'.

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, and the commands relating to them will not be present.

5.7. Local File Management Commands

OS/2 Kermit allows some degree of local file management from interactive command level:

`directory` [*fn*]
: Displays a listing of the names, sizes, and dates of files matching *fn* (which defaults to '`*.*`'). Equivalent to '`dir`'.

`cwd directory-name`
: Changes Kermit's working directory to the one given. The directory specification may be preceeded by a drive specifier, in which case that becomes the current drive. This command affects only the Kermit process and any processes it may subsequently create.

`space`
: Display information about disk space and/or quota in the current directory and device. Equivalent to '`chkdsk`'.

`! [`*command* `]`
: The command is executed by the OS/2 command interpreter CMD.EXE. If no command is specified, then CMD.EXE itself is started; terminating it by typing 'exit' 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. The '`!`' command has certain peculiarities:

- At least one space must separate the '`!`' from the *command*.
- A '`cd`' or '`chdir`' (change directory) command executed in this manner will have no effect on returning to Kermit -- use the C-Kermit '`cwd`' command instead.

5.8. The 'set' Command

Syntax: `set` *parameter [parameter] value*

Since Kermit is designed to allow diverse systems to communicate, it is often necessary to issue special instructions to allow the program to adapt to peculiarities of the another system or the communication path. These instructions are accomplished by the '`set`' command. The '`show`' command may be used to display current settings. Here is a brief synopsis of settings available in the current release of C-Kermit:

`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 would be used to advantage when transferring 8-bit binary files over noisy lines.

`delay` *n*
: How many seconds to wait before sending the first packet after a '`send`' command, in remote mode only. It is irrelevant for OS/2 Kermit, since it is always in local mode.

`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 typein itself.

`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 OS/2 command level. The normal escape character is Control-`]` (29).

`file {display, names, type, warning}`

Establish various file-related parameters:

`display {on, off}`

Normally '`on`'; when in local mode, display progress of file transfers on the screen (stdout), and listen to the keyboard for interruptions. If '`off`' (equivalent to '`-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 console terminal.

`names {converted, literal}`

Normally '`converted`, which means that outbound filenames have path specifications stripped and non-alphanumeric characters changed to `X`'s (except for the dot). '`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.

`type {binary, text} [{7, 8}]`

The file type is normally text, which means that any control-Z in a file being transmitted is treated as an end-of-file mark. Binary means transmit file contents without conversion. Binary ('`-i`' in command line notation) is necessary for binary files.

The optional trailing parameter tells the bytesize for file transfer. It is 8 by default. If you specify 7, the high order bit will be stripped from each byte of sent and received files. This is useful for transferring text files that may have extraneous high order bits set in their disk representation (e.g. Wordstar or similar word processor files).

`warning {on, off}`

Normally '`off`', which means that incoming files will silently overwrite existing files of the same name. When '`on`' ('`-w`' on command line) Kermit will check if an arriving file would overwrite an existing file; if so, it will construct a new name for the arriving file, 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.

> *CAUTION:* If F or X is used to cancel an incoming file, and a file of the same name previously existed, *and* the "file warning" feature is not enabled, then the previous copy of the file will disappear.

`flow-control {none, 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.

`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 such file fragments will be kept.

`handshake {xon, xoff, cr, lf, bell, esc, none}`

Normally '`none`'. Otherwise, half-duplex communication line turnaround handshaking is done, which means 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'.

`line [`*dev* `]`

The device name for the communication line to be used for file transfer and terminal connection, e.g. COM2. If you omit the device name, Kermit will revert to its default device, COM1.

`modem-dialer {direct, hayes, racalvadic, ventel, ...}`

The type of modem dialer on the communication line. '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' (or the '-l' argument) will prepare for a subsequent 'dial' command for the given dialer. 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. See the description of the 'dial' command.

`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 binary data, which can be used successfully only if the other Kermit agrees; if not, 8-bit binary data cannot be successfully transferred.

`prompt [`*string* `]`

The given string will be substituted for 'C-Kermit>' as this program's prompt. If the string is omitted, the prompt will revert to 'C-Kermit>'. If the string is enclosed in double quotes, the quotes will be stripped and any leading and trailing blanks will be retained.

`send` *parameter*

Establish parameters to use when sending packets. These will be in effect only for the initial packet sent, since the other Kermit may override these parameters during the protocol parameter exchange (unless noted below).

`end-of-packet` *cc*

Specifies the control character needed by the other Kermit to recognize the end of a packet. C-Kermit sends this character at the end of each packet. Normally 13 (carriage return), which most Kermit implementations require. Other Kermits require no terminator at all, still others may require a different terminator, like linefeed (10).

`packet-length` *n*

Specify the maximum packet length to send. 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. This command overrides the value requested by the other Kermit during protocol initiation unless the other Kermit requests a shorter length.

`pad-character` *cc*

Designate 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.

`padding` *n*

How many pad characters to send, normally 0.

`start-of-packet` *cc*

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.

`timeout` *n*

Specifies the number of seconds you want the other Kermit to wait for a packet before timing it out and requesting retransmission. Defaults to 10 seconds.

`receive` *parameter*

Establish parameters to request the other Kermit to use when sending packets.

`end-of-packet` *cc*

Requests the other Kermit to terminate its packets with the specified character.

`packet-length` *n*

Specify the maximum packet length to that you want the other Kermit to send, normally 90. If you specify a length of 95 or greater, then it will be used if the other Kermit supports, and agrees to use, the Kermit protocol extension for long packets. In this case, the maximum length depends upon the systems involved, but there would normally be no reason for packets to be more than about 1000 characters in length. The '`show parameters`' command displays C-Kermit's current and maximum packet lengths.

`pad-character` *cc*

C-Kermit normally does not need to have incoming packets preceded with pad characters. This command allows C-Kermit to request the other Kermit to use *cc* as a pad character. Default *cc* is NUL, ASCII 0.

`padding` *n*

How many pad characters to ask for, normally 0.

`start-of-packet` *cc*

Change the prefix C-Kermit looks for on inbound packets to correspond with what the other Kermit is sending.

`timeout` *n*

Normally, each Kermit partner sets its packet timeout interval based on what the opposite Kermit requests. This command allows you to override the normal procedure and specify a timeout interval for OS/2 Kermit to use when waiting for packets from the other Kermit. If you specify 0, then no timeouts will occur, and OS/2 Kermit will wait forever for expected packets to arrive.

`speed {110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200, 38400, 57600}`

The baud rate for the external communication line. '`set baud`' is a synomym for '`set speed`'. 19200 baud and higher rates may not be available, depending on your communications hardware.

`terminal` *parameter value*

Used for specifying terminal parameters. Currently, '`bytesize`' is the only parameter provided, and it can be set to 7 or 8. This controls the width of the data path between the console and the remote system when C-Kermit is in connect (ie terminal emulation) mode. It's 7 by default.

5.9. The 'show' Command

Syntax: `show {parameters, versions}`

The '`show`' command with the default argument of '`parameters`' displays the values of all the '`set`' parameters described above. If you type '`show versions`', then C-Kermit will display the version numbers and dates of all its internal modules. You should use the '`show versions`' command to ascertain the vintage of your Kermit program before reporting problems to Kermit maintainers.

5.10. The 'statistics' Command

The statistics command displays information about the most recent Kermit protocol transaction, including file and communication line i/o, timing and efficiency, as well as what encoding options were in effect (such as 8th-bit prefixing, repeat-count compression).

5.11. The 'take' and 'echo' Commands

Syntax: `take` *fn1*

`echo` *[text to be echoed]*

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. This means that a command file like this:

```
set speed 9600
connect
login myuserid
mypassword
etc
```

will *not* send "login myserid" or any of the following text to the remote system. To carry on a canned dialog, use the '`script`' command, described later.

The '`%`' command is useful for including comments in take-command files. It may only be used at the beginning of a line.

The '`echo`' command may be used within command files to issue greetings, announce progress, ring the terminal bell, etc. This command simply displays its text argument (almost) literally at the terminal; the argument may contain octal escapes of the form `\ooo`, where `o` is an octal digit (0-7), and there may be 1, 2, or 3 such digits, whose value specify an ASCII character, such as `\007` (or `\07` or just `\7`) for beep, `\012` for newline, etc.

Take-command files are in exactly the same syntax as interactive commands. Note that this implies that if you want to include special characters like question mark or circumflexes that you would have to quote with `^` 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 `^`.

Command files may be used in lieu of command macros, which have not been implemented in this version of C-Kermit. For instance, if you commonly connect to a system called 'B' that is connected to com2 at 4800 baud, you could create a file called `b` containing the commands

```
% C-Kermit command file to connect to System B thru com2
set line com2
set speed 4800
% Beep and give message
echo \007Connecting to System B...
connect
```

and then simply type 'take b' (or 't b' since no other commands begin with the letter 't') whenever you wish to connect to system B. Note the comment lines and the beep inserted into the 'echo' command.

For connecting to IBM mainframes, a number of 'set' commands are required; these, too, can be conveniently collected into a 'take' file like this one:

```
% Sample C-Kermit command file to set up current line
% for IBM mainframe communication
%
set parity mark
set handshake xon
set flow-control none
set duplex half
```

Note that no single command is available to wipe out all of these settings and return C-Kermit to its default startup state; to do that, you can either restart the program, or else make a command file that executes the necessary 'set' commands:

```
% Sample C-Kermit command file to restore normal settings
%
set parity none
set handshake none
set flow-control xon/xoff
set duplex full
```

An implicit 'take' command is executed upon your CKERMIT.INI file when C-Kermit starts up, upon either interactive or command-line invocation. The CKERMIT.INI file should 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 the command

```
set file warning on
```

in your CKERMIT.INI file.

Errors encountered during execution of take files (such as failure to complete dial or script operations) cause termination of the current take file, popping to the level that invoked it (take file, interactive level, or the command interpreter).

You may also use the redirection mechanism to cause C-Kermit to execute commands from a file:

```
kermit < cmdfile
```

or you can even pipe commands in from another process:

```
genkcmds | kermit
```

5.12. The 'connect' Command

The 'connect' command ('c' is an acceptable non-unique abbreviation) 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' command. 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 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-] (^]) (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 include:

- `c` Close the connection
- `h` Hangup the phone
- `q` Hangup the phone and quit C-Kermit
- `!` Enter a child CMD.EXE command line interpreter
- `0` (zero) send a null
- `^]` Send Control-] itself (whatever you have defined the escape character to be, typed twice in a row sends one copy of it).
- `b` Send a BREAK signal for about 275ms
- `L` Send a LONG BREAK signal for about 1.8s
- `\` Send the value of a backslash escape sequence
- `?` Display help information about these options

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.

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

Note that when in interactive command mode, C-Kermit reads its keyboard input from stdin and writes its screen output to stdout, allowing command-line redirection to be used as described in the previous section. However, 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.

5.13. The 'dial' command

Syntax: `dial` *telephone-number-string*

This command controls dialout modems; you should have already issued a 'set line' and 'set speed' command to identify the terminal device, and a 'set modem' command to identify the type of modem to be used for dialing. 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. The telephone-number-string may contain imbedded modem-dialer commands, such as comma for Hayes pause, or '&' for Ventel dialtone wait and '%' for Ventel pause (consult your modem manual for details).

At the time of this writing, support is included for quite a number of different modems. See the Unix C-Kermit manual for a complete list.

A number of these modems are not generally found connected to PCs. The most common modem type used with an OS/2 system is "Hayes compatible". Support for new modems is added to the program from time to time; you can check the current list by typing 'set modem ?'.

The device used for dialing out is the one selected in the most recent 'set line' command.

Example:

```
kermit -l com1 -b 1200
C-Kermit>set modem hayes
C-Kermit>dial 9,5551212
Connected!
C-Kermit>connect                        hint: abbreviate c
logon, request remote server, etc.
^]c                                     escape back
C-Kermit> ...
C-Kermit>quit                           hint: abbreviate q
```

this disconnects modem, and hangs up the line.

C-Kermit requires 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.

For Hayes dialers, two important switch settings are no. 1 and no. 6. Switch no. 1 should be normally be UP so that the modem can act according to your computer's DTR signal. Switch no. 6 should normally be UP so carrier-detect functions properly (but put it DOWN if you have trouble with the UP position). Switches no. 2 (English versus digit result codes) and no. 4 (Hayes echoes modem commands) may be in either position.

5.14. The 'script' Command

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.

C-Kermit's script facility operates in a manner similar to that commonly used by the Unix UUCP system's 'L.sys' file entries. 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
~~	tilde
~"	double quote
~x	XON (Control-Q)
~c	don't append a carriage return
~d	delay approx 1/3 second during send
~*o[o[o]]*	an octal character

`~w`*[d[d]]* wait specified interval during expect, then time out

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:

Using a Hayes-compatible modem, dial up a PAD, simulating pressing CR four times to get the '`PAD>`' prompt. (Note how '`~0`' stands for a null expect string *and* for a null send string - ie just send a carriage return.) Call a VAX system named 'ERCVAX'. Get the `..name:` prompt, and respond with the user name and password. Notice that the '`^`' character is used to continue the script command onto the next line.

```
set modem hayes
set line com2
set baud 1200
dial 0319871234
script ~0 ~0 ~0 ~0 ~0 ~0 ~0 ~0 PAD> CALL~sERCVAX ^
name: SMITH word: SECRET
```

Note that '`set line`' is issued *after* '`set modem`', but *before* '`set baud`' or other line-related parameters.

5.15. The 'help' Command

Syntax: `help`
or `help` *keyword*
or `help {set, remote}` *keyword*

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.

5.16. The 'exit' and 'quit' Commands

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

- Relinquish access to any communication line assigned via '`set line`'.
- Hang up the modem, if any, by dropping DTR.
- Close any open logs or other files.
- Exit the program.

After exit from C-Kermit, your current 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.

6. Terminal Emulation

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 comms port, the current baud rate 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. These options are detailed in the section on the connect command. 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 of line
ESC # 4       Double height and width emulation - bottom half of 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 - modes supported as shown below
ESC [ ? Pn l  Reset DEC private mode - modes supported as 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
```

7. 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.

8. 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. Answerback: We should have a '`set terminal answerback ...`' command to let us change the VT102 answerback message.

5. File type: The way Control-Z is handled could be better. There should be no '`set file type {binary, text}`'; instead we want a '`set ctrlz {on, off}`' rather like Kermit-MS. A better display of progress of a transfer is needed.

6. Login Scripts: The present login scripts implementation follows the Unix conventions of UUCP's '`L.sys`' file, rather than the normal Kermit 'INPUT/OUTPUT' style.

Appendix I
Invoking 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.

```
#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]);
        }
}
```

Index

Table of Contents