KERMIT USER GUIDE

Seventh Edition

Christine Gianone, Editor

Columbia University Center for Computing Activities
New York, New York 10027

May 26, 1988

PREFACE

Kermit is the name of a protocol for transferring files from one computer to another over ordinary asynchronous terminal connections. Kermit programs have been written for many different computers, and in general any two computers that have Kermit programs can exchange sequential files correctly and completely. This manual gives a brief and general overview of what Kermit is and how to use it, but consists mostly of detailed instructions for use and installation of specific Kermit programs.

For a more detailed introduction to Kermit, complete with illustrations, diagrams, and tutorials, consult the book *Kermit, A File Transfer Protocol*, by Frank da Cruz, Digital Press, Bedford MA (1987), ISBN 0-932376-88-6, DEC order number EY-6705E-DP (phone 1-800-343-8321). The Kermit book describes Kermit in detail, from the points of view of the beginner, the user, the computer professional who must install Kermit programs or support their use, and the programmer who wishes to write new Kermit implementations. Also included are general introductions to computers, data communications, and file organization, plus a detailed troubleshooting guide, bootstrapping hints, and various appendices and tables. The latter half of the book is taken up by a complete description of the Kermit file transfer protocol, with programming examples in the C language, plus some analysis and comparisons of Kermit with other popular protocols such as Xmodem.

The seventh edition of the *Kermit User Guide* (May 1988) includes chapters on new releases of most major Kermit programs, including MS-DOS Kermit 2.30, VAX/VMS Kermit 3.3, Portable IBM Mainframe Kermit 4.0, Unix Kermit 4E, Macintosh Kermit, CP/M-80 Kermit 4.09, plus a new chapter on PDP-11 Kermit.

History and Acknowledgements

The Kermit file transfer protocol was designed at the Columbia University Center for Computing Activities (CUCCA) in 1981-82 by Bill Catchings and Frank da Cruz. Bill wrote the first two programs, one for the DECSYSTEM-20 and one for a CP/M-80 microcomputer.

The initial objective was to allow users of our DEC-20 and IBM 370 timesharing systems to archive their files on microcomputer floppy disks. The design owes much to the ANSI and ISO/OSI models, and some ideas were borrowed from similar projects at Stanford University and the University of Utah. The protocol was designed to accommodate the "sensitive" communications front end of the full-duplex DEC-20 system as well as the peculiarities of half-duplex IBM mainframe linemode communications. The protocol was soon implemented successfully on our IBM mainframe systems under VM/CMS by Daphne Tzoar of CUCCA.

Meanwhile it was becoming apparent that Kermit was useful for more than just file archiving; IBM PCs were beginning to appear in the offices and departments, and there arose a general need for file transfer among all our systems, as well as a need to use the IBM PCs as terminals. Daphne soon had prepared an IBM PC implementation.

After our initial success with Kermit, we presented it at conferences of user groups like DECUS and SHARE, and began to get requests for it from other sites. Since we had written down a description of the protocol, some sites wrote their own implementations for new computers, or adapted one of our implementations to run on additional systems, and sent back these new versions to us so that we could share them with others. In this way, Kermit has grown to support nearly 300 different machines and operating systems; it has been sent on magnetic tape or diskette from Columbia University to nearly ten thousand sites all over the world, and has reached many thousands more through various user groups and networks.

Thanks to the hundreds of individuals and institutions who have contributed to the Kermit storehouse over the years.

The Kermit protocol was named after Kermit the Frog, star of the television series *THE MUPPET SHOW*; the name Kermit is used by permission of Henson Associates, Inc., New York.

Disclaimer

Neither Columbia University, nor the editor, nor the authors of the individual chapters, nor any individual or institution contributing Kermit programs or documentation to the Columbia University Kermit Distribution, acknowledge any liability for any claims arising from use or misuse of Kermit programs or for inaccuracies in the documentation or bugs in the programs. Kermit programs are produced on a voluntary basis and contributed freely for public use in the hope that they will be useful, but without any kind of warranty or guarantee, or any commitment to address or fix problems. In practice, Kermit programs and documentation are contributed in good faith, and will be supported on a best-effort basis, time and other commitments permitting.

Customizing This Manual

Although the *Kermit User Guide* was produced at Columbia University, all attempts have been made to keep it free of site-specific information. However, due to the large number of Kermit implementations, descriptions of each one would make the manual prohibitively thick. Therefore, the manual is sent from Columbia with specific documentation about a selection of systems. Some of these descriptions may not be of interest at your site, while others that are may be lacking.

Each site, upon receiving a Kermit tape, may decide which versions of Kermit are important to it, and include the appropriate documentation in this manual. This is most conveniently done if your site has the Scribe text formatting system (from UNILOGIC Ltd in Pittsburgh PA, USA), with which this manual was produced. Scribe runs on a wide variety of systems. There are also Scribe subsets, such as Perfect Writer and Final Word, that run on various microcomputers. Many have asked why Scribe is used for Kermit manuals instead of TeX. The answer is simply that TeX can only produce output for typesetters, not plain-text ASCII files, which are necessary for online documentation.

The system-specific parts of the Kermit User Guide are included with "`@INCLUDE`" statements at the end of the Scribe source file for this manual, whose filename is `KUSER.MSS`. You may add or delete `@INCLUDE` statements to suit your needs, and run the result through the text formatter to produce a customized manual. If you do this, you should include an indication on the title page that the manual has been customized for your site.

Not all system-specific documentation is provided in `.MSS` (Scribe input) format, since some Kermit contributors do not have Scribe at their sites. In that case, you will either have to add Scribe formatting commands, or else enclose the whole subfile in `@BEGIN(VERBATIM)...@END(VERBATIM)` brackets (and replace all atsigns (`@`) in the text with double atsigns (`@@`)).

If you do not have SCRIBE, you may still use an editor to delete or add sections to the finished documentation file, though the results will not be as satisfactory -- the table of contents, index, cross references, and page numbers will not be automatically adjusted.

If you are running a version of Kermit for which adequate documentation has not been provided (after all, this is a distributed, volunteer effort!), please feel free to write some, preferably in Scribe input format, and send it back to Columbia so that others may benefit from it. Likewise if you produce a new implementation of Kermit. If you don't know Scribe, you can use one of the existing chapters as a model.

How To Get Kermit

The Kermit software is free and available to all, source code and documentation included, from a variety of sources. For example, most universities are connected to academic computer networks from which the Kermit files at Columbia University can be reached. The Kermit files are also available from many user groups, dialup information or bulletin board services, diskette reproduction services, and private volunteers.

Kermit software is not in the public domain. The Kermit manuals and most Kermit programs bear copyright notices to protect Columbia University and the various contributors from having their work taken by others and sold as a product, for profit. This is not to say that the Kermit file transfer protocol can never be included as a feature of a commercial product; the conditions under which this may be done are spelled out in a flyer *POLICY ON COMMERCIAL USE AND DISTRIBUTION OF KERMIT*.

Columbia University distributes Kermit programs by mail order on various magnetic media (primarily 9-track reel-to-reel tape and certain kinds of diskettes), charging a distribution fee to defray costs for media, printing, postage, materials, labor, and computing resources. This is not a software license fee; no license is required. To receive a current list of Kermit implementations, the statement on commercial policy, and a Kermit order form, write to:

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

Everyone is free to copy and redistribute Kermit programs and documentation, and is encouraged to do so, with the following stipulations: Kermit programs should not be sold for commercial gain; credit should be given where it is due; and new material should be sent back to Columbia University at the address above so that we can maintain a definitive and comprehensive set of Kermit implementations for further distribution.

Since new Kermit programs are added -- and old ones improved -- so frequently, sites that use Kermit heavily are encouraged to contact Columbia for updates two or three times a year for news.

-- PLEASE USE KERMIT ONLY FOR PEACEFUL AND HUMANE PURPOSES --

Organization of This Manual

Chapter 2, *How to Use Kermit*, describes the basics of text file transfer, and shows some specific examples. If you follow the examples but you can’t make a terminal connection or you can’t transfer files successfully, consult Chapter 3, *When Things Go Wrong*.

If you expect to be a heavy user of Kermit, you should read Section 4, *Kermit Commands*, which describes most of Kermit’s features and commands. You may find that familiarity with the material in this section will help you get past difficulties that can crop up when you are making new kinds of connections or transferring unusual kinds of files. You will also find descriptions of some advanced file management features that have been omitted from the earlier sections.

The subsequent chapters describe selected popular Kermit programs in detail. You should read the appropriate section for each system you expect to use; each section describes the file naming conventions and other system features that are important to Kermit users, and lists the Kermit commands for that system mainly in terms of their differences from the "ideal" Kermit described in section 4.

1. Introduction

There is an ever-increasing need to move information from one computer to another. Information can be exchanged using magnetic media -- tapes or disks -- or over networks. Networks are expensive, and when your computer is not connected to one (or to the right one), you must find other means to transfer information. In the early days of computing, magnetic media formats were relatively standardized, but with the arrival of microcomputers things have changed: most microcomputer users have no access to tapes, and disk formats are incompatible between most microcomputer makes and models. Even when disk formats agree, the disk must be physically moved from one system to the other in order for information to be exchanged -- the effort and delay can be significant if the systems are widely separated.

The telecommunication line provides a cheap and widely available alternative to networks and magnetic media. Asynchronous telecommunication is the method used by most terminals to connect to most computers. When dedicated "hardwired" connections, such as those found between a timesharing computer and its local terminals, are not available, computer users can create their own dialup connections with a telephone and a modem.

Most computers come equipped with asynchronous telecommunications interfaces, or "serial ports", which allow them to act as, or communicate with, terminals. The question is how to use the serial port to exchange data. Fortunately, the standards for connecting terminals to computers are almost universally followed: connector configuration (DB-25 or DB-9), transmission signals (EIA RS-232), a commonly accepted set of transmission speeds (baud rates), and a convention for encoding characters in storage and during transmission (ASCII). These standards provide the physical medium and the data format, but they do not specify a process for exchanging data.

1.1. Why Kermit?

When data is transmitted from one computer to another; the receiving computer has to be instructed to take in the data and put it somewhere, and it also needs a way of ensuring that the data has been received correctly and completely in spite of several factors that will tend to interfere with this process:

1. *Noise* -- It is rarely safe to assume that there will be no electrical interference on a line; any long or switched data communication line will have occasional interference, or noise, which typically results in garbled or extra characters. Noise corrupts data, perhaps in subtle ways that might not be noticed until it's too late.

2. *Synchronization* -- Data must not come in faster than the receiving machine can handle it. Although line speeds at the two ends of the connection must match before communication can take place, the receiving machine might not be able to process a steady stream of input at that speed. Its central processor may be too slow or too heavily loaded, its buffers too full or too small, or its disk too slow. The typical symptom of a timing problem is lost data; most operating systems will simply discard incoming data they are not prepared to receive.

To prevent corruption of data and to synchronize communication, cooperating computers can send special messages to one another along with the data. Intermingling of control information with data together requires a set of rules for distinguishing messages from data, and specifying what the messages are and the actions associated with each message. Such a set of rules is called a *protocol*.

Kermit is a file transfer protocol. It is specifically designed for transfer of sequential files over ordinary telecommunication lines. Kermit is not necessarily better than other terminal-oriented file transfer protocols but it is free, it is well documented, and it has been implemented compatibly on a wide variety of microcomputers, PCs, workstations, minicomputers, mainframes, and supercomputers.

1.2. How Kermit Works

Kermit transfers data by encapsulating it in *packets* of control information. This information includes a synchronization marker, a packet sequence number to allow detection of lost packets, a length indicator, and a "block check" to allow verification of the data, as shown in Figure 1-1. The MARK (usually an ASCII Control-A

```
+------+------+------+------+--------- - - - -+-------+
| MARK | LEN  | SEQ  | TYPE | DATA            | CHECK |
+------+------+------+------+--------- - - - -+-------+
```

Figure 1-1: A Kermit Packet

character) appears at the beginning of the packet. The next character is a length field (LEN), specifying how long the rest of the packet is. The sequence number (SEQ) is used to detect lost or duplicated packets; retransmission is requested for lost packets and duplicate packets are discarded. The TYPE field specifies whether the packet contains data or control information. The CHECK field contains a quantity obtained by combining all the other characters in the packet together in one of several ways; the sender computes this value and sends it. The packet receiver also computes the value and checks it against the value sent; if they agree, the packet is accepted; if they disagree, then the packet has been corrupted and retransmission is requested. The DATA field contains up to 90 characters of data. All fields except the mark are encoded as printable ASCII characters, to prevent host or network interference. Figure 1-2 shows how a typical file transfer proceeds. Figure 1-2 does not show how Kermit recovers from errors. Very briefly, here's how it works:

- If a packet is corrupted in transit by noise or loss of characters, the block check will be wrong. A file receiver will NAK ("negatively acknowledge") a corrupted packet, which causes the sender to retransmit the same packet (or, alternatively, it will ACK the last correctly received packet again). A file sender only receives ACKs and NAKs from the receiver; a corrupted ACK, or a NAK, from the receiver causes the sender to retransmit its most recent packet.
- If the file sender does not receive an ACK within the prescribed timeout interval, it retransmits the same packet. If the file receiver does not receive an expected packet within the timeout interval, it sends a NAK for the expected packet (or another ACK for the most recently correct packet).

Many encoding, compression, block check, timeout, and packet length options are provided. These options are automatically negotiated by the two Kermit programs when they initially make contact, and the greatest common set of features is used. For this reason, any two Kermit programs should be able to communicate successfully, from the oldest, most bare-bones version, to the newest, most feature-laden version. The protocol is described in detail in the Kermit book.

```
Sender                       Receiver

Send-Init ------------->               Sender and Receiver exchange greetings
     <-------------------- ACK

File-Header ----------->               Sender sends first file name to Receiver
     <-------------------- ACK         Receiver acknowledges

File-Data ------------->               Sender sends first file data packet
     <-------------------- ACK

File-Data ------------->               Sender sends second file data packet
     <-------------------- ACK

File-Data --xx~~~p'''-->               Third data packet is corrupted by noise
     <-------------------- NAK          and Receiver negatively acknowledges it

File-Data ------------->               Sender retransmits third packet
     <-------------------- ACK

File-Data packets are sent and acknowledged until the whole file is sent

End-Of-File ----------->               Sender indicates first file is complete
     <-------------------- ACK

File-Header ----------->               Name second of file
     <-------------------- ACK

File-Data ------------->               First data packet for second file
     <-------------------- ACK

File-Data packets are sent and ack'd until the whole file is sent

End-Of-File ----------->               Sender indicates second file is complete
     <-------------------- ACK

End-Of-Transaction ---->               Sender indicates no more files to come
     <-------------------  ACK
```

Figure 1-2: Kermit File Transfer

2. How to Use Kermit

Kermit embodies a set of rules for transferring files reliably between two computers. In general, one computer is a large system (a *host*, for instance a timesharing system with many terminals), and the other is a personal computer (PC). The host believes that the PC is an ordinary terminal. In order for the Kermit protocol to occur, a Kermit *program* must be running on each end of the communication line -- one on the host, one on the PC.

Your task is just to get the two Kermits started. You have to use a single keyboard and screen to talk to two different computers, two different programs. Let's talk about a common case: you are sitting at a personal computer (PC), which has a serial communication port. The serial port is connected to a host computer using, say, a dialup modem.

Normally, when you use your PC, you are "talking" directly to it; your commands are interpreted directly by the PC's operating system (CP/M, MS-DOS, UNIX, etc), or by some program that runs on the PC (an editor, a text formatter, space invaders...). The version of Kermit on your PC is a program like any other, but it has a special ability to either interpret your commands directly, like other programs, or to pass everything you type through to the other, remote computer. When you tell Kermit to CONNECT, it sends every character you type out the serial port, and it puts every character that comes in the serial port onto the screen. This is called "terminal emulation" -- one computer acts as if it were a terminal to the other. You are now "talking" to the remote computer, and the PC is (mostly) ignoring you.

Kermit, like most interactive programs, has a *prompt*. The prompt is a string of characters it types on the left margin to indicate that it is ready for you to type a command. Kermit's prompt is normally "`Kermit-`*xx*`>`". The *xx* identifies the implementation of Kermit; the Kermit that runs on MS-DOS systems is called "Kermit-MS" and its prompt is "`Kermit-MS>`"; the Kermit that runs on Z80 and 8080-based microcomputers is called "Kermit-80" and its prompt is "`Kermit-80>`", and so forth. If you become confused about which Kermit you are talking to, the prompt should provide a clue. In addition, most Kermits print an informative message like

```
[Connecting to remote host, type CTRL-]C to return]
```

when you CONNECT, and type another message like

```
[Connection closed, back at PC]
```

when you return.

Having "connected" to the host, there must be a way for you to "get back" to the PC. This is accomplished by an *escape sequence*. As Kermit passes your characters through to the host, it checks each one to see if it's a special predefined *escape character*. When the PC sees this character, it stops ignoring you -- you are once again "talking" to the PC, not to the host. The escape character is normally chosen to be one that you will not need to type while talking to the host, and one that is hard to type by accident -- it's usually a *control character*, such as Control-`]`, which is entered by holding down the key marked CTRL or CONTROL and typing the indicated character (in this case, a right bracket "`]`"). The CTRL key works just like a SHIFT key. Control characters are written either as Ctrl-A or `^A`, where A is the character to be typed while holding down the Ctrl key.

2.1. Transferring a File

From system command level on your PC, first run the Kermit program. Then tell Kermit to CONNECT you to the host. Now you're talking to the remote host -- at this point you must log in, and then run Kermit on the host.

Now you have a Kermit program on each end of the connection. The next step is to tell *each* Kermit what to do. Suppose you want to transfer a file from the host to the PC; you would first tell the host Kermit to SEND the file, then "escape" back to the PC Kermit and tell it to RECEIVE the file. The transfer begins -- you can sit back and watch, or go make yourself a sandwich. The PC Kermit will produce a running display on your screen as the transfer proceeds, and will notify you when it is complete.

The desired file should now be on your PC's disk. The Kermit protocol has ensured that the file arrived correctly and completely. Now you must clean up after yourself: CONNECT back to the host, exit from Kermit on the host, log out from the host, "escape" back to PC Kermit and exit from it. Now you can do whatever you had planned for your file -- edit it, print it on your PC printer, etc. Transferring a file in the other direction works the same way, but with the SEND and RECEIVE commands exchanged.

The Kermit protocol, and most Kermit programs, allow you to send text files reliably from the host to the PC, from the PC to the host, from host to host, or PC to PC, usually without any special regard for the nature of the particular machines involved. Most implementations also allow files to be sent in groups, with a single command, such as `"send *.*"` The scenario for each of these is the same as above -- only the details of how to establish the actual connection differ.

Kermit works best with "printable" files -- files composed only of letters, digits, punctuation marks, carriage returns, tabs, and so forth -- since these can be represented on almost any kind of computer. Kermit is also able to transfer "binary" files -- files such as executable programs -- composed of arbitrary bit patterns, but binary files normally are meaningful only to the kind of computer on which they are generated. Nevertheless, Kermit can usually move such files from system A to system B (where they are not much use) and back to system A in their original condition, although in most cases special measures must be taken to accomplish this.

Let's look at some more concrete examples. First you need to know what the basic Kermit commands are.

2.2. Basic Kermit Commands

These are generic descriptions of the most basic Kermit commands. Detailed descriptions will come later. In these descriptions, *local* refers to the system that you are using directly, *remote* refers to the system to which you are CONNECTed via Kermit. Commands may take one or more operands on the same line, and are terminated by a carriage return.

SEND *filespec*
: Send the file or file group specified by *filespec* from this Kermit to the other. The name of each file is passed to the other Kermit in a special control packet, so it can be stored there with the same name. A file group is usually specified by including "wildcard" characters like "`*`" in the file specification. Examples:

```
send foo.txt
send *.for
```

Some implementations of Kermit may not support transfer of file groups; these versions would require a separate SEND command for each file to be transferred.

RECEIVE
: Receive a file or file group from the other Kermit. If an incoming file name is not legal, then attempt to transform it to a similar legal name. Options are be provided for handling filename collisions.

CONNECT
: Make a terminal connection to the remote system. To "escape" from the terminal connection, type Kermit's *escape character* (e.g. `CTRL-]`, control-rightbracket), followed by the letter "C" for "Close Connection".

SET
: Establish various nonstandard settings, such as CONNECT escape character, file characteristics, communication line number, speed (baud rate), parity, or flow control. All of these are explained later.

SHOW
: (or STATUS) Display the values of SET options.

HELP
: Type a summary of Kermit commands and what they do.

EXIT
: Exit from Kermit back to the host operating system.

`?`
: Typed almost anywhere within a Kermit command: List the commands, options, or operands that are possible at this point. This command may or may not require a carriage return, depending on the host operating system.

2.3. Real Examples

Kermit can be used in several ways: from a PC that is connected to a larger host computer; from a host computer which is connected to another host; from one PC to another.

2.3.1. PC to Host

In this example, the user is sitting at an IBM Personal Computer (PC), which is connected through its serial port to a DEC VAX/VMS host computer. The IBM PC is *local*, the VAX is *remote*. This example will also apply almost literally to any other microcomputer implementation of Kermit.

You have started up your PC and have the Kermit program on your disk. Begin by running Kermit on the PC. Use Kermit's CONNECT command to become a terminal to the VAX. In fact, the PC emulates the popular DEC VT-102 (VT-100), so so it is desirable to tell the host that your terminal is of this type. Login on the VAX and run Kermit there. Here is an example of this procedure with commands you type underlined; the material lined up on the right is commentary, not system typeout or part of a command:

```
A>Kermit                   Run Kermit on the PC.
Kermit-MS V2.30
IBM-PC Kermit-MS: V2.30  8 Jan 88
Type ? for help

Kermit-MS>                 This is the Kermit prompt for the PC.
Kermit-MS>connect          Connect to the VAX.
[Connecting to host, type Control-] to return to PC]

                           You are now connected to the VAX.

Welcome to CUMIN, MicroVMS V4.6    i(The system prints its herald.)

Username: my-id            Type your user ID.
Password: my-password      Type your password.

(Various greeting or notice messages are displayed.)

$                          This is the VMS system prompt.
$ Kermit                   Run Kermit on the VAX.

VMS Kermit-32 version 3.3.111
Default terminal for transfers is: _TXA0:
Kermit-32>                 This is VMS Kermit's prompt.
```

You are now ready to transfer files between the two machines.

The following example illustrates how to send files from the VAX to the PC. Note the use of the "*" *wildcard* character to denote a *file group*.

```
Kermit-32>send *.for       Send all my FORTRAN files.
^]c                        Now return back to the PC by
                           typing the escape sequence, in this case
                           ^]C  (Control-] followed by "C")
[Back at PC.]
Kermit-MS>receive          Tell the PC that files are coming.
```

If you take more than about 5 seconds to get back to Kermit-MS and issue the RECEIVE command, the first packets from Kermit-32 may arrive prematurely and appear on your screen, but no harm will be done because the packet will be retransmitted by the VAX until the PC acknowledges it.

Once the connection is established, the PC will show you what is happening -- it first clears the screen and waits for incoming packets; as packets arrive, the current file name and packet number will be continuously displayed on the screen. When the PC's `"Kermit-MS>"` prompt returns to your screen (with an accompanying beep to catch your attention) the transfer is done. Notice the screen display; the status should be indicated as "complete". If not, an error has occurred and an appropriate message should be displayed to tell you why.

After you're finished transferring files, you must CONNECT back to the VAX host, EXIT from Kermit there, logout, and "escape back" to the PC as you did previously:

```
Kermit-MS>connect         Get back to the VAX.
[Connecting to host. Type CTRL-]C to return to PC.]
Kermit-32>                Here we are.
Kermit-32>exit            Get out of VMS Kermit.
$ logout                  Logout from the VAX.

MY-ID   logged out at 25-JAN-1988 15:12:27.85

^]c                       Now "escape" back to the PC,
[Back at PC.]
Kermit-MS>exit            and exit from the PC's Kermit.
A>                        Now you see the DOS prompt again.
```

The files you transferred should now be on your PC disk.

To send files from the PC to the VAX, follow a similar procedure. First follow the instructions in the previous section to log in to the VAX through the PC. Then in response to the host Kermit's "`Kermit-32>`" prompt you type RECEIVE rather than SEND. Now escape back to the PC and use the SEND command to send the local PC files to VAX. The PC will show you the progress of the transmission on its screen.

When the "`Kermit-MS>`" prompt indicates that the transmission is complete you should follow the procedure shown above to logout from the VAX host, except that you may first wish to confirm that the files have been stored correctly in your directory on the VAX.

2.3.2. Host to Host

A "host" is considered to be a large or multi-user system, whose distinguishing characteristic is that it has multiple terminals. Use of Kermit for host-to-host file transfers differs from the PC-to-host case in that the line your terminal is connected to is not the same as the line over which the data is being transferred, and that some special SET commands may have to be issued to allow one Kermit to conform to unusual requirements of the other host.

In this example, you are already logged in to a Unix system, and you use an *autodialer* to connect to an IBM 370-series system running VM/CMS through Unix device `/dev/tty12`.

```
% kermit
C-Kermit, 4E(070) 24 Jan 88, 4.2 BSD
Type ? for help
C-Kermit>set modem hayes
C-Kermit>set line /dev/tty12
C-Kermit>set speed 1200
C-Kermit>dial 7654321
Connected!
```

Other methods exist for connecting two hosts with a serial line. For instance, dedicated hookups can be made by running an RS-232 "null modem" cable between TTY ports on the two systems (null modem cables, RS-232 signals, modems, and other data communication apparati are described in detail in the Kermit book). The following procedure would be the same in any case, once a connection is made. The four "set" commands below are necessary when connecting to IBM mainframes in "linemode" (as opposed to full-screen 3270 mode; if you don't use IBM mainframes, you can ignore them for now).

```
C-Kermit>set duplex half         The IBM mainframe is half duplex.
C-Kermit>set flow none           No full duplex XON/XOFF.
C-Kermit>set handshake xon       Use XON for line turnaround handshake.
C-Kermit>set parity mark         Our IBM system uses mark parity.
C-Kermit>connect                 Connect to the mainframe.
Connecting thru /dev/tty31, speed 1200.
```

```
The escape character is CTRL-\ (28).
Type the escape character followed by C to get back,
or followed by ? to see other options.
                                         (Type carriage return here.)
VM/370 ONLINE                            The IBM system prints its herald.

.login myuserid mypassword               Login to IBM system.

LOGON AT 15:33:02 EST MONDAY 02/08/88
CUVMA CMS 3.1 8409 01/25/85
.
.Kermit
Kermit-CMS Version 4.0 (87/12/17)
Enter ? for a list of valid commands

Kermit-CMS>.send profile exec
^\c                                      C-Kermit's escape sequence typed here.
[Back at Local System]
C-Kermit>receive                         Tell Unix Kermit to RECEIVE.
```

The transfer takes place now; Unix Kermit will print the names of incoming files, followed by dots or percents to indicate the packet traffic (a dot for every 4 packets successfully transferred, a percent for every timeout or retransmission). The transfer is complete when when you see "`[OK]`", a beep is sounded, and the C-Kermit prompt next appears. At that point we connect back to the remote IBM system, exit from the remote Kermit and log out.

```
.
profile.exec ..%%.[OK]
C-Kermit>connect             Get back to mainframe and clean up.

Kermit-CMS>.
Kermit-CMS>.exit
R;
.
SP/CMS
.logout

CONNECT= 00:03:01 VIRTCPU= 000:00.12 TOTCPU= 000:00.60
LOGOFF AT 15:40:13 EST MONDAY 02/08/88

^\c                          Type C-Kermit's escape sequence
[Back at Local System]
C-Kermit>exit                All done with Kermit.
```

That's the whole procedure. The file is in your Unix directory, completely readable, as `profile.exec` -- note that Kermit-CMS translated from the IBM EBCDIC character encoding into standard ASCII, and converted the space between the file name and file type to a dot.

To send a file from the local host to the remote host, we would merely have reversed the SEND and RECEIVE commands in the example above.

2.3.3. Micro to Micro

Kermit also works between personal computers (microcomputers, workstations). The difference here is that commands are typed on *two* keyboards, rather than a single one. This is because a personal computer normally only accepts commands from its own keyboard. If one PC Kermit CONNECTs to another, there will normally be no program on the other side to listen.

You can make the physical connection between two micros in two ways: direct or dialup. If the two units are in close proximity (say, 50 feet or less), you can connect their serial ports with a null modem cable.

Connections at longer distances can be made via dialup, providing the required modems are available (one side needs autoanswer capability), or using any kind of dedicated or switched circuit that may be available -- CBX, port contention unit, almost anything you can plug an EIA connector into.

In this example, a DEC VT180 "Robin" CP/M-80 microcomputer is connected to a Intertec "SuperBrain" CP/M-80 micro, using a female-to-male null modem cable (these systems have nostalgia value, being among the first for which Kermit programs were written). The connection can be tested by running Kermit and issuing the CONNECT command on both ends: typein from each micro should appear on the screen of the other.

Suppose you want to send a file `FOO.HEX` from the Robin to the SuperBrain. Proceed as follows:

1. Run Kermit on the SuperBrain, and give the RECEIVE command:

```
A>Kermit
Intertec SuperBrain Kermit-80 - V4.09
Kermit-80>receive
```

2. Run Kermit on the Robin, and give the SEND command for `FOO.HEX`.

```
A>Kermit
DEC VT18X Kermit-80 - V4.09
Kermit-80>send foo.hex
```

   Watch the packets fly. When you get the next `Kermit-80>` prompt, the transfer is done, and you can EXIT from both Kermits.

The key point is to start the *receiving* end first -- some microcomputer Kermits do not include a timeout facility, and if the receiver is not ready to receive when the sender first sends, there will be a protocol deadlock.

2.4. Another Way -- The Kermit Server

So far, we have been describing the bare-bones version of the Kermit protocol. An optional extension to the protocol includes the concept of a *Kermit server*. A Kermit server is a Kermit program that does not interact directly with the user, but only with another Kermit program. You do not type commands to a Kermit server, you merely start it at one end of the connection, and then type all further commands at the other end.

Not all implementations of Kermit can be servers, and not all know how to talk to servers -- but most of the major ones can and do. The server is run on the remote computer, which would normally be a timesharing system, such as an IBM mainframe, a Unix system, or VAX/VMS, but may be a minicomputer or even a PC. It depends on whether the particular Kermit program has a "server" command. You must still connect to the remote host to log in and start the server, but you no longer have to tell one side to SEND and the other to RECEIVE, nor must you connect back to the remote side to clean up and log out when you're done. Using the server, you can do as many send and receive operations as you like without ever having to connect back to the remote host. Some servers also provide additional services, such as directory listings, file deletion, or disk usage inquiries.

A Kermit server is just a Kermit program running in a special way. It acts much like ordinary Kermit does after you give it a RECEIVE command -- it waits for a message from the other Kermit, but in this case the message is a command saying what to do, normally to send or to receive a file or group of files. After escaping back to the local system, you can give as many SEND and GET commands as you like, and when you're finished transferring files, you can give the BYE command, which sends a message to the remote Kermit server to log itself out. You don't have to connect back to the remote host and clean up. However, if you *want* to connect back to the host, you can use the FINISH command instead of BYE, to shut down the Kermit server on the remote host without logging it off, allowing you to CONNECT back to your job there.

Here's an example of the use of a Kermit server. The user is sitting at an IBM PC and a DECSYSTEM-20 is the remote host.

```
A>Kermit                      Run Kermit on the micro.
Kermit-MS V2.30
IBM-PC Kermit-MS: V2.30  8 Jan 88
```

```
Type ? for help

Kermit-MS>                  This is the Kermit prompt for the PC.
Kermit-MS>connect           Connect to the VAX.

CU20B                       The DEC-20 prints its herald.
@login my-id password       Log in normally.
```

(The DEC-20 prints various login messages here.)

```
@Kermit                     Run Kermit-20 normally
Kermit-20>server            Tell it to be a server.

Kermit Server running on DEC-20 host.  Please type your escape
sequence to return to your local machine.  Shut down the server by
typing the Kermit BYE command on your local machine.

^]c                         Now escape back to the PC.

Kermit-MS>get *.pas         Get all my DEC-20 Pascal programs.
Kermit-MS>send foo.*        Send all the "foo" files from my PC.
Kermit-MS>exit              Exit from Kermit back to DOS.
A>
```

(Here you can do some work on the micro, edit files, whatever you like.)

```
A>Kermit                    Run Kermit-80 some more.
Kermit-MS>send file.pas     Send another file.
Kermit-MS>bye               That's all.  Shut down the Kermit server.
A>                          Back at DOS automatically.
```

This is *much* simpler. Note that once you've started the Kermit Server on the remote end, you can run Kermit as often as you like on the micro without having to go back and forth any more; just make sure to shut the server down when you're done by typing the BYE command.

If it's so much simpler, why not do it this way all the time? You can, provided your remote Kermit has a "server" command. But server operation, plus the special commands the local Kermit needs to communicate with the server (GET, REMOTE, BYE, FINISH) are optional Kermit features, so some Kermit programs might not have them. All Kermit programs, however, should provide the basic SEND/RECEIVE mode of operation.

Here are the basic commands available for talking to servers:

SEND *filespec*
: Sends a file or file group from the local host to the remote host in the normal way.

GET *filespec*
: Ask the remote host to send a file or file group. Example:

```
get *.c
```

This command is exactly equivalent to typing "send `*.c`" at the remote host followed by "receive" on the local host. Note that the local Kermit does not attempt to validate the filespec. If the server cannot access the specified file(s), it will send back an appropriate error message. Please note that GET and RECEIVE are not the same! RECEIVE tells Kermit to passively wait for a file. GET actively sends a request to a Kermit server to send the named file.

REMOTE *command* [*argument*]
: Ask the server to perform the specified command, and send the results to your screen. Not all servers are capable of performing REMOTE commands; those that can most commonly provide REMOTE DIRECTORY, REMOTE DELETE, REMOTE SPACE, and similar file management services.

BYE
: Shut down the remote server and exit from Kermit. This will cause the job at the remote end to log itself out. You need not connect back and clean up unless you get an error message in response to this command.

FINISH
: Shut down the server without having it log itself out, and don't exit from Kermit. A subsequent CONNECT command will put you back at your job on the remote host, at system command level.

Server operation is not limited to mainframes. Some PC Kermit implementations can also act as servers, notably MS-DOS and Unix. For instance, an IBM PC at the office with an autoanswer modem can be left in server mode at the end of the day, and then dialed up from home in the evening for file transfer.

3. When Things Go Wrong

Connecting two computers can be a tricky business, and many things can go wrong. Before you can transfer files at all, you must first establish terminal communication. But successful terminal connection does not necessarily mean that file transfer will also work. And even when file transfer seems to be working, things can happen to ruin it. The following sections treat a few basic problems. See the troubleshooting section of the Kermit book for greater detail.

3.1. Basic Connection Problems

If you have a version of Kermit on your microcomputer, but the CONNECT command doesn't seem to work at all, please:

- Make sure all the required physical connections have been made and have not wiggled loose. If you are using a modem, make sure the carrier light is on.
- If you have more than one connector on your micro, make sure you are using the right one.
- Make sure that the port is set to the right communication speed, or *baud rate*. Some versions of Kermit have a built-in SET BAUD or SET SPEED command, others require that you set the baud rate using a system command or setup mode before you start the Kermit program. Some versions of Kermit have SHOW or STATUS commands that will tell you what the current baud rate is.
- Make sure that the other communication line parameters, like parity, bits per character, handshake, and flow control are set correctly.

You may have to consult the appropriate manuals for the systems and equipment in question.

If all settings and connections appear to be correct, and communication still does not take place, the fault may be in your modem. Internal modems (i.e. those that plug in to a slot inside the microcomputer chassis) are *not* recommended for use with Kermit unless they totally mimic the asynchronous serial port hardware they purport to replace, or unless the Kermit program claims to support the particular internal modem. Many microcomputer Kermit programs are written to control the communication hardware explicitly; internal modems can interfere with that control.

Even external modems can cause trouble -- the "smarter" they are, the more potential danger of disagreement between the modem and the microcomputer about settings of baud rate, character framing, echo, and so forth. Make sure your modem is set up correctly (consult your modem manual).

3.2. Terminal Connection Works But The Transfer Won't Start

Once you've made a terminal connection to the remote system, you will generally also be able to transfer files. But not always. If Kermit's terminal emulation seems to work correctly, but a file transfer will not start at all, then something in the communication path is probably interfering with the packet data:

PARITY:

A device can impose *parity* upon the communication line. This means that the 8th bit of each character is used by the equipment to check for correct transmission. Use of parity will:

- Cause packet checksums to appear incorrect to the receiver and foil any attempt at file transfer. In most cases, not even the first packet will get through.
- Prevent the use of the 8th bit for binary file data.

If terminal connection works but file transfer does not, parity is the most likely culprit. To overcome this impediment, you should find out what parity is being used, and inform the Kermits one side or both (using the SET PARITY command) so that they can:

- Compose and interpret the checksums correctly.

- Employ a special encoding to allow 8-bit data to pass through the 7-bit communication channel.

Many packet-switched networks, such as GTE TELENET, require parity to be set, as do IBM mainframes and their front end processors.

ECHOING:
Some communication processors, typically front ends, echo their input. When this happens, every Kermit packet that is sent to it will bounce right back, causing no end of confusion. Some Kermit programs have been designed to ignore echoed packets, but others have not. If you encounter this problem, there are several possible solutions:

- Disable the front end echoing by typing some special command, if such a command is provided by the system.

- Some front ends respond to certain escape or control sequences as commands to turn off echoing, either from that point onward, or else on a per-line basis. In this case, the appropriate control sequence can be inserted between packets by Kermit programs instructed to do so, for instance using the SET PAD command.

- If the echoing cannot be disabled, then the two Kermit programs should be instructed to use differing packet start markers, using the SET START-OF-PACKET command -- for instance, one Kermit uses Control-A as usual, and the other uses Control-B. This can only be done if both Kermits have this SET command.

3.3. Special Characters

There is one problem that can prevent a file transfer from starting at all, or may crop up after the file transfer is underway. For instance, during a file transfer operation you might find your smart modem suddenly hanging up your current connection and placing a call to Tasmania. Or you might find that packets containing a certain character like "@" cannot be transmitted successfully.

This is the problem of "special characters". Some device in the communication path -- a front end, a port switcher, a multiplexer, a "smart" modem -- interprets certain characters in the data stream as commands rather than as data to be passed them along to the other side. Usually such equipment interferes only with the transmission of ASCII control characters; so long as Control-A and Carriage Return -- Kermit's normal packet start and end delimiters -- are not molested, then Kermit can operate. However, equipment may exist which swallows even printable characters. Since Kermit assumes that ALL printable ASCII characters (ASCII 40 through 176, octal) can be transmitted without interference or modification, such equipment will prevent Kermit file transfer unless its printable-character-swallowing features can be disabled.

3.4. The Transfer Starts But Then Gets Stuck

Once a Kermit file transfer has begun, there are certain conditions under which it can become stuck. Since many hosts are capable of generating timeout interrupts when input doesn't appear within a reasonable interval, they can resend unacknowledged packets or request that missing packets be retransmitted. But since not all Kermit programs are capable of timing out, a means for manual intervention is provided in the local Kermit -- you can type a carriage return on the keyboard of most micros to wake up and continue the transfer.

The following sections discuss various reasons why a transfer in progress could become stuck. Before examining these, first make sure that you really have a Kermit on the other end of the line, and you have issued the appropriate command: SEND, RECEIVE, or SERVER. If the remote side is not a server, remember that you must connect back between each transfer and issue a new SEND or RECEIVE command.

3.4.1. The Connection is Broken

Check the connection. Make sure no connectors have wiggled loose from their sockets. If you're using a modem, make sure you still have a carrier signal. Reestablish your connection if you have to.

If upon reconnection you get no response, maybe the remote host or the remote Kermit program crashed. Get back to command level on the local Kermit (on microcomputer implementations, you may be able to do this by typing about five RETURNs, or one or more Control-C's). Issue the CONNECT command so that you can see what happened. If the remote system has crashed then you will have to wait for it to come back, and restart whatever file that was being transferred at the time.

3.4.2. The Disk is Full

If your local floppy disk or remote directory fills up, the Kermit on the machine where this occurs will inform you and then terminate the transfer. You can continue the transfer by repeating the whole procedure either with a fresh floppy or after cleaning up your directory. Some Kermits also have a feature that allows you to keep incompletely received files; this would allow you go back to the sending system, extract the unsent portion of the file, and send it, and then append the two received portions together using an editor or other system utility. Kermit does not provide the ability to switch disks during a file transfer.

3.4.3. Transmission Delays

Packet transmission can be delayed by various agents: congested timesharing systems or networks, earth satellites, etc. When transmission delay exceeds the per-packet timeout interval for a significant length of time, the transfer could fail. Most Kermit programs provide commands that allow you to adjust the timeout interval or the packet transmission retry threshhold in order to accommodate to severe transmission delays.

3.4.4. Noise Corruption

If your connection is extremely noisy, packets will become corrupted -- and require retransmission -- more often. The probability that successive retransmissions will fail because of corruption rises with the noise level until it exceeds the retry threshhold, at which point the file transfer fails. There are several recourses. First, try to establish a new connection. If that is impractical, then use SET commands (when available) to reduce the packet length and increase the retry threshhold. Shorter packets reduce the probability that a particular packet will be corrupted and the retransmission overhead when corruption does occur, but they also increase the overall protocol overhead. In a noisy environment, you should also request a higher level of error checking (SET BLOCK 2 or 3).

3.4.5. Host Errors

Various error conditions can occur on the remote host that could effect file transmission. Whenever any such error occurs, the remote Kermit normally attempts to send an informative error message to the local one, and then breaks transmission, putting you back at Kermit command level on the local system.

3.5. File is Garbage

There are certain conditions under which Kermit can believe it transferred a file correctly when in fact, it did not. The most likely cause has to do with the tricky business of *file attributes*, such as text vs binary, 7-bit vs 8-bit, blocked vs stream, and so forth. Each system has its own peculiarities, and each Kermit has special commands to allow you to specify how a file should be sent or stored. However, these difficulties usually crop up only when sending binary files. Textual files should normally present no problem between any two Kermit programs.

4. Kermit Commands

An "ideal" Kermit program will be described here, which has most of the features specified in the Kermit book. Few Kermit programs will have all these commands or support all these options. The exact form of some of the commands may differ from version to version. Some Kermit programs may support system-dependent options not described here. The intention of this description is to provide a base from which specific Kermit programs can be described in terms of their differences from the "ideal."

4.1. Remote and Local Operation

In any connection between two Kermit programs, one Kermit is *remote* and the other is *local*. The remote Kermit is usually running on a mainframe, which you have CONNECTed to through a PC or other computer. When Kermit runs remotely, all file transfer is done over the job’s controlling terminal line -- the same line over which you logged in, and to which you would type interactive commands. What the system thinks is your terminal is really another computer, usually a microcomputer, running its own copy of Kermit.

When Kermit is in "local mode", file transfer is done over an external device, such as a microcomputer’s serial communication port, or an assigned terminal line on a mainframe. The local Kermit is connected in some way (like a dialout mechanism) to another computer, again running its own copy of Kermit. A local Kermit is in control of the screen, a remote Kermit has no direct access to it. Microcomputer Kermits are run in local mode unless instructed otherwise; mainframe Kermits run remotely unless some special command places them in local mode. Some commands make sense only for remote Kermits, others only for local, still others can be used with either. Local and remote operation of Kermit is shown schematically here: The Kermit program on the PC is a *local* Kermit. It can

PC is Local, Mainframe is Remote:

```
           Communication
           Line                    (Packets)
           +---------------/  /-----------------+  Other terminals
           |                                    |  |  |  |
           |                                    |  |  |  |
 PC        |     LOCAL              Mainframe   |  |  |  |  REMOTE
+----------+----------+            +------------+--+--+--+--------+
|   Serial Port        |            |            |                 |
|                     |            |            |                 |
|                     |            |            |                 |
|  +---------------+  |            |            Your job’s        |
|  | Packets: 724  |  |            |            terminal line     |
|  | Retries:   7  |  |            |                              |
|  | File: FOO.BAR |  |            |                              |
|  +---------------+  |            |                              |
|   Screen            |            |                              |
|                     |            |                              |
+---------------+-----+            +------------------------------+
                |
                | (Commands)
                |
   +------------+---------+
    \       Keyboard      \
     +----------------------+
                You
```

Figure 4-1: Local and Remote Kermits

control the screen, the keyboard, and the port separately, thus it can update the screen with status information, watch for interrupt signals from the keyboard, and transfer packets on the communications port, all at the same time.

The Kermit program running on the mainframe is a *remote* Kermit. The user logs in to the mainframe through a terminal port. The host computer cannot tell that the user is really coming in through a microcomputer. The keyboard, screen, and port functions are all combined in user's mainframe terminal line. Therefore a remote Kermit is cut off from your screen and keyboard during file transfer.

4.2. The Command Dialog

Most Kermit programs communicate with you through interactive keyword-style command dialog (a prominent exception is Macintosh Kermit, which uses pulldown menus that overlay the terminal emulation screen). The program issues a *prompt*, indicating that it is waiting for you to type a command. The prompt is usually of the form

Kermit-*xx*>

where *xx* indicates the version of Kermit -- `Kermit-MS>` for MS-DOS Kermit, `Kermit-86>` for CP/M-86 Kermit, etc.

In response to the program's prompt you may type a keyword, such as SEND, RECEIVE, or EXIT, possibly followed by additional keywords or operands, each of which is called a *field*. You can abbreviate keywords (but not file names) to any length that makes them distinguishable from any other keyword valid for that field. You can type a question mark at any time to get information about what's expected or valid at that point. The ESC and "?" features work best on full duplex systems, where the program can "wake up" immediately and perform the required function. On half duplex or record-oriented systems, the ESC feature is not available, and the "?" requires a carriage return to follow.

In this example, the user types "set" and then a question mark to find out what the SET options are. The user then continues the command at the point where the question mark was typed, adding a "d" and another question mark to see what set options start with "d". The user then adds a "u" to select "duplex" (the only SET option that starts with "du") followed by an ESC (shown here by a dollar sign) to complete the current field, then another question mark to see what the possibilities are for the next field, and so forth. The command is finally terminated by a carriage return. Before carriage return is typed, however, the command can be edited or erased using RUBOUT or other command editing keys. Finally, the same command is entered again with a minimum of keystrokes, with each field abbreviated to its shortest unique length. In the example, the parts the user types are underlined; all the rest is system typeout:

```
Kermit-20>set ? one of the following:
 debugging        delay            duplex           escape
 file             handshake        IBM              line
 parity           receive          send
Kermit-20>set d? one of the following:
 debugging   delay       duplex
Kermit-20>set du$plex (to) ? one of the following:
 full   half
Kermit-20>set duplex (to) h$alf
Kermit-20>set du h
```

4.3. Notation

In the command descriptions, the following notation is used:

italics	A parameter - the symbol in italics is replaced by an argument of the specified type (number, filename, etc).
[*anything*]	A field enclosed in square brackets is optional. If omitted, the field defaults to an appropriate value. You don't type the brackets.
{x,y,z}	A list of alternatives is enclosed in curly braces; you type one of the alternatives.
number	A whole number, entered in prevailing notation of the system.

character A single character, entered literally, or as a number (perhaps octal or hexadecimal) representing the ASCII value of the character.

floating-point-number
A "real" number, possibly containing a decimal point and a fractional part.

filespec A file specification, i.e. the name of a file, possibly including a search path, device or directory name, or other qualifying information, and possibly containing "wildcard" or pattern-matching characters to denote a group of files.

^X A control character may be written using "uparrow" or "caret" notation, since many systems display control characters this way. Control characters are produced by holding down the key marked CTRL or Control and typing the appropriate character, e.g. X. Control characters may also be written Ctrl-X, CTRL-X, CTRL/X, etc.

Commands are shown in upper case, but can be entered in any combination of upper and lower case.

4.4. Summary of Kermit Commands

Here is a brief list of Kermit commands as they are to be found in most Kermit programs. The following sections will describe these commands in detail.

For exchanging files:
SEND, RECEIVE, GET

For connecting to a remote host:
CONNECT, SET LINE, SET PARITY, SET DUPLEX, SET HANDSHAKE, SET ESCAPE, SET FLOW-CONTROL, SET SPEED (or BAUD)

For acting as a server:
SERVER

For talking to a server:
BYE, FINISH, GET, SEND, REMOTE

Setting nonstandard transmission and file parameters:
SET BLOCK-CHECK, SET DEBUG, SET DELAY, SET FILE, SET INCOMPLETE, SET PARITY, SET RETRY;
SET SEND (or RECEIVE) END-OF-LINE, START-OF-PACKET, PACKET-LENGTH, PAUSE, TIMEOUT, PADDING

For defining and executing "macros" of commands:
DEFINE, DO

For interrupting transmission:
Control-X, Control-Z, Control-C, Control-E

Getting information:
HELP, STATISTICS, SHOW

Executing command files:
TAKE

For recording the history of a file transfer operation:
LOG TRANSACTIONS

For non-protocol file capture or transmission:
LOG SESSION, TRANSMIT, INPUT, OUTPUT, PAUSE, CLEAR, SCRIPT

For closing log files:
CLOSE

Leaving the program:
EXIT, QUIT

If you have a file called `KERMIT.INI` in your default or home disk, Kermit will execute an automatic TAKE command on it upon initial startup. `KERMIT.INI` may contain any Kermit commands, for instance SET commands, or DEFINEs for macros to configure Kermit to various systems or communications media. *Note:* Your particular implementation of Kermit may use a different name for this file, like `MSKERMIT.INI` for MS-DOS Kermit, or `VMSKERMIT.INI` for VAX/VMS Kermit.

4.5. The SEND Command

Syntax:

Sending a single file:

`SEND` *filespec1* `[`*filespec2*`]`

Sending multiple files:

`SEND` *wild-filespec1*

The SEND command causes a file or file group to be sent to the other system. There are two forms of the command, depending on whether *filespec1* contains "wildcard" characters. Use of wildcard characters is the most common method of indicating a group of files in a single file specification. For instance if `FOO.FOR` is a single file, a FORTRAN program named FOO, then `*.FOR` might be a group of FORTRAN programs.

Sending a File Group --

If *filespec1* contains wildcard characters then all matching files will be sent, in directory-listing order by name. If a file can't be opened for read access, it will be skipped.

Sending a Single File --

If *filespec1* does not contain any wildcard characters, then the single file specified by *filespec1* will be sent. Optionally, *filespec2* may be used to specify the name under which the file will arrive at the target system; *filespec2* is not parsed or validated locally in any way. If *filespec2* is not specified, the file will be sent with its own name.

SEND Command General Operation --

Files will be sent with their filename and filetype (for instance `FOO.BAR`, no device or directory field, no generation number or attributes). If communication line parity is being used (see SET PARITY), the sending Kermit will request that the other Kermit accept a special kind of prefix notation for binary files. This is an advanced feature, and not all Kermits have it; if the other Kermit does not agree to use this feature, binary files cannot be sent correctly.

The sending Kermit will also ask the other Kermit whether it can handle a special prefix encoding for repeated characters. If it can, then files with long strings of repeated characters will be transmitted very efficiently. Columnar data, highly indented text, and binary files are the major beneficiaries of this technique.

SEND Remote Operation --

If you are running Kermit remotely (for instance, from a microcomputer), you should "escape back" to your local Kermit within a reasonable amount of time and give the RECEIVE command. Don't take more than a minute or two to complete the switch, or Kermit may "time out" and give up (in that case, you'll have to CONNECT back to the remote system and reissue the SEND command).

SEND Local Operation --

If you're running Kermit locally, for instance on a microcomputer, you should have already run Kermit on the remote system and issued either a RECEIVE or a SERVER command.

Once you give Kermit the SEND command, the name of each file will be printed on your screen as the transfer begins, and information will be displayed to indicate the packet traffic. When the specified operation is complete, the program will sound a beep, and the status of the operation will be indicated by a message like OK, Complete, Interrupted, or Failed.

If you see many packet retry indications, you are probably suffering from a noisy connection. You may be able to cut down on the retransmissions by using SET SEND PACKET-LENGTH to decrease the packet length; this will reduce the probability that a given packet will be corrupted by noise, and reduce the time required to retransmit a corrupted packet.

If you notice a file being sent which you do not really want to send, you may cancel the operation immediately by typing either Control-X or Control-Z. If your are sending a file group, Control-X will cause the current file to be skipped, and Kermit will go on to the next file, whereas Control-Z will cancel sending the entire group and return you to Kermit-20 command level.

4.6. The RECEIVE Command

Syntax: `RECEIVE [`*filespec*`]`

The RECEIVE command tells Kermit to wait for the arrival a file or file group sent by a SEND command from the other system. If only one file is being received, you may include the optional *filespec* as the name to store the incoming file under; otherwise, the name is taken from the incoming file header. If the name in the header is not a legal file name on the local system, Kermit will attempt to transform it to a legal name.

If an incoming file has the same name as an existing file, Kermit will either overwrite the old file or else try to create a new unique name, depending on the setting of FILE WARNING.

If you have SET PARITY, then 8th-bit prefixing will be requested. If the other side cannot do this, binary files cannot be transferred correctly. The sending Kermit may also request that repeated characters be compressed.

If an incoming file does not arrive in its entirety, Kermit will normally discard it; it will not appear in your directory. You may change this behavior by using the command SET INCOMPLETE KEEP, which will cause as much of the file as arrived to be saved in your directory.

RECEIVE Remote Operation --

If your are running Kermit remotely, you should escape back to your local Kermit and give the SEND command. You should do this within about two minutes, or the protocol may time out and give up; if this happens, you can CONNECT back to the remote system and reissue the RECEIVE command.

RECEIVE Local Operation --

If you are running Kermit locally, you should already have issued a SEND command to the remote Kermit, and then escaped back to DEC-20 Kermit (you can not issue a RECEIVE command to a Kermit server, you must use the GET command for that).

As files arrive, their names will be shown on your screen, along with a continuous display the packet traffic.

If a file begins to arrives that you don't really want, you can attempt to cancel it by typing Control-X; this sends a cancellation request to the remote Kermit. If the remote Kermit understands this request (not all implementations of Kermit support this feature), it will comply; otherwise it will continue to send. If a file group is being sent, you can request the entire group be cancelled by typing Control-Z.

4.7. The GET Command

Syntax: `GET [`*remote-filespec*`]`

The GET command requests a Kermit *server* to send the file or file group specified by *remote-filespec*. Note the distinction between the RECEIVE and GET commands: RECEIVE instructs the program to wait passively, whereas GET actively sends a request to a server.

The remote filespec is any character sequence that can be a legal file specification for the remote system; it is not parsed or validated locally. As files arrive, their names will be displayed on your screen, along with a continuous indication of the packet traffic. As in the RECEIVE command, you may type Control-X to request that the current incoming file be cancelled, Control-Z to request that the entire incoming batch be cancelled.

Optional Syntax: If you are requesting a single file, you may type the GET command without a filespec. In that case, Kermit programs that implement the optional GET syntax will prompt you for the remote filespec on the subsequent line, and the name to store it under when it arrives on the line after that:

```
Kermit-MS>get
 Remote Source File: aux.txt
 Local Destination File: auxfile.txt
```

4.8. The SERVER Command

Syntax: `SERVER`

The SERVER command instructs Kermit to cease taking commands from the keyboard and to receive all further instructions in the form of Kermit packets from another system. The other Kermit must have commands for communicating with remote servers; these include GET, SEND, FINISH, and BYE.

After issuing this command, return to the "client" system and issue SEND, GET, BYE, FINISH, or other server-directed commands from there. If your local Kermit does not have a BYE command, then it does not have the full ability to communicate with a Kermit server and you should not put the remote Kermit into SERVER mode. If your local Kermit does have a BYE command, use it to shut down and log out the Kermit server when you are done with it.

Any nonstandard parameters should be selected with SET commands before putting Kermit in server mode.

4.9. The BYE Command

Syntax: `BYE`

When running talking to a Kermit server, use the BYE command to shut down the server and, if the server is on a timesharing system, also log out the job. This will also close any open log files and exit from the local Kermit.

4.10. The FINISH Command

Syntax: `FINISH`

When running as a local Kermit talking to a remote Kermit server use the FINISH command to shut down the server without logging out the remote job, so that you can CONNECT back to it.

4.11. The REMOTE Command

Syntax: `REMOTE` *command*

When talking to a remote Kermit server, use the REMOTE command to request special functions of the remote server. If the server does not understand the command or offer the requested service (all of these commands and services are optional features of the Kermit protocol), it will reply with a message like "Unknown Kermit server command". If it does understand, it will send the results back, and they will be displayed on the screen. The REMOTE commands include:

REMOTE CWD [*directory*]
: Change Working Directory. If no directory name is provided, the server will change to the default directory. Otherwise, you will be prompted for a password, and the server will attempt to change to the specified directory. If access is not granted, the server will provide a message to that effect. If the remote system does not require a password for changing directories (UNIX is an example), then you can simply type a carriage return in response to the password prompt.

REMOTE DELETE *filespec*
: Delete the specified file or files. The names of the files that are deleted should be displayed on your screen.

REMOTE DIRECTORY [*filespec*]
: The names of the files that match the given file specification will be displayed on your screen, possibly along with additional information about file sizes and dates. If no file specification is given, all files from the current directory will be listed.

REMOTE SPACE [*directory*]
: Information about disk usage in the current remote directory -- quota, current storage, or amount of remaining free space -- is displayed on your screen.

REMOTE HELP
: A list of available server functions is displayed.

REMOTE HOST [*command*]
: The given command is passed to the server's host command processor, and the resulting output is displayed on your screen.

REMOTE KERMIT [*command*]
: The given command, which is expressed in the server Kermit's own interactive-mode command syntax, is passed to the server for execution. This is useful for changing settings, logging, and other functions.

REMOTE TYPE *filespec*
: The contents of the specified file is displayed on your screen.

REMOTE WHO [*username*]
: List users, or a specified user, logged in on the server's system.

4.12. Local Commands

Syntax: `[LOCAL]` *command*

Execute the specified command on the local system -- on the system where Kermit to which your are typing this command is running. These commands provide some local file management capability without having to leave the Kermit program, which is particularly useful on microcomputers. On most systems, the LOCAL prefix for these commands can be omitted.

CWD [*directory*]
: "Change Working Directory" to the specified directory.

DELETE *filespec*
: Delete the specified file or files.

DIRECTORY [*filespec*]
: Provide a directory listing of the specified files.

SPACE
: Display local disk usage and/or free space.

RUN *filespec* [*operands*]
: Run the indicated program with the supplied command-line operands.

PUSH Invoke the local system command interpreter in such a way that it can return (or "pop" or "exit") back to Kermit.

Some Kermit programs may provide commands for these or other functions in the syntax of their own system, when this would cause no confusion. For instance, CP/M Kermit may use ERA in place of LOCAL DELETE.

4.13. The CONNECT Command

LOCAL ONLY -- Syntax: `CONNECT [`*terminal-designator*`]`

Establish a terminal connection to the system at the other end of the communication line. On a microcomputer, this is normally the serial port. On a mainframe, you will have to specify a terminal line number or other identifier, either in the CONNECT command itself, or in a SET LINE command. Get back to the local Kermit by typing the escape character followed by a single character "command". Several single-character commands are possible:

`C` Close the connection and return to the local Kermit.
`S` Show status of the connection.
`B` Send a BREAK signal.
`0` (zero) Send a NUL (0) character.
`F` Copy the current screen into a disk file.
`D` Drop the line, hangup the modem.
`P` Push to the local system command processor without breaking the connection.
`Q` Quit logging session transcript.
`R` Resume logging session transcript.
`?` List all the possible single-character arguments.
`^]` (or whatever you have set the escape character to be)
Typing the escape character twice sends one copy of it to the connected host.

You can use the SET ESCAPE command to define a different escape character, and SET PARITY, SET DUPLEX, SET FLOW-CONTROL, SET HANDSHAKE to establish or change those parameters.

4.14. HELP

Syntax: `The HELP Command`

Typing HELP alone prints a brief summary of Kermit and its commands, and possibly instructions for obtaining more detailed help on particular topics. Most Kermit implementations also allow the use of "?" within a command to produce a short help message.

4.15. The TAKE Command

Syntax: `TAKE` *filespec*

Execute Kermit commands from the specified file. The file may contain contain any valid Kermit commands, including other TAKE commands.

4.16. The EXIT and QUIT Commands

Exit from Kermit. `QUIT` is a synonym for EXIT.

4.17. The SET Command

Syntax: `SET` *parameter* [*option*] [*value*]

Establish or modify various parameters for file transfer or terminal connection.

When a file transfer operation begins, the two Kermits automatically exchange special initialization messages, in which each program provides the other with certain information about itself. This information includes the maximum packet size it wants to receive, the timeout interval it wants the other Kermit to use, the number and type of padding characters it needs, the end-of-line character it needs to terminate each packet (if any), the block check type, the desired prefixes for control characters, characters with the "high bit" set, and repeated characters. Each Kermit program has its own preset "default" values for these parameters, and you normally need not concern yourself with them. You can examine their values with the SHOW command; the SET command is provided to allow you to change them in order to adapt to unusual conditions.

The following parameters may be SET:

BAUD	Set the speed of the current communications port
BLOCK-CHECK	Packet transmission error detection method
DEBUG	Mode or log file
DELAY	How long to wait before starting to send
DUPLEX	For terminal connection, full (remote echo) or half (local echo)
END	Packet termination character (normally CR)
ESCAPE	Character for terminal connection
FILE	For setting file parameters like name conversion and byte size
FLOW-CONTROL	Selecting flow control method, like XON/XOFF
HANDSHAKE	For turning around half duplex communication line
IBM	Set parameters for IBM mainframe linemode connection
INCOMPLETE	What to do with an incomplete file
KEY	Establish a key redefinition or keyboard macro
LOCAL-ECHO	Specify who echoes during terminal connection
LINE	Terminal line to use for terminal connection or file transfer
MODEM	Modem type or characteristics
PARITY	Character parity to use
PORT	For switching communication ports
PROMPT	For changing the program's command prompt
RECEIVE	Various parameters for receiving files
RETRY	How many times to retry a packet before giving up
SEND	Various parameters for sending files
SPEED	Synomym for BAUD.
TERMINAL	Parameters for terminal emulation
TIMER	Enable/disable timeouts
WARNING	Filename collision protection

The DEFINE command may be used to compose "macros" by combining SET and possibly other commands. The SET commands are now described in detail.

SET BAUD

Set or change the baud rate (approximate translation: transmission speed in bits per second) on the currently selected communications device. Ten bits per second is usually equivalent to one character per second; 300 baud = 30 cps. The way of specifying the baud rate varies from system to system; in most cases, the actual number (such as 1200 or 9600) is typed. Systems that do not provide this command generally expect that the speed of the line has already been set appropriately outside of Kermit. Common values are 300, 1200, 2400, 4800, 9600, 19200.

SET BLOCK-CHECK {1, 2, 3}

Kermit normally uses a 1-character block check, or "checksum", on each packet. The sender of the packet computes the block check based on the other characters in the packet, and the receiver recomputes it the same way. If these quantities agree, the packet is accepted and transmission proceeds. If they disagree, the packet is rejected and retransmission is requsted.

However, the block check is not a foolproof method of error detection. The normal single-character Kermit block check is only a 6-bit quantity (the low order 8 bits of the arithmetic sum folded upon itself). With only six bits of accuracy, the chances are one in 2^6 -- that is, 1/64 -- that an error can occur which will not be detected in the checksum, assuming that all errors are equally likely (they aren't).

You can decrease the probability that an error can slip through, at the expense of transmission efficiency, by using the SET BLOCK-CHECK command to select more rigorous block check methods. Note that all three methods will detect any single-bit error, or any error in an odd number of bits. The options are:

1-CHARACTER-CHECKSUM:
: The normal single-character 6-bit checksum.

2-CHARACTER-CHECKSUM:
: A 2-character, 12-bit checksum. Reduces the probability of an error going undetected to 1/4096, but adds an extra character to each packet.

3-CHARACTER-CRC:
: A 3-character, 16-bit Cyclic Redundancy Check, CCITT format. In addition to errors in any odd number of bits, this method detects double bit errors, all error bursts of length 16 or less, and more than 99.99% of all possible longer bursts. Adds two extra characters to each packet.

The single character checksum has proven to be quite adequate in practice, much more effective than straightforward analysis would indicate, since all errors are *not* equally likely, and a simple checksum is well suited to catching the kinds of errors that are typical of telecommunication lines. The other methods should be requested only when the connection is *very* noisy and/or when sending binary files, or when using "long packets" (see SET RECEIVE PACKET-LENGTH).

Note that the 2- and 3-character block checks are not available in all versions of Kermit; if the other Kermit is not capable of performing the higher-precision block checks, the transfer will automatically use the standard single-character method.

SET DEBUG {ON, OFF}

Syntax: `SET DEBUG` *options*

Record debugging information, either on your terminal or in a file. Options are:

ON
: Turn on debugging.

OFF
: Don't display debugging information (this is the default). If debugging was in effect, turn it off and close any log file.

or possibly others, like STATES, PACKETS, SESSION, etc., to select logging of different phenomena. Some Kermit programs may control debugging by use of the LOG DEBUG command.

SET DELAY

Syntax: `SET DELAY` *number*

Specify how many seconds to wait before sending the first packet after a SEND command. Use when remote and SENDing files back to your local Kermit. This gives you time to "escape" back and issue a RECEIVE command. The normal delay is 5 seconds. In local mode or server mode, Kermit does not delay before sending the first packet.

SET DUPLEX

Syntax: `SET DUPLEX {FULL, HALF}`

For use when CONNECTed to a remote system. The keyword choices are FULL and HALF. FULL means the remote system echoes the characters you type, HALF means the local system echoes them. FULL is the default, and is used by most hosts. HALF is necessary when connecting to IBM mainframes. Half duplex is also called "local echo"; in some Kermits, use SET LOCAL-ECHO ON instead of SET DUPLEX HALF.

SET ESCAPE

Syntax: `SET ESCAPE` *character*

Specify or change the character you want to use to "escape" from remote connections back to Kermit. This would normally be a character you don't expect to be using on the remote system, perhaps a control character like `^\`, `^]`, `^^`, or `^_`. Most versions of Kermit use one of these by default. After you type the escape character, you must follow it by a single-character "argument", such as "C" for Close Connection. The arguments are listed above, under the description of the CONNECT command.

SET FILE

Syntax: `SET FILE` *parameter value*

Establish file-related parameters. Depending on the characteristics of the system, it may be necessary to tell Kermit how to fetch an outbound file from the disk, or how to store an incoming file. The actual parameters you can specify in this command will vary from system to system, and you should consult the documentation for your particular version of Kermit. Some examples would be file type (text or binary), byte size (PDP-10 architecture), record length or block size (record oriented systems), end-of-file detection method (on microcomputers), file naming conversion option.

This can be a very important command if you intend to transfer binary files, but is normally unecessary for transmitting textual files.

SET FLOW-CONTROL

Syntax: `SET FLOW-CONTROL {XON/XOFF,NONE}`

For communicating with full duplex systems. System-level flow control is not necessary to the Kermit protocol, but it can help to use it if the same method is available on both systems. The most common type of flow control on full duplex systems is XON/XOFF. When a system's input buffer comes close to being full, it will send an XOFF character (Control-S) to request the other system to stop sending. When it has emptied sufficient characters from its input buffer, it signals the other system to resume sending by transmitting an XON character (Control-Q). This process operates in both directions simultaneously. The options for the Kermit SET FLOW command are usually restricted to XON/XOFF and NONE, which is used to disable this feature.

SET HANDSHAKE

Syntax: `SET HANDSHAKE` *option*

For communicating with half duplex systems. This lets you specify the line turnaround character sent by the half duplex host to indicate it has ended its transmission and is granting you permission to transmit. When a handshake is set, Kermit will not send a packet until the half duplex host has sent the specified character (or a timeout has occurred). The options may include:

NONE	No handshake; undo the effect of any previous SET HANDSHAKE.
XOFF	Control-S.
XON	Control-Q.
BELL	Control-G.
CR	Carriage Return, Control-M.
LF	Linefeed, Control-J.
ESC	Escape, Control-`[`.

Some Kermit programs may require the option to be specified by typing the character literally or entering its numeric ASCII value. If you use this command to enable handshaking, you should also SET FLOW OFF.

SET INCOMPLETE

Syntax: `SET INCOMPLETE {KEEP, DISCARD}`

Specify what to do when a file transfer fails before it is completed. The options are DISCARD (the default) and KEEP. If you choose KEEP, then if a transfer fails to complete successfully, you will be able to keep the incomplete part that was received.

SET LINE

Syntax: `SET LINE [`*terminal-designator*`]`

Specify the terminal line to use for file transfer or CONNECT. This command is found on mainframe Kermits, which normally run in "remote mode" using their own controlling terminal for file transfer. Specifying a separate line puts the program in "local mode." If no line is specified, revert to the job’s controlling terminal, i.e. go back to "remote mode."

SET PORT

Syntax: `SET PORT` *terminal-designator*

Specify the communications port for file transfer or CONNECT. This command is found on microcomputer Kermits that run in "local" mode. SET PORT does not change the remote/local status but simply selects a different port for local operation.

SET PARITY

Syntax: `SET PARITY {EVEN, ODD, MARK, SPACE, NONE}`

Parity is a technique used by communications equipment for detecting errors on a per-character basis; the "8th bit" of each character acts as a check bit for the other seven bits. Kermit uses block checks to detect errors on a per-packet basis, and it does not use character parity. However, some systems that Kermit runs on, or equipment through which these systems communicate, may be using character parity. If Kermit does not know about this, arriving data will have been modified and the block check will appear to be wrong, and packets will be rejected.

If parity is being used on the communication line, you must inform both Kermits, so the desired parity can be added to outgoing characters, and stripped from incoming ones. SET PARITY should be used for communicating with hosts that require character parity (IBM mainframes are typical examples) or through devices or networks (like GTE TELENET) that add parity to characters that pass through them. Both Kermits should be set to the same parity. The specified parity is used both for terminal connection (CONNECT) and file transfer (SEND, RECEIVE, GET).

The choices for SET PARITY are:

NONE (the default) eight data bits and no parity bit.

MARK
: seven data bits with the parity bit set to one.

SPACE
: seven data bits with the parity bit set to zero.

EVEN seven data bits with the parity bit set to make the overall parity even.

ODD seven data bits with the parity bit set to make the overall parity odd.

NONE means no parity processing is done, and the 8th bit of each character can be used for data when transmitting binary files.

If you have set parity to ODD, EVEN, MARK, or SPACE, then most versions of Kermit will request that binary files be transferred using 8th-bit-prefixing. If the Kermit on the other side knows how to do 8th-bit-prefixing (this is an optional feature of the Kermit protocol, and not all implementations of Kermit have it), then binary files can be transmitted successfully. If NONE is specified, 8th-bit-prefixing will not be requested.

SET PROMPT

Syntax: `SET PROMPT` *string*

This allows you to change the program's prompt. This is particularly useful if you are using Kermit to transfer files between two systems of the same kind, in which case you can change the prompts of the Kermit programs involved to include appropriate distinguishing information.

SET SEND

`SET SEND` *parameter value*

Establish parameters for outgoing packets. This command is generally used to override negotiated values, or to establish before negotiation takes place.

END-OF-LINE *character*
: The ASCII character to be used as a line terminator for outbound packets, if one is required by the other system, carriage return by default. You will only have to use this command for systems that require a line terminator other than carriage return.

PACKET-LENGTH *number*
: Maximum packet length to send between 10 and 94 (decimal). Shortening the packets might allow more of them to get through through without error on noisy communication lines. Lengthening the packets increases the throughput on clean lines. This command can be used to specify a shorter length than the one requested by the other Kermit, but not a longer one.

TIMEOUT *number*
: How many seconds to wait for a packet before trying again. A value of zero means don't time out -- wait forever.

PAUSE *floating-point-number*
: How many seconds to pause before sending each data packet. Setting this to a nonzero value may allow a slow system enough time to consolidate itself before the next packet arrives. Normally, no per-packet pausing is done.

PADDING *number*, PADCHAR *character*
: How much padding to send before a packet, if the other side needs padding, and what character to use for padding. Defaults are no padding, and NUL (0) for the padding character. This command is also handy for inserting special characters that may be required by communications equipment.

QUOTE *character*

What printable character to use for quoting of control characters, "#" (43) by default. There should be no reason to change this.

START-OF-PACKET *character*
The start-of-packet character is the only control character used "bare" by the Kermit protocol. It is Control-A by default. If a bare Control-A causes problems for your communication hardware or software, you can use this command to select a different control character to mark the start of a packet. You must also issue the reciprocal command (SET RECEIVE START-OF-PACKET) to the Kermit on the other system (providing it has such a command).

SET RECEIVE

Syntax: `SET RECEIVE` *parameter value*

Parameters to request or expect for incoming packets, as follows:

END-OF-LINE *character*
Carriage return (15) by default.

PACKET-LENGTH *number*
Maximum length packet for the other side to send, decimal number, between 10 and 94, decimal. Some Kermit programs also support a "long packet" protocol extension for improved file transfer efficiency. If you specify a value greater than 94 (and normally less than 1000 or 2000), then the two Kermits will attempt to negotiate the use of long packets in the receiver's direction. If the negotiation is unsuccessful (e.g. because the sending Kermit does not support long packets), then ordinary packets will be used automatically.

TIMEOUT *number*
How many seconds the other Kermit should wait for a packet before asking for retransmission.

PAUSE *floating-point-number*
How many seconds to pause before acknowledging a packet. Setting this to a nonzero value will slow down the rate at which data packets arrive, which may be necessary for systems that have "sensitive" front ends and cannot accept input at a high rate.

PADDING *number*, PADCHAR *character*
How many padding characters to request before each incoming packet, and what the padding character should be. No Kermits are known to need padding, and if one did, it would request it without your having to tell it to do so. This command would only be necessary, therefore, under very unusual circumstances.

QUOTE *character*
What printable character to use for quoting of control characters, "#" (43) by default. There should be no reason to change this.

START-OF-PACKET *character*
The control character to mark the beginning of incoming packets. Normally SOH (Control-A, ASCII 1) (see SET SEND START-OF-PACKET, above).

SET RETRY

`SET RETRY` *option number*

Set the maximum number of retries allowed for:

INITIAL-CONNECTION
How many times to try establishing the initial protocol connection before giving up, normally something like 15.

PACKETS
How many times to try sending a particular packet before giving up, normally 5. If a line is very noisy, you might want to increase this number.

SET WARNING

```
SET WARNING {ON, OFF}
```

Tell Kermit whether to let incoming files overwrite existing files of the same name. If WARNING is ON, then Kermit will warn you of filename collisions and will attempt to construct a new, unique name for the arriving file, and inform you what the new name is. When OFF, Kermit silently overwrites existing files when there's a name collision. This command may also be called SET FILE WARNING.

4.18. The DEFINE Command

DEFINE *macroname* [*command* [, *command* [, ...]]]

Define a "macro" to allow convenient association of one or more Kermit commands with a mnemonic keyword of your choice. The definition consists of a list a list of one or more Kermit commands, separated by commas. If you use Kermit to communicate with several different kinds of systems, you may set up a macro for each, for instance:

```
DEFINE IBM SET PARITY MARK, SET DUPLEX HALF, SET HANDSHAKE XON
DEFINE UNIX SET PARITY NONE, SET DUPLEX FULL, SET HANDSHAKE NONE
DEFINE TELENET SET PARITY MARK, SET RECEIVE TIMEOUT 20
```

You may then type DO IBM, DO UNIX, and so forth to set all the desired parameters with a single command. It is convenient to include these definitions in your Kermit initialization file.

Another other handy use for macros would be for rapid adaptation to different conditions of line noise:

```
DEFINE CLEAN SET BLOCK-CHECK 1, SET REC PACKET-LEN 94, SET RETRY 5
DEFINE NOISY SET BLOCK 2, SET SEND PACKET 60, SET RETRY 10
DEFINE VERY-NOISY SET BLOCK 3, SET SEND PACKET 40, SET RETRY 20
```

You may redefine an existing macro in the same manner as you defined it. You can undefine an existing macro by typing an empty DEFINE command for it, for instance:

```
DEFINE IBM
```

You can list all your macros and their definitions with the SHOW MACROS command. Syntax of SET and DO commands may vary among Kermit programs.

4.19. The SHOW Command

Syntax: SHOW [*option*]

The SHOW command displays the values of the parameters settable by the SET command. If a particular option is not requested, a complete display will be provided. Type "show ?" for a list of what can be shown.

4.20. The STATISTICS Command

Give statistics about the most recent file transfer, such as the total number of characters transmitted, the effective baud rate, and so forth. On some systems, this is SHOW STATISTICS.

4.21. The LOG Command

Syntax: `LOG` [*option*] [*filespec*]

Log the specified entity to the specified log file.

TRANSACTIONS Direct Kermit to log transactions, such as files successfully sent or received or files that could not be successfully sent or received. A transaction is useful recording the progress of a long, unattended multifile transfer.

SESSION Create a transcript of a CONNECT session, when running a local Kermit connected to a remote system, in the specified file. The log is closed when connection is closed. In some implementations, logging can be "toggled" by typing the connect escape character followed by Q (Quit logging) or R (Resume logging) or similar single-character commands. Session-logging is useful for recording dialog with an interactive system, and for "capturing" from systems that don't have Kermit. No guarantee can be made that the file will arrive correctly or completely, since no error checking takes place.

DEBUGGING Record debugging information in the specified file. There may be several options to select the desired information -- entire packets, state transitions, internal program trace, etc -- available via the SET DEBUGGING command.

PACKETS Record packets, and all communication line traffice during file transfer, in the specified file.

4.22. The TRANSMIT Command

Syntax: `TRANSMIT` *filespec*

Send the contents of the specified file to the other system "bare", without protocol, packets, error checking, or retransmission. This command is useful for sending files to systems that don't have Kermit. No guarantee can be made that the target system will receive the file correctly and completely. When receiving a file, the target system would normally be running a text editor in text collection mode.

The current communication settings, such as parity, flow control, and handshake, are obeyed by most Kermit versions that have a TRANSMIT command. The action is normally line-at-a-time. Each line of the file is sent, terminated by a carriage return (no linefeed), just as you would type it. Kermit waits for the remote system to echo a linefeed (as it does when you type carriage return) before it sends the next line.

Thus TRANSMIT provides a "raw upload" capability. The opposite, "raw download", may be accomplished by using the LOG SESSION command.

4.23. Login Scripts: The INPUT, OUTPUT, CLEAR, and PAUSE Commands

When running Kermit in local mode, you may use the INPUT, OUTPUT, CLEAR, and PAUSE commands to carry on a dialog with the remote system. When combined into a "script" in a Kermit TAKE command file, these commands provide the ability to initially connect and log in to a remote system, and to set it up for file transfer. While you may do this yourself manually using the CONNECT command, the login script facility allows this often tedious task to be automated, and more important, allows it to take place unattended -- perhaps late at night when the phone rates are low and the system is faster.

The CLEAR Command

Syntax: `CLEAR`

Clear the input and output buffers of the currently selected line, and attempt to clear any XOFF deadlock.

The PAUSE Command

Syntax: `PAUSE` [*interval*]

Pause the specified number of seconds before executing the next command. The default interval is one second.

The INPUT Command

Syntax: `INPUT` [*interval*] [*string*]

On the currently selected communication line, look for the given string for the specified interval of time, which is specified in seconds. If no interval is specified, then wait for the default interval, which may be specified by SET INPUT DEFAULT-TIMEOUT, and is normally 5 seconds. Specifying an interval of 0 (or less) means no timeout -- wait forever for the specified string. An INPUT command can normally be interrupted by typing one or more Control-C's, which will return you to Kermit prompt level.

Characters coming in from the line will be scanned for the search string, and when a match is found, the command will terminate successfully; if the string is not found within the given interval, the command will terminate unsuccessfully.

The search string may contain any printable characters. Control or other special characters that you could not normally type as part of a command may be included by preceding their ASCII values with a backslash, for instance `foo\13` is "foo" followed by a carriage return (ASCII 13, decimal). (Some Kermit programs expect or allow other number bases, such as octal or hexadecimal.)

While scanning, alphabetic case is ignored ("a" = "A") unless you have SET INPUT CASE OBSERVE. If no search string is given, then the INPUT command will simply display all incoming characters on your screen until it times out or is interrupted.

If the INPUT command finds the specified string within the alloted amount of time, it terminates immediately, without an error message, and the next command will be executed. If the INPUT command fails to find the requested string, it will "fail"; failure is only significant if the command was issued from a TAKE command, and INPUT TIMEOUT-ACTION is SET to QUIT. When a timeout occurs under these conditions, the command file is immediately terminated and control is returned to the invoking level, either Kermit prompt level or a superior command file. If INPUT TIMEOUT-ACTION is SET to PROCEED, then the next command (if any) will be executed from the current command file.

In addition to otherwise untypable control characters (like Control-C), certain printable characters in the search string may need to be "quoted" using the backslash mechanism, including at-sign, question mark, or other characters that are significat to Kermit's command processor.

The OUTPUT Command

Syntax: `OUTPUT` *string*

The given string is sent out the currently selected communication line. The string is in the same form as the INPUT string; control or special characters may be included by prefacing their octal ASCII value with a backslash. Note that any terminating carriage return must be included explicitly as `\13` (decimal).

How to Use Login Scripts

Login scripts are useful on computers that have autodialers or TTY ports hardwired or otherwise connected to terminal ports on other systems. Scripts can be used to automate the task of connecting and logging in. For instance, suppose your PC is connected to a VAX Unix system through a hardwired line on communication port 2. To send a file to the VAX, you must connect to the VAX through the port, log in, run Unix Kermit, escape back to the PC, and issue the appropriate file transfer commands, then connect back to the VAX and logout. This may all be automated by means of the following "script" stored in a PC file invoked by the Kermit TAKE command:

```
set port 2
output \13
input login:
out myuserid\13
in 10 Password:
out mypassword\13
in 20 %
out kermit -r\13
send foo.bar
out \4
input
```

The first line points Kermit at the communication line. The next line sends a carriage return, which makes Unix issue a "`login:`" prompt; the following INPUT command waits for this prompt to appear. When it does, Kermit outputs "myuserid" followed by a carriage return. Unix then prompts for a password; after the prompt appears, Kermit supplies the password. Then, Kermit waits up to 20 seconds for the Unix shell's "`%`" prompt; this allows time for various system messages to be displayed. When the shell prompt appears, Kermit sends the command "kermit -r", which tells Unix Kermit to receive a file; then a SEND command is given to the local Kermit. After the file is successfully transferred, Kermit sends a logout command (Control-D, "\4") to Unix. The final INPUT command causes Kermit to display any typeout (in this case the Unix system's logout message) that occurs up to the default timeout interval.

The INPUT command is very important, because it ensures synchronization. One might expect to be able to simply send all the characters out the communication line at once, and let the remote host's typeahead and buffering facilities take care of the synchronization. In rare or simple cases, this might work, but it assumes that (a) the remote host allows typeahead, (b) the remote host's typeahead buffers are big enough to accommodate all the characters, *and* (c) that the remote host never clears pending typeahead. These conditions rarely hold. For instance, Unix clears its input buffer *after* issuing the "`Password:`" prompt; any typeahead will be lost. Interactive users as well as login script facilities must wait for the prompt before entering the password. This is the function of the INPUT command. On half duplex systems, this function is critical -- these systems cannot accept any input in advance of a prompt; there is no typeahead.

The Kermit script facility is not a programming language; there is no conditional execution of commands, no branching, no labels. Nevertheless, the SET INPUT command provides a degree of control. If the Unix system were down in the sample script above, Kermit would still proceed merrily through the entire script, sending its output into the void and waiting the entire timeout interval on each INPUT command, then attempting to send a file to a Kermit that wasn't there. It could take several minutes of timing out to terminate the script. This could be avoided by including the command

```
SET INPUT TIMEOUT-ACTION QUIT
```

at the top of the script. When the "`login:`" prompt failed to appear within the timeout interval, the rest of the script would be cancelled.

See the chapters on MS-DOS and DEC-20 Kermit for further examples of login scripts.

5. MS-DOS KERMIT

Program:	Joe R. Doupnik (Utah State University), with contributions by James Harvey (Indiana/Purdue University), James Sturdevant (A.C. Nielson Company), and many others. Originally by Daphne Tzoar and Jeff Damens (Columbia University). See History.
Language:	Microsoft Macro Assembler (MASM)
Version:	2.31
Released:	July 1, 1988.
Documentation:	Christine Gianone, Frank da Cruz (Columbia University), Joe R. Doupnik (Utah State University)
Dedicated To:	Peppi

Kermit-MS Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
File transfer interruption:	Yes
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count compression:	Yes
Alternate block check types:	Yes
Terminal emulation:	VT102, H19, VT52, Tektronix 4010
Communication settings:	Speed, Parity, Flow Control, Echo
Transmit BREAK:	Yes (and Long BREAK)
IBM mainframe communication:	Yes
Transaction logging:	Yes
Session logging (raw download):	Yes
Raw upload:	Yes
Act as server:	Yes
Talk to server:	Yes
Advanced server functions:	Yes
Advanced commands for servers:	Yes
Local file management:	Yes
Command/init files:	Yes
Command macros:	Yes
Extended-length packets:	Yes
Local area networks:	Yes (NetBIOS and other support)
MS-Windows compatibility:	Yes
Attribute packets:	Yes
Sliding windows:	No

MS-DOS Kermit, or "Kermit-MS" (or MS-Kermit), is a program that implements the Kermit file transfer protocol for the entire IBM PC family, including the PS/2 series, IBM compatibles, and several other machines based on the Intel 8086 processor series (8088, 80286, 80386, etc) and the DOS operating system family (PC-DOS or MS-DOS, henceforth referred to collectively as MS-DOS or simply DOS).

It is assumed you are acquainted with your PC and with DOS, and that you are familiar with the general ideas of data communication and Kermit file transfer. A very brief overview is given here, but for details consult the early chapters of the *Kermit User Guide* (of which this document is a chapter), or the book Kermit, A File Transfer Protocol, by Frank da Cruz, Digital Press (1987), order number EY-6705E-DP (phone 1-800-343-8321), which also includes background tutorials on computers, file systems, and data communication (including modems, cabling, etc).

For further information about Kermit documentation, updates, lists of current available versions, and ordering information, write to:

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

5.1. System Requirements

Kermit-MS version 2.31 runs in as little as 100K of memory, but will occupy up to 148K if it can be found for extra screen rollback memory. Versions not using screen rollback memory will not require the additional space. It will also try to leave 24 Kbytes free for a second copy of `COMMAND.COM` which is needed for execution of certain commands.

On the IBM PC family, Kermit-MS `2.31` performs almost complete emulation of the DEC VT-102 and Heath/Zenith-19 terminals at speeds up to 19,200 baud or greater, lacking only the VT102's smooth scrolling and (on some display boards) 132 column features. And as of version 2.30, Kermit-MS also performs Tektronix 4010 graphics terminal emulation on IBM PC family systems equipped with CGA, EGA, or other graphics adapters, with either color or monochrome monitors.

Much of Kermit's speed is accomplished by direct writes to screen memory, but this is done in a "TopView-aware" manner to allow successful operation in windowing environments like MS-Windows, DesqView, and TopView itself. Speed is also due to direct access of the serial port 8250 UART (Universal Asynchronous Receiver/Transmitter) chip, with buffered, interrupt-driven receipt of characters and selectable XON/XOFF flow control. Full speed 9600 baud operation is possible on 4.77Mhz systems without flow control, but flow control is required on these systems for 19,200 baud or higher rates. The IBM PC version should also run on near-clones like the DG/1 that differ from true PCs only in their choice of UART; non-8250 UARTs are detected automatically, and slower non-interrupt driven Bios serial port i/o is used, in which case the top speed is in the 1200 baud range.

Kermit-MS `2.31` runs on the entire IBM PC family (the PC, XT, AT, PCjr, Portable PC, PC Convertible, PS/2) and compatibles (Compaq, VAXmate, Z150, etc), and there are also specially tailored versions for non-IBM-compatibles like the DEC Rainbow, HP-110, HP-150, HP Portable Plus, Grid Compass II, and others, plus a "generic DOS" version that should run (slowly) on any 8086-based MS-DOS machine. This document concentrates on the IBM version; some of the system-dependent capabilities described here may be lacking in the non-IBM versions. See section 5.10 for features of different systems.

`KERMIT.EXE` for the IBM PC family occupies about 99K of disk storage (the figure will vary for other versions). This can be reduced by about 15K if you run it through EXEPACK. MS-Kermit is not distributed in packed form, because problems have been reported on certain systems when this is done. So if you decide to pack it, make sure to keep an unpacked version available to fall back to in case of problems.

5.2. History

Over the years, MS-Kermit has grown from a Kermit file transfer program that embodied a simple terminal emulator into a complex and powerful communication program that includes the Kermit file transfer protocol. As a result, the bulk of this manual is devoted to the communication features, rather than Kermit protocol operation. Skip ahead to the next section if you're not interested in the history of MS-Kermit.

MS-DOS Kermit (like the Kermit file transfer protocol itself) is a product of the Systems Group of the Columbia University Center for Computing Activities, and it was one of the four original Kermit programs (with the CP/M, DEC-20, and IBM mainframe versions). It was initially written for the IBM PC with DOS 1.1 by Daphne Tzoar in 1981-1982, based largely on Bill Catchings's original CP/M 8080 assembler version. PC-Kermit (as it was called then) provided basic Kermit file transfer and VT52 emulation. Joellen Windsor of the University of Arizona added conditional assembly support for the Heath/Zenith-100 shortly thereafter, and soon after that Dave King of Carnegie-Mellon University added Heath-19 terminal emulation, and some patches to let the program run under the new DOS version, 2.0. During this era, the program version numbers went from 1.0 to 1.20.

With the appearance in the marketplace of many new MS-DOS machines that were not compatible with the IBM PC, it became apparent that conditionally assembled code supporting each of these machines within a single monolithic source file was not the best way to organize the program. Therefore Daphne, along with Jeff Damens of Columbia, undertook to reorganize the program in a modular way, isolating system dependencies into separate files. The result was version 2.26, released in July 1984. It included support for the DEC Rainbow, the HP-150, the Wang PC, and generic MS-DOS, as well as for the IBM PC family and the H/Z-100. It also included many new features, like 8th-bit prefixing (code contributed by The Source Telecomputing), alternate block check selection, byte-count compression, server/client operation, access to local file and DOS operations, command macros, initialization and command files, screen rollback, key redefinition, and more. For the 2.26 release, the executable Kermit programs were encoded printably as ‘‘`.BOO`’’ files, designed by Bill Catchings as part of this effort, for network and electronic-mail distribution.

Release 2.27 was produced by Daphne and Jeff in December 1984. Unlike 2.26, it ran correctly on the new PC/AT under DOS 3.0, and included support for the NEC APC from Ron Blanford of Seattle, WA, and Ian Gibbons of the University of Hawaii, and for the TI Professional from Joe Smith of the Colorado School of Mines, plus some bug fixes and reorganization. 2.27 is the last version that runs under pre-2.0 versions of DOS.

Version 2.28 (Daphne, Jeff, June 1985) added dynamic memory allocation to reduce disk storage for the `.EXE` file, and to allow the program to adjust itself to the PC’s memory size, plus the inevitable bug fixes (many of them contributed by Edgar Butt of the University of Maryland and Gregg Small of the University of California at Berkeley). During this period, support for additional MS-DOS systems was added by various people.

In December 1985, a tape showed up at Columbia sent by Prof. Joe R. Doupnik of the Center for Atmospheric and Space Studies and EE Department at Utah State University. This tape contained version 2.28 modified to fully support the DOS 2.0 file system, and to which many new features had been added, notably the ability of the MS-DOS Kermit server to process various REMOTE commands (DIR, CWD, SPACE, etc). And at about the same time, a tape arrived from James Harvey of Indiana/Purdue University, who had changed Kermit’s CONNECT command to emulate the popular DEC VT100 terminal. James’s material was sent to Joe, who then laboriously fitted the VT100 emulation into his own code, keeping the VT52 and H19 emulation alive as options, and upgrading the VT100 emulation to VT102 by adding features such as line and character insertion and deletion. The result was version 2.29, released in May 1986.

Soon after the release of 2.29, some disks were sent in by James Sturdevant of the A.C. Nielson Company, containing a full implementation of the Kermit script facility, as described in the Kermit book. This material was sent to Joe, who had by now become keeper of MS-DOS Kermit and had already begun work on version 2.30 by adding support for extended-length packets. Joe had been carrying on voluminous network correspondence (Thanks, BITNET!) with Columbia and with MS-DOS Kermit users and testers all over the world, giving birth to many new features, including Tektronix graphics terminal emulation, support for operation over local area networks, support for 8-bit ASCII terminal connections and international character sets, ANSI printer control, and a redesigned, more powerful, more portable key redefinition mechanism.

Version 2.30 was formally released on January 1, 1988, after many "alpha" and "beta" tests. Among the many contributors to this version were Brian Holley and Joe Smith for the Tektronix emulation, Robert Goeke for the NEC AP3 support, Brian Peterson and Andreas Stumpf for the Victor 9000, Bob Babcock and Joe White for the Sanyos, Christopher Lent for the Wang PC, Jack Bryans for an Intel iRMX version, Jim Noble for the Grid Compass, Geoff Mulligan and others for the Zenith 100, and David Knoell for the special Rainbow edition. And thanks to Gisbert Selke, Jack Bryans, and others for proofreading drafts of this manual. And apologies to anyone we neglected to mention.

Work on version 2.31 began within weeks of the release of 2.30. The emphasis in this new version is on an improved command interface, more sophisticated script commands, and inclusion of file attributes packets to send the time, date and size of files along with the data. This version can operate with input redirected to a file or pipe, as in

```
kermit < todo.lst > todo.log
```

or

```
sort < todo.lst | kermit
```

Support for Ungermann-Bass Net One LAN is new, thanks to contributions from Henrik Levkowetz and Renne Rehmann. These changes led to a fairly thorough revision of the interior while providing the familiar commands and new features.

In contrast to parts of the commercial marketplace, Kermit-MS version numbers grow by small digits, more like the serial numbers they really are. 2.31 is the follow-on to version 2.30, regardless of level of improvement. Like all Kermit programs, MS-DOS Kermit may be freely copied and shared, so long as this is not done for profit.

5.3. Using MS-Kermit

MS-DOS Kermit performs two major functions, terminal emulation and file transfer. File transfer can be done using either the Kermit file transfer protocol, or else (without error checking), ASCII or XON/XOFF capture and transmission methods. To use Kermit for "raw" uploading or downloading of files, see the descriptions of the TRANSMIT and LOG SESSION commands.

Before you can transfer files with another system using Kermit protocol, you must first connect to it as a terminal, login if necessary, and start up a Kermit program there. Kermit's CONNECT command lets you do this by making your PC act like a terminal. After setting things up on the other computer, you must return to the PC and tell it what to do. Returning to the PC is accomplished by typing a special sequence of characters, called the "escape sequence."

The following example shows this process; the other computer is a Unix system, but the method is the same with most others. The parts you type are underlined (if this document was printed on a printer that can underline), and when you type a command, you terminate it with a carriage return, which you can't see in the example. The mysterious ``^]c'' is MS-Kermit's escape sequence, which you enter by holding down the Control (Ctrl) key and pressing ``]'' (right square bracket), and then typing the letter C. The example assumes the MS-Kermit program is stored on disk as `KERMIT.EXE`.

Program Dialog:	*Explanation:*
`A>kermit`	
`IBM PC Kermit-MS V2.31 17 July 1988`	*Program's greeting.*
`Type ? or HELP for help`	
`Kermit-MS>set speed 1200`	*Set the right baud rate.*
`Kermit-MS>connect`	*Connect as a terminal.*
`ATDT7654321`	*Dial the modem if necessary.*
`CONNECT 1200`	*The modem tells you you're connected.*

Now you're talking to the Unix system.
Type a carriage return to get its attention.

`Login: christin`	*Login to the host.*
`password:______`	*(Passwords normally don't echo.)*
`% kermit`	*Run Kermit on the host.*
`C-Kermit>receive`	*Tell it to receive a file.*
`^]c`	*Escape back to the PC.*
`Kermit-MS>send autoexec.bat`	*Send a file.*

(The file is transferred . . .)

```
Kermit-MS>                               Transfer complete, prompt reappears.
```

In this example, the user types "kermit", and sees the program's herald and its prompt, ``Kermit-MS>''. Then she sets the appropriate communication speed ("baud rate"), connects as a terminal, issues a dialing command to a Hayes-like modem (you would skip this step if you had a direct connection), logs in to her ID on the Unix system which she has dialed, starts "C-Kermit" on the Unix system, tells it to receive a file, escapes back to the PC, and tells MS-Kermit to send a file. After the file is transferred, the user would normally connect back to the Unix system, exit from the Kermit program there, and log out:

```
Kermit-MS>connect               Connect again.
C-Kermit>exit
% ^D                            Logout from Unix by typing Ctrl-D.
^]c                             Escape back to the PC.
Kermit-MS>exit                  Return to DOS.
```

To transfer a file in the other direction, simply exchange the "send" and "receive" commands above. That's the easiest and quickest way to use Kermit. If this simple scenario does not work for you, issue the MS-Kermit SHOW COMMUNICATIONS command and look for any obvious incorrect settings (port, speed, parity), fix them with SET commands (described in Section 5.6.10), and try again. (IBM mainframe linemode connections have so many "different" settings, there's a special command to do them all at once, "do ibm", which you would type as the first Kermit-MS command above.) If that doesn't help, read on. Many problems can crop up when you attempt to connect two unlike systems over a possibly hostile communication medium. And if you intend to be a frequent user of Kermit, there are many options you can take advantage of to adapt MS-Kermit to different systems, improve its performance, and automate common tasks.

5.4. The MS-DOS File System

The features of the MS-DOS file system of greatest interest to Kermit users are the form of the file specifications, and the formats of the files themselves.

5.4.1. File Specifications

MS-DOS file specifications (in version 2.0 or later of DOS) are of the form

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

where the DEVICE is 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. Some DOS implementations (like Wang) may use slash (``/'') rather than backslash as a directory separator.

Pathname is normally omitted, but can be specified in all Kermit-MS commands (as of version 2.29). 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 DOS PATH environment variable, depending on the context in which the file name appears.

> When this document says that a file is searched for "in the current path," it means that Kermit-MS looks on the current disk and directory first, and if the file is not found, then the directories listed in the PATH environment variable are searched. If the PATH environment variable is empty, Kermit looks only at the current disk and directory.

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

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-DOS systems. 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.

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 and .COM are the normal suffixes for executable programs.

MS-DOS 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 whose types are exactly one character long, or have 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.

Users of Kermit-MS should bear in mind that other (non-MS-DOS) systems may use different wildcard characters. For instance VMS and the DEC-20 use ‘‘%’’ instead of ‘‘?’’ as the single character wildcard; when using Kermit-MS to request a wildcard file group from a Kermit-20 server, the DOS ‘‘?’’ must be replaced by the DEC-20 ‘‘%’’.

5.4.2. File Formats

MS-DOS 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, so Kermit-MS has no need for a SET FILE TYPE BINARY command. But since a non-MS-DOS receiving system might need to make distinctions as to file type, you will probably have to issue SET FILE TYPE commands there if you are sending it non-text files. In transmitting files between Kermit-MS programs, regardless of file contents, the receiving MS-DOS system is equally capable of processing text, code, and data, and in fact requires no knowledge of how the bytes in the file are to be used.

MS-DOS (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 programs continue to use the CP/M convention of terminating a text file with a Control-Z character, and won’t operate correctly unless this terminating byte is present. Therefore, you should be aware of a special SET EOF option for both incoming and outbound files, described later.

Non-MS-DOS systems may be confused by nonstandard ASCII files sent by Kermit-MS:

- 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 Word Star may contain special

binary formatting codes, and could need conversion to conventional 7-bit ASCII format prior to transmission, using an "export" procedure.

- Files created by word processors that store formatting data at the end of the file, after the Control-Z and before physical end, may require special processing via SET EOF to strip the formatting data, lest they confuse non-MS-DOS recipients.
- Spreadsheet or database files usually need special formatting to be meaningful to non-MS-DOS recipients (though they can be transmitted between MS-DOS systems with Kermit-MS). Such programs usually come with an "export" procedure to convert their files to plain ASCII text.
- BASIC programs are normally saved in a binary "tokenized" form. Use BASIC's ‘‘`,a`’’ SAVE option to save them as regular ASCII text, as in

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

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

5.5. Program Setup and Invocation

The MS-DOS Kermit program can be run from any disk without any special installation procedure. On hard disk systems, it is convenient to store the program in one of the directories listed in your DOS PATH, and it is often desirable to customize Kermit's operation to your communications and computing environment by creating an initialization file.

Kermit-MS can be run interactively, from a batch file, or as an "external" DOS command. Commands consist of one or more fields, separated by "whitespace" -- one or more spaces or tabs.

Upon initial startup, the program executes any commands found in the file `MSKERMIT.INI` on the current disk, or in the current path. This initialization file may contain command macro definitions, communications settings for one or more ports, or any other Kermit-MS commands, and you may create it using any text editor capable of saving files in plain ASCII text format. Here is a sample:

```
comment -- MSKERMIT.INI, MS-DOS Kermit initialization file

comment -- Don't overwrite my files!
 set warning on

comment -- Define macros for the systems I use...
 define unix set local-echo off,set par non,set flow xon,set timer off
 def ibm set par odd,set loc on,set hands xon,set flo none,set tim on
 def modem set port 2, set speed 1200

comment -- Define a macro for quickly adapting to noisy connections...
 def noisy set block-check 3, set send packet-length 40, set retry 20

comment -- I always start out by connecting to my UNIX system...
 set port 1
 set speed 4800
 do unix
 connect
```

A different file may be substituted for `MSKERMIT.INI` by using "`-f` *filename*" on the DOS command line, e.g.

```
kermit -f monday.ini
```

The meanings of these commands will emerge below. For now, just note how you can use command files (and "macro definitions") to easily adapt MS-Kermit to widely differing communication environments. A more advanced initialization file is shown in section 5.9.

Interactive Operation:

To run Kermit-MS interactively, invoke the program from DOS command level by typing its name, normally "kermit" (this means the program should be stored in your path with the name `KERMIT.EXE`). When you see the program's prompt,

```
Kermit-MS>
```

you may type Kermit commands repeatedly until you are ready to exit the program, as in the following example (which assumes there's already a Kermit "server" set up on the other end):

```
A>
A>kermit

IBM PC Kermit-MS V2.31  17 July 1988
Type ? or HELP for help

Kermit-MS>set speed 19200
Kermit-MS>send foo.*
```

The files are sent.

```
Kermit-MS>get fot.*
```

The requested files are received.

```
Kermit-MS>exit
A>
```

Interactive commands are described in Section 5.6.

Command Line Invocation:

Kermit-MS may be invoked with command line arguments from DOS command level, for instance:

```
A>kermit send peter.amy
```

or

```
A>kermit set port 1, set baud 9600, connect
```

In this case, help and completion are not available (because the program that provides them won't start running until after you type the entire command line), and Kermit-MS will exit back to DOS after completing the specified command or commands. Therefore, when invoked with command line arguments, Kermit-MS will behave as if it were an external DOS command, like MODE. Note that several commands may be given on the command line, separated by commas. This can't be done interactively or from TAKE command files.

As of version 2.30, two new Kermit commands can be given on the DOS command line. First is the keyword STAY which prevents Kermit from exiting naturally when the last command has completed (unless, of course, EXIT or QUIT was among the commands). The second command is

`-F` *filename*

This means use the indicated filename as the initialization file rather than `MSKERMIT.INI`. The path will be searched for this file, if necessary. A space or tab must separate -F from the filename, and the F may be in upper or lower case. Example:

```
kermit -f tuesday.ini, set port 2, do ibm, stay
```

You can run Kermit with no initialization file at all by using the command

```
kermit -f nul
```

If `-F` is the only command line option, STAY is implied.

Redirected Input and Output

Kermit-MS also can be operated by redirecting input to it from a file, as in:

```
C>kermit < myscript.txt > myscript.log
```

or from a DOS "pipe", as in

```
C>sort < sends.txt | kermit
```

The file `MYSCRIPT.TXT` contains Kermit commands as if they were typed manually. The DOS symbol "<" means that Kermit should read from the following file rather from the keyboard.

Kermit knows this is occurring and takes special steps to avoid the real keyboard and to quit when the file has been completely examined. The filename can also be the name of a device, such as COM1, to converse on the same or different line as file transfer traffic. Information destined for the screen still goes to the screen unless the phrase "> *filespec*" is added to the command line above to send the normal screen output to a file or device (device COM1 also works). Note that the terminal emulation screen cannot be redirected.

Batch Operation:

Like many other MS-DOS programs, Kermit-MS may be operated under DOS batch with command line arguments. If you invoke it without command line arguments, it will run interactively, reading commands from the keyboard and not the batch file. When it exits, batch processing will continue to the end of the batch file.

Kermit-MS returns the "errorlevel" parameter used as program exit status. Present values are in the range 0 to 7 with three areas yielding success or failure reports for the entire Kermit session. The errorlevel values are:

errorlevel	Kermit session status
0	entirely successful operation
1	a Send command completed unsuccessfully
2	a Receive or GET command completed unsuccessfully
4	a REMOTE command completed unsuccessfully
3,5,6,7	combinations (addition) of the above conditions

Note that failures are remembered for the whole session and are not canceled by a following successful operation of the same type. Thus, sending several files individually yields an errorlevel of 0 only if all the files were sent successfully. The "errorlevel" parameter also applies to script commands where OUTPUT corresponds to SEND and INPUT to RECEIVE. An example of Batch invocation of Kermit is shown in Figure 5-4.

New to version 2.31 are the commands SET ERRORLEVEL number, to force a result code, and IF ERRORLEVEL, to test the value within scripts.

Remote Operation:

The MS-DOS CTTY command allows an MS-DOS system to be used from a terminal connected to its communication port. Such sessions must be conducted with great care, since many programs assume that they are running on the real console, and explicitly reference screen memory or keyboard scan codes. Kermit can be used in this manner too, but before you give it any file transfer commands, you must inform it that it is running in "remote mode" rather than its normal "local mode." Use the SET REMOTE ON command for this purpose, to prevent the file transfer display from being sent out the port.

RAM Disk Operation:

If you invoke Kermit frequently, and you have sufficient memory on your PC, you may find it convenient to copy Kermit and its initialization file to a RAM disk when you start your system. This allows Kermit to be started and used quickly and silently, with no mechanical disk operations.

For instance, if you're using IBM's VDISK facility to create the RAM disk, you might put statements like this in your `CONFIG.SYS` file:

```
DEVICE=VDISK.SYS 384 512 128 /e
```

This assumes you have 384K of extended (`/e`) memory installed and `VDISK.SYS` is in the root directory of the boot disk. It creates a 384K RAM disk with 512B sector size and space for 128 directories in the extended memory, assigning it the disk letter of your first unused disk. And then in your `AUTOEXEC.BAT` file (assuming the RAM disk is disk `D:`) . . .

```
COPY KERMIT.EXE   D: >NUL
COPY MSKERMIT.INI D: >NUL
COPY COMMAND.COM  D: >NUL
SET COMSPEC=D:\COMMAND.COM
PATH D:\; ...
```

The PATH command allows DOS to find `KERMIT.EXE`, and Kermit to find `MSKERMIT.INI` and `COMMAND.COM`, on the RAM disk. If you use Kermit transfer files to your RAM disk, remember to copy those files to a real disk before you turn off the system.

Use of MS-Kermit in Windowing Environments:

Kermit-MS can operate within windowing environments like such as TopView, DESqview, and MS-Windows. It runs in an active window under MS-Windows, accepts cut and paste material, talks with mice, and shrinks to an icon (a boxed "KER"). An MS-Windows `.PIF` file can be constructed for Kermit using the PIFEDIT program, supplied with Windows. Memory requirements should be listed as 100 to 148KB. It should state that Kermit does not modify the screen, keyboard, memory, COM1, or COM2 (not true but it satisfies Windows). Program switch and exchange should be marked as Text, and Close Window on Exit should be checked. This configuration will let you run Kermit with all the Windows features, but slowly. To run at full speed under Windows, tell PIFEDIT that Kermit modifies the screen. Then you lose the Windows features (cutting, pasting, running the clock at the same time, etc), but you still get back to the Windows interface when you EXIT Kermit.

Local Area Network Operation:

MS-Kermit 2.31 is capable of using a serial port on another local area network (LAN) node, so long as that node is running an asynchronous communication server and you have installed a device driver on your own PC that makes COM1 or other communication port i/o use the network server. This type of connection works because MS-Kermit 2.30 and later releases on IBM PCs check the selected port to see if it's a real 8250 UART chip, and if it isn't, Kermit uses only Bios calls for port i/o, and the network routes these through your network device driver. It may be desirable to give the command SET PORT BIOS*n* (*n* is a digit 1-4) to actively select the Bios port rather than a real hardware device. This style of operation should be transparent to Kermit, but not all asynchronous communications servers utilize this technique.

As of version 2.30, the IBM PC version of Kermit can also communicate directly with another PC on a local area network through the IBM NetBIOS emulator distributed with the LAN. In essence, the LAN substitutes for the serial port, modem, and other wiring. Kermit running on one user machine can transfer files with another Kermit also on the network much as if they were connected by modems, and Kermit can talk with some larger machines the same way. The important network command is

`SET PORT NETBIOS` *nodename*

for NetBios, or

`SET PORT UB-NET1` *nodename*

for Ungermann-Bass Net-One NETCI. For details, see the description of the SET PORT and SERVER commands, and (if you're interested) Section 5.17.1 for a technical description.

Kermit can even communicate with some other computers, such as Unix systems, which accept logins via this remote pathway. The initial startup is the same as calling a mainframe and logging in except the command SET PORT NET *nodename* is used instead of SET PORT COM1. A connection is established with the first use of the communications circuit, such as CONNECT, REMOTE DIR, SEND, or other file transfer command, and terminated with the HANGUP command.

5.6. Kermit-MS Commands

MS-DOS Kermit supplies most of the commands and features of "ideal" Kermit. Here's a summary:

-F	specify alternate init file name on DOS command line.
BYE	to remote server, exit from MS-Kermit.
CLEAR	serial port buffer.
CLOSE	log files and stop logging remote session.
COMMENT	For including comments in command files.
CONNECT	as terminal to remote system (C).
CWD or CD	change local working directory.
DEFINE	a macro of Kermit-MS commands.
DELETE	local files.
DIRECTORY	listing of local files.
DISABLE	server recognition of selected commands.
DO	a command macro.
ECHO	a line of text on the screen.
ENABLE	server recognition of selected commands.
EXIT	from Kermit-MS.
FINISH	Shut down a remote Kermit server.
GET	remote files from server.
GOTO	jump to labeled line in script file.
HANGUP	the phone or network connection.
HELP	about Kermit-MS.
IF	decision-making in Take or Macro scripts.
INPUT	specified string from serial port, for scripts.
LOG	remote terminal session, transactions, or packets.
LOGOUT	remote server, don't exit from Kermit-MS.
MAIL	send file to remote Mailer via Kermit.
OUTPUT	string out serial port, for scripts.
PAUSE	between commands.
POP	exit Take file or Macro.
PUSH	to MS-DOS command level.
QUIT	from Kermit-MS (same as EXIT).
RECEIVE	files from remote Kermit (R).
REINPUT	reread script Input buffer.
REMOTE	Prefix for remote file management commands.
RUN	an MS-DOS program or command.
SEND	files to remote Kermit (S).
SERVER	mode of remote operation.
SET	various parameters.
SHOW	various parameters.
SPACE	inquiry (about disk space).
STATUS	inquiry (about settings).
STAY	stay within Kermit after DOS command line invocation.
STOP	exit all Take files or Macros.
TAKE	commands from a file.
TRANSMIT	a file "raw" (no error checking).
TYPE	a local file on the screen.
VERSION	display Kermit-MS program version number.
WAIT	for the specified modem signal to appear.

Not all of these commands are necessarily available on all MS-DOS systems, and some of the commands may work somewhat differently between DOS versions.

A command keyword, such as SEND, RECEIVE, HELP, etc, may be abbreviated, so long as you have typed enough letters to distinguish it from other keywords that are valid in that position. For instance, you can type CLE for

CLEAR and CLO for CLOSE. Several common commands also have special non-unique abbreviations, like C for CONNECT, S for SEND, and R for RECEIVE. Kermit will notify you if you have typed a word with too few letters.

During interactive operation, you may edit the command you're currently typing using BACKSPACE to erase the character most recently typed, Ctrl-W to delete the most recent field, or Ctrl-U to delete the entire command. The editing characters may be used in any combination until the command is finally entered by typing RETURN (Carriage Return, Enter) or Ctrl-L.

You may use the help (``?'') and keyword completion (ESC) features freely while typing Kermit-MS commands. A question mark typed at almost any point in a command produces a brief description, or "menu", of what is expected or possible at that point. ESC typed at any point, except in a local filename, will cause the current field to be filled out if what you have typed so far is sufficient to identify it, and will leave you in position to type the next field (or to type a ``?'' to find out what the next field is); otherwise, the program will beep at you and wait for you to type more characters.

As of version 2.31, Kermit-MS recognizes full 8-bit character inputs, with only NUL, ESC, DEL/BS, Ctrl-W (delete word), Ctrl-U (delete line), and Ctrl-C being special. This is to enhance support for various languages and keyboards. The SET KEY and SHOW KEY commands can prompt for keyboard input and understand 8-bit characters but only at their interactive prompt. The SET KEY, INPUT, and OUTPUT commands accept "backslash number format" on the main Kermit command line. Thus, national characters which are full 8-bit codes can be expressed on command lines in backslash number form (\ddd), provided the Kermit command itself can understand the form. Most commands that want numbers or single characters as operands understand this notation. To enter characters in backslash number format, type a backslash (``\'') followed by a number corresponding to the ASCII code for the character. MS-Kermit accepts many different backslash codes in different contexts. These are summarized in Table 5-1; letters following the backslach may be either upper or lower case.

`\123`	(up to 3 decimal digits) - A decimal number
`\d123`	(up to 3 decimal digits) - A decimal number
`\o123`	(up to 3 octal digits) - An octal (base 8) number
`\x123`	(up to 3 hexadecimal digits) - a hexadecimal (base 16) number
`\{ }`	For grouping, e.g. `\{12}6 = Ctrl-L 6`, not ~
`\;`	Include a semicolon in a TAKE-file command or macro definition.
`\%`	Introduce a Kermit variable, `\%1, \%2, ..., \%a, \%b, ... \%z`
`\K`	A Kermit connect-mode verb like `\Kexit` (see Table 5-6)
`\B`	Send a BREAK (OUTPUT command only)
`\255`	Shorthand for CRLF or LFCR (INPUT command only)
`\CD`	Carrier Detect RS-232 signal (WAIT command only)
`\DSR`	Data Set Ready RS-232 signal (WAIT command only)
`\CTS`	Clear to Send RS-232 signal (WAIT command only)

Table 5-1: MS-DOS Kermit Backslash Codes

Table 5-2 shows all of the 7-bit ASCII codes in decimal. Most Kermit commands understand backslash-ASCII codes, both imbedded within character strings, and alone, as when a single character or number is to be specified.

Some Kermit-MS commands like GET, SHOW KEY, and SET KEY, may prompt for additional information on subsequent lines. If you have reached one of these prompts and then wish to cancel the command, you may type Control-C to get back to the main `Kermit-MS>` prompt.

Dec	Name	Ctrl	Dec	Char	Dec	Char	Dec	Char
0	NUL	^@	32	SP	64	@	96	‘
1	SOH	^A	33	!	65	A	97	a
2	STX	^B	34	"	66	B	98	b
3	ETX	^C	35	#	67	C	99	c
4	EOT	^D	36	$	68	D	100	d
5	ENQ	^E	37	%	69	E	101	e
6	ACK	^F	38	&	70	F	102	f
7	BEL	^G beep	39	’	71	G	103	g
8	BS	^H backspace	40	(	72	H	104	h
9	HT	^I tab	41	)	73	I	105	i
10	LF	^J linefeed	42	*	74	J	106	j
11	VT	^K	43	+	75	K	107	k
12	FF	^L formfeed	44	,	76	L	108	l
13	CR	^M return	45	-	77	M	109	m
14	SO	^N shift out	46	.	78	N	110	n
15	SI	^O shift in	47	/	79	O	111	o
16	DLE	^P	48	0	80	P	112	p
17	DC1	^Q XON	49	1	81	Q	113	q
18	DC2	^R	50	2	82	R	114	r
19	DC3	^S XOFF	51	3	83	S	115	s
20	DC4	^T	52	4	84	T	116	t
21	NAK	^U	53	5	85	U	117	u
23	ETB	^W	54	6	86	V	118	v
22	SYN	^V	55	7	87	W	119	w
24	CAN	^X	56	8	88	X	120	x
25	EM	^Y	57	9	89	Y	121	y
26	SUB	^Z	58	:	90	Z	122	z
27	ESC	^[escape	59	;	91	[	123	{
28	FS	^\	60	<	92	\	124	|
29	GS	^]	61	=	93	]	125	}
30	RS	^^	62	>	94	^	126	~
31	US	^_	63	?	95	_	127	RUBOUT,DELETE

Table 5-2: The US ASCII Character Set (ANSI X3.4-1977)

Summary of Kermit-MS command editing characters:

SPACE Separates fields within the command.

TAB Same as Space, and echoes as Space. You may also use Ctrl-I for Tab.

BACKSPACE
Deletes the character most recently typed. May be typed repeatedly to delete all the way back to the prompt. You may also use DELETE, RUBOUT, Ctrl-H, or equivalent keys.

Ctrl-W Deletes the most recent "word", or field, on the command line. May be typed repeatedly.

Ctrl-U Deletes the entire command line, back to the prompt.

Ctrl-C Cancels the current command and returns to the "`Kermit-MS>`" prompt. Also, terminates execution of a TAKE command file.

ESC If enough characters have been supplied in the current keyword to identify it uniquely the remainder of the field is supplied and the cursor is positioned to the next field of the command. Otherwise, a beep is sounded. ESC does not provide filename completion in version 2.31.

? Displays a brief message describing what may be typed in the current command field. Also, wildcard character for matching any single character in all but the first position of a filename.

\# Wildcard character for matching single characters in filenames. Equivalent to MS-DOS ‘‘?’’, but used in the first position of a filename only, so that ‘‘?’’ may be used to get help at the beginning of a filename field.

RETURN
: Enters the command. On most keyboards, you may also use ENTER or Ctrl-M.

Ctrl-L
: Clears the screen and enters the command.

Liberal use of ‘‘?’’ allows you to feel your way through the commands and their fields. This feature is sometimes called "menu on demand" or "context sensitive help" -- unlike systems that force you to negotiate menus at every turn, menu-on-demand provides help only when it is needed.

Command reading is done through DOS calls and Kermit key redefinition does not apply at Kermit-MS command level. But `ANSI.SYS` or other external console drivers can be used for this purpose, for instance to assign ESC to the PC's backquote key (`ANSI.SYS` is the IBM-supplied extended screen and keyboard device driver, described in the IBM DOS Technical Reference Manual). Other console drivers available include ProKey, SuperKey, `NANSI.SYS` (a public-domain replacement for `ANSI.SYS`), and FANSICONSOLE.

The notation used in command descriptions is as follows:

[square brackets]
: An optional field. This field may be omitted.

{curly braces}
: A list of alternatives, separated by commas. Choose one of the items from the list.

italics
: Shows parameters, such as numbers or filenames, are shown in italics (providing the printer is capable of printing italics). You substitute the actual number or filename.

<u>underlining</u>
: In dialog examples, the characters you should type are underlined (on printers that can show it) to distinguish them from computer typeout.

`hh:mm:ss`
: A time of day, in 24-hour notation (10:00:00 is 10 AM; 23:30:00 is 11:30 PM), which may not be more than 12 hours later than the current time.

The following sections describe all the MS-DOS Kermit commands. Since some command descriptions may contain references to other commands that haven't been explained yet, you might find that this manual makes more sense on a second reading.

5.6.1. Program Management Commands

"Program management" is a rubric for Kermit-MS commands like TAKE, EXIT, HELP, COMMENT, ECHO, and VERSION, that don't fall into any other category.

HELP displays a one screen introduction to frequently used Kermit commands and their editing keys, and suggests using the question mark command to see the terse list of primary level Kermit commands.

VERSION displays the MS-Kermit program version number, which you should know in case you are reporting bugs or seeking technical assistance.

Other program management commands require a bit more explanation.

The EXIT Command

Syntax: `EXIT` *or* `QUIT`

EXIT and QUIT are synonyms for each other. They cause MS-Kermit to return control to DOS or whatever program invoked MS-Kermit. The specific actions taken are:

- Close any open log or other files.
- Close any open network connection.
- Release all memory claimed by the program.
- Return interrupts for the currently selected communication device to their original owner.
- Terminate execution.

The serial port RS-232 signals are left alone upon EXIT, so that modem connections are not broken. Kermit-MS may be restarted with the connection intact. Use HANGUP to explicitly break a modem connection; and use SHOW MODEM or SHOW COMMUNICATIONS to view the status of modem signals CARRIER DETECT, DATA SET (modem) READY, and CLEAR TO SEND.

The STAY Command

Syntax: `STAY`

The STAY command, if included among command line arguments, instructs MS-Kermit not to exit upon completion but rather to enter interactive mode, unless EXIT or QUIT was among the command arguments. STAY has no effect when entered interactively or from a TAKE file.

The PUSH Command

Syntax: `PUSH`

PUSH is similar to EXIT, except it leaves MS-Kermit intact by invoking an MS-DOS command processor "under" Kermit-MS, either `COMMAND.COM` or whatever shell you have specified with COMSPEC (or SHELL, depending on the system) in your `CONFIG.SYS` file. You can return to Kermit-MS by typing the MS-DOS EXIT command, and you will find Kermit-MS as you left it, with all settings and the terminal emulation screen intact. The same function is invoked by the CONNECT escape-level command P. Example:

```
Kermit-MS>push                     Push to DOS.
Command v3.10                      COMMAND.COM program herald.

C>diskcopy a: b:                   Run a DOS program.

  DISKCOPY dialog here . . .

C>dir b:                           More DOS commands . . .

  DOS session continues . . .

C>exit                             When done, type DOS EXIT command.
Kermit-MS>                         Back at Kermit.
```

The TAKE Command

Syntax: `TAKE` *filespec*

The TAKE command gives you way a to collect MS-Kermit commands into a single file, so that you can execute many commands by typing a single (TAKE) command. TAKE instructs MS-Kermit to execute commands from the file that you specify. The current directory is searched for the file first, and then any directories listed in the PATH environment variable. The command file may include any valid Kermit-MS commands, including TAKE, but it cannot include characters to be sent to a remote host after a CONNECT command (use scripts for that, described below). Execution of a TAKE file may be cancelled by typing Control-C at the keyboard.

An implicit TAKE command is executed upon the initialization file, `MSKERMIT.INI` (or another file specified in the ‘‘-f’’ command-line argument), whenever you start MS-Kermit. The MSKERMIT.INI file contains any commands you want to be executed each time you run Kermit. A sample is shown above, and a more ambitious example is shown in section 5.9.

Commands within TAKE files, unlike interactive commands, may include trailing comments, preceded by semicolons:

```
set port 2       ; Select the modem port.
set speed 1200   ; Set the baud rate for the modem.
connect          ; Conduct a terminal session.
hangup           ; Hang up the phone after escaping back.
```

Note the HANGUP command after CONNECT. The HANGUP command is not executed until after you escape back from your CONNECT session. If this file were called MODEM.CMD, the following TAKE command would execute it:

```
Kermit-MS>take modem.cmd
```

This directs MS-Kermit to find the MODEM.CMD file, open it, execute the commands in it, close it, and return to the MS-Kermit> prompt when done. This process can take a while on floppy-disk based systems.

Since TAKE file processing discards all characters from a line beginning with the first semicolon, it is normally not possible to include semicolons as part of the commands themselves, e.g.

```
get dska:foo.bar;6
```

To get around this restriction, you may precede such semicolons with a backslash:

```
get dska:foo.bar\;6
```

Commands from the TAKE file will normally not be displayed on your screen during execution. If you want to see them as they are executing, you can SET TAKE-ECHO ON (for instance, at the beginning or end of your MSKERMIT.INI file). With the echoing ON, comments are also displayed for reference, but the semicolon is not shown.

TAKE files may be nested to a reasonable level. A command file that was invoked by another command file normally returns to its invoking command file, rather than to the MS-Kermit> prompt, when the end of the command file is reached.

TAKE files have two commands to quit processing before the end of the file is reached. The POP command exits the current TAKE file (or macro) and returns control to the previously executing TAKE or macro, where one is invoked within another. The STOP command exits all TAKE files and macros and returns directly to the Kermit prompt.

In TAKE files (and macro definitions, which are discussed later), long commands may be continued on subsequent lines by terminating each continued line with a hyphen (minus sign). If a line needs to terminate on a real minus sign it may be expressed numerically as \45 or can be extented with extra spaces. The overall command length is normally 127 bytes (a beep sounds near this limit).

An explicit quertion mark (‘‘?’’) in a TAKE file will cause a help message to be displayed and the rest of the line will be read as another command. If you need to include a question mark in a command, use the ASCII backslash notation "\63".

The -F Command

Syntax: `-F` *filespec*

The ‘‘`-f`’’ command is effective only on the DOS command line. It instructs MS-Kermit to use the specified file as its initialization file, rather than `MSKERMIT.INI`. Unlike other command-line arguments, ‘‘`-f`’’ does not, of itself, cause MS-Kermit to exit upon completion. Example:

```
C>kermit -f sunday.ini
Kermit-MS>
```

The -F command line option allows different MS-Kermit initialization files to coexist. You can create batch commands to invoke Kermit in different ways, for instance `MONDAY.BAT` might contain ‘‘`kermit -f monday.ini`’’, `TUESDAY.BAT` ‘‘`kermit -f tuesday.ini`’’, etc.

The ECHO Command

Syntax: `ECHO [`*string*`]`

The ECHO command writes the string to the screen, without adding a carriage return or line feed. ECHO may be used to report progress during execution of a TAKE command file, or to issue prompts during the execution of a script.

```
ECHO Part one completed...\13
```

The number at the end is a "backslash codes" for ASCII control characters, in this case carriage return (`\13`). Since the ECHO command interprets backslash codes, `ANSI.SYS` and similar console drivers can be programmed through this command by embedding ANSI escape sequences (see section 5.16.3) in the echo string. The ECHO command always outputs a linefeed before the string.

The COMMENT Command

Syntax: `COMMENT` *text*

The COMMENT command lets you add comments to a TAKE command file. The word COMMENT (or any unique prefix thereof) must appear as the first word on the line. The COMMENT command may also be entered interactively. It has no effect at all. Example:

```
COMMENT - MS-Kermit command file to connect port 2 to an IBM mainframe
set port 2
set speed 4800   ; Transmission rate is 4800
do ibm           ; Set parameters for IBM linemode
connect          ; Be a terminal
```

Question marks can be included in comments without invoking the help function.

5.6.2. Local File Management Commands

These commands are executed on your local PC, and generally invoke DOS services. This allows you to perform common DOS functions without leaving Kermit. All file specifications may include device and/or directory fields. The local file management commands are:

CWD *path*

Changes the current working directory to the given path. All references to local file names without explicit paths will refer to that path. A drive letter may be included to also change disk drives. This command affects Kermit and any inferior programs that you RUN or PUSH to, but your previous disk and directory are restored when you exit from Kermit. For consistency with DOS, you may also type CD.

DELETE *filespec*

Deletes the specified file or files. As in DOS, the names of the deleted files are not listed, only the message "file(s) deleted" or "file(s) not found", and if you give the command "delete `*.*`", Kermit-MS will prompt "Are you sure?" since DOS is doing the work.

DIRECTORY [*filespec*]
Lists the names, sizes, and creation dates of files that match the given file specification. If no filespec is given, the command is equivalent to `DIR *.*`. Normal DOS switches are effective.

SPACE Tells how much space is available on the current disk.

RUN *command*
Passes the command line to `COMMAND.COM` for execution. Any legal DOS operation is permitted: running a program (perhaps with command line arguments or i/o redirection), executing a DOS command, or executing a batch file. Kermit is suspended while the command is executed and automatically resumes afterward. You may even nest RUN KERMIT commands several times if memory is available. The command will be executed directly by `COMMAND.COM` so follow the rules of DOS. Example:

```
Kermit-MS>run more < kim.txt
```

TYPE *filespec*
Displays the specified local file on the screen. Automatic pause is not available at the end of a page (but see above example for how to accomplish this). On most systems, Ctrl-S can be typed to stop scrolling and Ctrl-Q to continue scrolling.

In most cases when you issue a local command, Kermit attempts to run the equivalent DOS command. If you get a message like "`?Unable to execute program`", it means that Kermit could not find `COMMAND.COM`, or that there was not enough memory left to load it. To ensure that Kermit can find `COMMAND.COM`, you should include a PATH statement in your `AUTOEXEC.BAT` file, which includes the device and directory where `COMMAND.COM` resides.

You can add your own local commands by defining macros for them. For example:

```
define edit run epsilon \%1
define more run more < \%1
define rename run ren \%1 \%2
```

Then you can use these commands at Kermit-MS prompt level: "edit foo.bar", "more oofa.txt", "rename old.txt new.txt". However, you cannot redefine built-in commands, for example:

```
define send receive \%1
```

See Section 5.7 for further information about macros.

5.6.3. COMMANDS FOR TERMINAL CONNECTION

The CONNECT command connects your PC as a terminal to the remote system so that you may conduct a session there, and the HANGUP command may be used to disconnect your modem (if you have one) from the remote system. There is presently no built-in DIAL command. Modems may be dialed "manually" during CONNECT, or you can construct your own DIAL command by using scripts, which are described in detail in subsequent sections.

For completeness, the descriptions below contain copious reference to the SET commands, which let you modify all sorts of terminal and communication parameters (the SET commands are described in a later section). MS-Kermit is initially set up with the following parameters, so that you only need to issue SET commands for those that need to be changed:

PORT	1 (in most cases, e.g. COM1 on the IBM PC family)
TERMINAL	VT102(*) emulation (IBM PC, DEC Rainbow)
SPEED	Whatever the serial card is currently set to.
PARITY	None
FLOW-CONTROL	XON/XOFF
HANDSHAKE	None
LOCAL-ECHO	Off
DISPLAY	7-bit characters
INPUT TRANSLATION	Off
ESCAPE	Control-Rightbracket

(*) The VT102 terminal is compatible with the VT100, but includes a few additional functions.

The CONNECT Command

Syntax: `CONNECT` *-or-* `C`

The CONNECT command establishes an interactive terminal connection to the remote system using the currently selected communications port (SET PORT COM1 or COM2) with all settings currently in effect for that port, emulating the currently selected type of terminal.

During CONNECT, the characters you type are sent out the communication port, and the characters that arrive at the port are displayed on the screen or interpreted by the selected terminal emulator. If you SET LOCAL-ECHO ON, MS-Kermit itself will display the characters you type on the screen.

Before you issue the CONNECT command, be sure to set the correct communication speed (SET SPEED) and any other necessary communication parameters (e.g. SET PARITY, SET LOCAL-ECHO). If you have SET DEBUG ON, then (on certain systems, particularly the IBM PC), received control characters will be displayed in special notation and no particular terminal will be emulated.

By default, 7-bit ASCII characters are displayed on the screen. If you SET DISPLAY 8, then 8-bit characters will be used (useful for "national" character sets). Character translation will be done according to any SET TRANSLATION INPUT and SET KEY commands you have issued. In addition, characters that are sent to the screen will also be recorded in a disk file or on a printer if you have issued a LOG SESSION command.

The CONNECT command turns your PC into a terminal to the other computer. To get back to the PC, type the escape character followed by the letter C (for "Close connection"). On most MS-DOS systems the escape character is Ctrl-`]` (Control-Rightbracket). That means, hold down the Ctrl key, press ‘‘`]`’’, and then type the letter C.

```
Kermit-MS>connect          Connect to remote system.

   Conduct terminal session here . . .

^]c                        Escape back to PC.
Kermit-MS>                 Prompt reappears.
```

This is called "escaping back". You can use the SET ESCAPE command to change the escape character to something besides ‘‘^]’’, or you can assign the escaping-back operation to a single key or key combination with SET KEY (on the IBM PC the default for this is Alt-X).

You can include the CONNECT command in a TAKE command file, but not "bare" text to be sent to the remote system during CONNECT (use scripts for that, see Section 5.8). When a TAKE file includes a CONNECT command, no further commands will be executed from the file until after you escape back. A curious side effect of allowing Kermit to accept input redirected from a file or device is that Connect mode will read characters from that file or device; not really that useful but it works if you happen to need it.

When you CONNECT, the program attempts to raise the DTR and RTS RS-232 signals (see Table 5-3), and it takes no specific action to lower them unless you explicitly issue the HANGUP command; thus you can EXIT from Kermit-MS and restart it without dropping a dialup connection. While CONNECTed, you can communicate directly with an autodialer or "smart modem" to control the communications line, hang it up, and the like, for instance, by typing AT commands to a Hayes-like modem.

```
Kermit-MS>set speed 2400       (See Section 5.6.10)
Kermit-MS>connect
AT                             Now you're talking to the modem.
OK                             Your modem responds
ATDT8765432                    Type the modem's dialing command.
RINGING
CONNECT 2400
Welcome to ...                 Now you're talking to the host computer.
Please login:
```

MS-Kermit makes no attempt to monitor the modem’s Carrier Detect (CD) or Data Set Ready (DSR) signals (see Table 5-3), and will take no notice if they drop. Thus it is not possible to automatically terminate a session if the connection is broken. However, you may query or test the status of these modem signals yourself using Kermit’s SHOW MODEM, SHOW COMMUNICATIONS, and WAIT commands.

Signal	DB25	DB9	Description
FG	1	-	Frame (protective) ground
TD	2	3	Transmitted data (from PC to modem)
RD	3	2	Received data (by PC from modem)
RTS	4	7	Request to Send (by PC)
CTS	5	8	Clear to Send (by modem)
DSR	6	6	Dataset Ready (Modem is turned on)
SG	7	5	Signal Ground
CD	8	1	Carrier Detect (Modem is communicating with remote modem)
DTR	20	4	Data Terminal Ready (PC is online)
RI	22	9	Ring Indicate (Modem tells PC phone is ringing)

Table 5-3: RS-232-C Modem Signals

When using Kermit to connect two PCs "back to back," SET LOCAL-ECHO ON so that when you CONNECT to the other PC to send messages to its operator, you can see what you are typing. You should also SET TERMINAL NEWLINE ON, so that that a linefeed will be automatically supplied for each carriage return you type.

The HANGUP Command

On serial port connections, the HANGUP command attempts to momentarily lower the modem signals DTR and RTS (Table 5-3). It may be used to hang up the phone when dialed up through a modem, or to get the attention of port contention units or terminal concentrators that operate in this manner. On direct connections, it will probably have no effect. On local area network connections, the network session is fully terminated. HANGUP affects only the currently selected port.

TERMINAL EMULATION

The IBM PC version of Kermit-MS emulates the DEC VT102 terminal by default, and may also be instructed to emulate the DEC VT52, the Heath/Zenith-19, the Tektronix 4010 graphics terminal, or no terminal at all, selectable with the SET TERMINAL command (or you may "toggle" among the different emulations by typing the Alt-Minus key). Emulation of each of these terminals is nearly complete. VT102 emulation lacks only smooth scroll and 132 column mode (132 column mode is supported for a number of popular EGA and VGA boards). Double-height, double-width characters are supported, but simulated using ordinary characters.

The IBM PC's 40-column (large character) screen mode may be used during CONNECT (but you may also have to inform the remote host that your screen width is 40). This can provide improved readability to visually impaired persons. To use 40-column mode, enter the DOS command "MODE 40" (or CO40 or BW40). Other screen sizes are also sensed and used automatically.

On color monitors, the foreground and background colors may be set using SET TERMINAL COLOR, and inverse/normal video display may also be selected, along with many other terminal parameters. A complete list of the commands, default key configurations, and escape sequences accepted by the IBM PC Kermit terminal emulator is given in section 5.16.1. Non-IBM-compatible PCs have different terminal emulation options. See section 5.10.

Escape-Level Commands

The escape character, normally Control-`]`, is used to regain the attention of Kermit-MS during CONNECT (you can change the escape character using SET ESCAPE). When you type the escape character, Kermit-MS waits for you to follow it with a single character command. For instance, the single character command ''`?`'' produces a list of available single character commands. This command is executed immediately; it may not be edited, and the program does not wait for a carriage return to confirm it. Table 5-4 shows CONNECT escape-level commands available in Kermit-MS. Typing any other character (except the space bar, which is the "null command") after the

`?`	Help -- Lists the available single-character commands.
`0`	(the digit zero) Transmit a NUL (ASCII 0).
`B`	Transmit a BREAK signal.
`L`	Transmit a Long BREAK signal (on some systems).
`C`	Close the connection and return to Kermit-MS prompt level.
`H`	Hangup the phone by lowering DTR and CTS momentarily.
`F`	File the current screen in the screen dump file.
`M`	Toggle the mode line, i.e. turn it off if it is on or vice versa.
`P`	Push to DOS; get back to CONNECT by typing EXIT.
`Q`	Temporarily quit logging the remote session.
`R`	Resume logging the remote session.
`S`	Show the status of the connection.
`^]`	(or whatever you have set the escape character to be) Typing the escape character twice sends one copy of it to the connected host.

Table 5-4: Kermit-MS Single-Character CONNECT Escape Commands

escape character will cause Kermit-MS to beep, but will do no harm. These actions are also Kermit action verbs and can be assigned to single keys. See SET KEY for details.

The Mode Line

When you first issue the CONNECT command, a message (on some systems, an inverse video "mode line") will display the most important facts about the connection you've just established, so that you can quickly diagnose any problems. Here's what the IBM PC mode line looks like:

```
Esc-chr:^] help:^]? port:1 speed:9600 parity:odd echo:rem VT102 .... PRN
```

This shows that the escape character is Ctrl-Rightbracket, that you would type Ctrl-rightbracket followed by question mark (``^]?'') to get help during CONNECT, that you are connected on port 1 at 9600 baud with odd parity and remote echo, and that a VT102 terminal is being emulated. The four dots represent the VT102s LEDs (they turn into the digits 1,2,3,4 when "lit") and PRN will show up if the printer is activated (e.g. by Ctrl-PrintScreen).

The mode line may be turned on and off using SET MODE, or the CONNECT escape character followed by the letter M.

Screen Rollback

On the IBM PC and some other systems (see Table 5-7), Kermit-MS provides several pages of screen memory which let you recall earlier terminal screens. These may be scrolled up and down using keys as shown in Table 5-8. For instance, the IBM PC uses PgUp (previous screen), PgDn (next screen), Ctrl-PgUp and Ctrl-PgDn (one line at a time). Only lines that scroll off the top of the screen are saved. When an application clears the screen, that screen is not saved. These functions and others may be assigned to different keys with the SET KEY command.

If you have rolled the screen back and a new character must be displayed, it will normally appear at the current cursor position on the old screen. This is useful when you are trying to copy something from a previous screen. If you wish new characters to appear in their proper place on the "newest" screen, you can SET TERMINAL ROLL ON.

The number of lines in the roll back buffer depends on the machine, 10 full screens for IBM PCs and DEC Rainbows, and on the amount of memory available in the machine. Each screen needs 4KB on IBM PCs. Denser displays receive fewer roll back lines.

Screen Dump

The screen dump feature writes the contents of the current screen to a file (`KERMIT.SCN` unless another file was selected by the SET DUMP command) when the CONNECT escape-level command F is typed. The screen dump file is appended to on each successive screen dump, with each screen separated by a formfeed (Ctrl-L). This feature may be used in conjunction with screen rollback -- a handy way to recapture screenfuls of laboriously typed-in text after a remote host has crashed without saving your work. The corresponding action verb is "dump". Screen dump does not function when in Tektronix graphics mode; instead one of many graphics screen capture programs may be used independently commonly via the DOS Shift PrtSc key combination or by LOGging the incoming byte stream.

A screen dump differs from a session log in two ways. First, each desired screen must be manually filed, and second, the screen dump file has been stripped of any escape sequences, whereas the session log records them (see LOG SESSION).

Printer Control

During terminal emulation, a locally attached printer may be controlled in the normal manner, on most systems. Pushing the "Print Screen" key (shifted on some systems) will cause the current contents of the screen to be printed by DOS; holding down Ctrl while depressing Print Screen will alternately start and stop the spooling of incoming characters to the printer. On the IBM PC, the mode line will show PRN when the printer is activated in this manner. `^P` or `^N` are sent to the host during terminal emulation and do not toggle printing as they do when you're talking directly to DOS. CTRL-Print-Screen can be simulated with the Kermit-MS LOG PRN and CLOSE commands. VT102 (ANSI) style host-controlled transparent printing is also supported on the IBM PC. See section 5.17.6 for technical information about MS-Kermit's printer control.

Graphics

MS-Kermit on the IBM PC, compatibles, and several other systems, is capable of emulating a Tektronix 4010 graphics terminal, for use with host-based software that can generate Tektronix control codes. When you enter Tektronix emulation, your cursor will disappear. Don't be alarmed, this is how Tektronix terminals behave.

The Tektronix emulator implements a mixture of Tek 4010 and 4014 features to draw characters, lines, and dots in graphics mode. These Tektronix terminals have a graphics display 780 dots high by 1024 dots wide. They use storage tube technology whereby a dot stays illuminated until the full screen is erased. They also lack cursor keys. Kermit's Tek emulator maps the 1024 by 780 dot display to the PC's current screen dimensions, say 640 across by 200 or 350 dots high, and retains limited use of the cursor keys. It automatically senses the active display adapter (EGA, CGA, Hercules, Mono, and AT&T/Olivetti style 640x400) and retains screen coloring (EGA) and the current graphics image (EGA and Hercules) if the adapter has sufficient memory. Automatic sensing can be manually overriden to select a particular display mode, such as VGA (640x480), by SET TERMINAL GRAPHICS <display type>. Pure monochrome systems, of course, lack a graphics capability; in this case Kermit approximates the graphic image by writing dots as plus signs.

Tektronix graphics mode is entered two different ways, automatically and voluntarily:

1. Automatically (which you can prevent via the Kermit command DISABLE TEK). While emulating a VT102, VT52, or Heath-19, reception of the byte pair ESCAPE Control-L causes the PC to change to graphics mode, clear the screen, and obey new input as Tektronix commands. A second automatic entry is reception of the escape sequence ‘‘`ESC [ ? 3 8 h`’’ which does the same as above except the screen is not cleared. Automatic mode is exited by either reception of Control-X or ‘‘`ESC [ ? 3 8 l`’’ (lower case L), or by toggling the terminal type (ALT minus, Kermit verb`\KTermtype`) to VT102, or something other than TEK. (These ‘‘`ESC [ ? 3 8 h/l`’’ sequences derive from the DEC VT340 terminal.)

2. Voluntary mode is when terminal type TEK4010 is selected by the Kermit command SET TERMINAL TEK4010 or by toggling to it using Alt-Minus. It is exited by SET TERMINAL another-kind or by toggling to another kind. ENABLE or DISABLE TEK and the exit-Tek-mode escape sequences are not applicable to voluntary mode.

Here are several common questions about Tek mode, and their answers:

1. *"How do I escape from graphics mode back to being a regular terminal?"* Within CONNECT mode, you can type the `\KTermtype` key, which is assigned by default to Alt-Minus. Repeated pressing of this key "toggles" among Kermit's terminal types, VT102, VT52, Heath-19, and Tektronix. You can also escape back to Kermit-MS command level and issue an explicit SET TERMINAL command to change the terminal type.

2. *"How can I return to the graphics screen without erasing it?"* The graphics screen is preserved if your graphics adapter has sufficient memory (see Table 5-5). In this case, both your text and graphics screens will be preserved when you toggle back and forth between a character terminal (e.g. VT102) and Tektronix.

3. *"How do I erase the graphics screen?"* You can type the `\KReset` key, which is normally assigned to Alt-=. The screen also clears if the host sends a Control-L or ESC Control-L.

4. *"How do I print or save the graphics screen?"* Kermit does not currently provide a way to do this, but you can load drivers like `GRAPHICS.COM` alongside Kermit for this purpose.

While acting as a Tek terminal Kermit uses the keyboard translation appropriate to the VT102 terminal. However, received escape sequences are interpreted by the Tek emulator and VT102 escape codes are inoperative. The Tek emulator absorbs the ESCAPE and following character and treats any additional unknown items as ordinary text.

The emulator can display text characters from a built-in 8-by-8 dot font for characters Space through DELete (no control codes nor special characters). Tabs are converted to single spaces. Only the low 7 bits of the character are used.

While in Tek mode the emulator behaves as a simple TTY device for ordinary text and as a line or dot drawing Tektronix device for commands listed in Table 5-10. The screen resolution is governed by the kind of active display adapter and monitor in the PC (Table 5-5). Kermit senses this automatically when graphics mode is entered. Graphics are saved on page 1 of screen memory. Coloring is determined by the current terminal status, either the default screen or that overridden by the command SET TERMINAL COLOR.

Display Adapter	Display	Mode	Screen Resolution and Coloring
VGA	Hi res color	18	640x480, graphics saved (407 lines), 16 colors.
VGA	Monochrome	17	640x480, graphics saved (407 lines)
EGA w/256KB	Hi res color	16 dec	640x350, graphics saved, 16 colors.
	Med res color	14	640x200, graphics saved, 8 colors.
	Monochrome	15	640x350, graphics saved, b/w.
EGA w/64KB	Hi res color	16	640x350, graphics not saved, 4 colors of red, white, blue, black.
	Med res color	14	640x200, graphics saved, 8 colors.
	Monochrome	15	640x350, graphics not saved.
CGA	Color	6	640x200, graphics not saved, b/w.
Hercules	Monochrome	none	720x348, graphics saved if memory.
Monochrome	Monochrome	7	80 by 25 text, graphics not saved.
AT&T/Olivetti	any	72	640x400, grahics not saved, b/w.
DEC VAXMATE	any	208	640x400, graphics not saved, b/w.
TOSHIBA T3100	any	116	640x400, graphics not saved, b/w.

Table 5-5: Adapters Supported by IBM PC MS-Kermit for Tektronix Emulation

The technical details of Tektronix emulation are presented in section 5.16.7.

5.6.4. COMMANDS FOR FILE TRANSFER

MS-Kermit's SEND, GET, and RECEIVE invoke the Kermit file transfer protocol for error-checked transmission of files between MS-Kermit and another Kermit program on the other end of the connection. There are also commands for "raw" transfer of files (no error checking) with systems that don't have Kermit programs: LOG SESSION (for capturing text files on your PC) and TRANSMIT (for uploading text files to the remote system). The LOG TRANSACTION command opens a file to record the status, time, date, names, sizes of each file transfer.

During file transfer, MS-Kermit normally displays its progress on the screen as shown in Figure 5-1. The items in the right-hand column are updated more or less at random. The percent done is always filled in when sending files, and when receiving if the other Kermit sends the file's size in a special file-attribute packet. Several other display options are also available; see SET DISPLAY.

```
               Kermit-MS: V2.31  17 July 1988

               File name: FOT.
      KBytes transferred: 7
     Percent transferred: 52%
                 Sending: In progress

       Number of packets: 74
           Packet length: 93
       Number of retries: 2
              Last error: None
            Last warning: None
```

Figure 5-1: MS-Kermit File Transfer Display Screen

Although MS-Kermit makes no distinction between text and binary files, most other Kermit programs do. Therefore, before you attempt to transfer binary files with another type of system (say, a VAX, or an IBM mainframe), be sure to give the appropriate command -- usually something like SET FILE TYPE BINARY -- to the Kermit on the remote end. Kermit-MS itself neither has nor needs the command SET FILE TYPE, because the MS-DOS format for text files is exactly the same as Kermit's text-file transfer format, which means that MS-Kermit never needs to convert file data, no matter whether it be text or binary.

File transfers involving floppy disks will be slow and noisy. Hard disks are much faster (and quieter), and RAM disks faster still (and totally silent). But if you store new files on a RAM disk, be sure to move them to a real disk before turning off your PC.

Before attempting to transfer files to the PC, make sure you have enough room on the selected device. Kermit does not provide a way for you to change disks during a file transfer. However, the Kermit protocol will help you out a little bit by attempting to prevent transfer of files that are too big to fit in the available space. As of version `2.31`, MS-Kermit supports "file attributes" exchange, and if the other Kermit supports this option too, then the receiving program will check free disk space before letting the transfer proceed. MS-Kermit allows a margin of 6 percent inflation upon reception, because file construction differs markedly between systems. A multiple-file transfer can even skip automatically past files that are too big, allowing the little ones to pass though.

Other attributes exchanged by MS-Kermit include the file's creation date and time, and the system of origin. When two Kermit programs both have attribute capability, then files will be stored with the same timestamp on the receiving system as they had on the sending system.

Since exchange of attributes is a new feature to MS-Kermit, and a relatively scarce one elsewhere, it is possible that two Kermit programs might misunderstand each other because of differing interpretations by the programmers, and this could prevent otherwise normal file transfers from taking place. An escape clause is provided by the command SET ATTRIBUTES OFF, which makes MS-Kermit forget that it has attribute capability.

You may record the progress of a file transfer in a log file by issuing the command LOG TRANSACTIONS.

The SEND Command

Syntax: `SEND` *filespec1* [*filespec2*]

The SEND command causes a file or file group to be sent from the local MS-DOS system to the Kermit on the remote system. The remote Kermit may be running in server or interactive mode; in the latter case, you should already have given it a RECEIVE command and escaped back to your PC. S is a special non-unique abbreviation for SEND.

filespec1 may contain the wildcard characters ‘‘`*`’’ to match zero or more characters within a field, and/or ‘‘`#`’’ (first position) or ‘‘`?`’’ (elsewhere) to match any single character (a question mark in first position gives you a help message). If *filespec1* contains wildcard characters then all matching files will be sent, in the same order that MS-DOS would show them in a directory listing. If *filespec1* specifies a single file, you may direct Kermit-MS to send that file with a different name, given in *filespec2*, as in:

```
Kermit-MS>send foo.bar framus.widget
```

filespec2 begins with the first nonblank character after *filespec1* and ends with the carriage return; thus it may contain blanks or other unusual characters that may be appropriate on the target machine. The alphabetic case of text in *filespec2* is preserved in transmission, so if case matters on the target system, be sure to type *filespec2* appropriately.

If the SEND command is specified by itself on the command line, then you will be prompted separately for the name of the file to send, and the name to send it under:

```
Kermit-MS>send
 Local Source File: c:\chris\xcom1.txt
 Remote Destination File: com1.txt
```

If a file can’t be opened for read access, the message "Unable to find file" will be shown or else the standard MS-DOS recovery procedures will take place:

```
Not ready error reading drive A
Abort, Retry, Ignore?
```

Kermit remains active even if you select "Abort" (DOS’s word, not ours).

Files will be sent with their MS-DOS filename and filetype (for instance `FOO.TXT`, no device or pathname). Special characters in the file name are not converted. If there is no filetype, then only the name will be sent, without the terminating dot. Each file is sent as is, with no conversions done on the data, except for possibly stopping at a terminating Control-Z character (see the SET EOF command).

Once you give Kermit-MS the SEND command, the name of each file will be displayed on your screen as the transfer begins. Packet, retry, and other counts will be displayed along with informational messages during the transfer, in the style specified by SET DISPLAY. If the file is successfully transferred, you will see ‘‘`Complete`’’, otherwise there will be an error message. When the specified operation is done, the program will sound a beep.

Several single-character commands may be given while a file transfer is in progress:

- `^X` (Control-X) Stop sending the current file and go on to the next one, if any.
- `^Z` Stop sending this file, and don’t send any further files.
- `^C` Return to Kermit-MS command level immediately without sending any kind of notification to the remote system. (`^Z` or even `^E` is preferable.)

^E Like ^C, but send an Error packet to the remote Kermit in an attempt to bring it back to server or interactive command level.

CR Simulate a timeout: resend the current packet, or NAK the expected one.

Control-X, Control-Z, and Control-E send the proper protocol messages to the remote Kermit to bring it gracefully to the desired state. Control-C leaves the remote Kermit in whatever state it happens to be in, possibly retransmitting its last packet over and over, up to its retry limit. You should only have to use Control-C in dire emergencies (the remote Kermit is stuck, the remote system crashed, etc), or at those times when you realize that you have given a file transfer command to Kermit-MS without first having told the remote Kermit about it.

MS-Kermit does not have a built-in mechanism for sending an entire directory structure, but this may still be done using command files. A program called XSEND, distributed along with MS-Kermit, will construct such a command file automatically.

The RECEIVE Command

Syntax: `RECEIVE [`*filespec*`]`

The RECEIVE command tells Kermit-MS to receive a file or file group from the other system. The file is stored under the name it was transmitted with, except that any illegal characters are translated to X's. Kermit-MS passively waits for the file to arrive; this command is not to be used when talking to a Kermit server (use GET for that). You should already have issued a SEND command to the remote Kermit and escaped back to Kermit-MS before issuing the RECEIVE command. The RECEIVE command is intended for situations where the file name and sending operation originates at the other side; GET originates the request from our side and asks the server to perform the operation. R is a special non-unique abbreviation for RECEIVE.

If the optional filespec is provided, incoming files will be stored under that name. If the filespec is really just a path then files are stored where the path indicates. If it is an actual filename the first incoming file is renamed and any additional files either overwrite the first (if FILE WARNING is OFF) or are renamed slightly from the filespec (digits are added to the end of the main filename part before the dot and extension) if FILE WARNING is ON (the default). The filespec may include any combination of the following fields:

Device designator
: Store the file on the designated device, in the current directory for that device. If no device designator is given, store it on the current default device.

Directory path
: Store the file in the designated directory on the current disk. If no path given, store the file in the current directory.

File name
: Store the file under the name given. If no name is given, store it under the name it was sent under, converted, if necessary, to suit DOS conventions, and modified, if SET WARNING ON, to avoid overwriting any file of the same name in the same directory.

If an incoming file does not arrive in its entirety, Kermit-MS will normally discard it and it will not appear in your directory. You may change this behavior by using the command SET INCOMPLETE KEEP, which will cause as much of the file as arrived to be saved on the disk.

The same single-character commands are available as during SEND:

^X Request that the remote Kermit stop sending the current file, and proceed to the next one immediately. Since this is an optional feature of the Kermit protocol, the remote Kermit might not honor the request.

^Z Request that the remote Kermit terminate the entire transfer; this is also an optional feature that may or may not be supported by the remote Kermit.

^C, ^E, and CR operate in the same way as they do during SEND. In this case, ^E should always do what ^Z is supposed to do.

If WARNING is OFF and you type `^X` or `^Z` to interrupt the transfer, you'll either get a partial new file, or else both the old and the new file of that name will be lost, depending on SET INCOMPLETE. In any case, when WARNING is off, old files with the same name as incoming files will not survive.

Caution: If an incoming file's name (the part before the dot) corresponds to an MS-DOS device name, such as NUL, COM1, CON, AUX, or PRN, output will go to that device, rather than to a file with that name. This is a feature of MS-DOS.

5.6.5. Hints for Transferring Large Files

During a prolonged file transfer session, things can go wrong that are beyond Kermit's control. The longer the session, the greater the probability it will be fatally interrupted. But you can take a few precautions:

- Make sure there is sufficient disk space at the receiving end. If possible, first run a disk utility (such as CHKDSK) to clean out any bad disk blocks.
- If you are using a telephone connection, make sure your session won't be interrupted by call waiting, people picking up other extensions, etc.
- Don't attempt to transfer a single file of many megabytes over a telephone connection. The longer the call, the greater the chance of disconnection (carrier loss). Although it's a bother, it may save time in the long run to break the file up into smaller pieces, transfer the pieces, and then recombine on the other end.
- SET INCOMPLETE KEEP on the receiving end, so that if the transfer fails, then the partial file will be retained. Then chop the part that wasn't transferred into a separate file, reconnect, and send it. Then join the pieces together.

Consider moving truly massive amounts of data on magnetic media. "Never understimate the bandwidth of a station wagon full of magnetic tapes!" (or diskettes).

5.6.6. Commands for Raw Uploading and Downloading

MS-Kermit can be used to send files to, or capture files from, remote systems that do not have Kermit programs available. No error checking or correction is done, so the results can very likely contain corrupted characters, spurts of noise, gaps, or extraneous system messages or prompts. The command for uploading is TRANSMIT, and for downloading LOG SESSION.

To minimize loss of data during these operations, be sure to SET the FLOW-CONTROL and HANDSHAKE parameters to match the characteristics of the system on the other end.

The TRANSMIT Command

Syntax: `TRANSMIT` *filespec* `[`*prompt-character*`]`

The TRANSMIT command provides a basic raw upload (export) facility to send straight ASCII text files to the host without packets, error checking, or retransmissions, but using all the currently selected communication parameters for flow control, parity, etc. Information is read from the disk file a line at a time, sent out the serial port, and the command waits for a single character prompt (normally linefeed) from the host before sending the next file line. A disk file line ends with carriage-return-linefeed (CRLF), but only the carriage return is sent, just as you only type carriage return at the end of a line, not CR and LF. Most remote systems will echo the CR and then also supply a LF, which indicates that they have processed the line and are ready for another one. Setting the prompt to binary zero, `\0`, makes the TRANSMIT command proceed without waiting for a prompt. Pressing the local Return key simulates arrival of a prompt character.

Typically, before using this command to upload a file, you would start a text editor (preferably a line-oriented, rather than full-screen, editor) on the remote host and put it into text insertion mode. When the file has been completely transmitted, you would manually enter the required sequence for getting the editor out of text insertion mode, and then make any necessary corrections by hand. Here's an example for VAX/VMS:

```
Kermit-MS>set flow xon/xoff       Set flow control to match VAX/VMS.
Kermit-MS>connect                 Connect to VAX.
$ edt foo.txt                     Start the EDT editor.
*i                                Put it into "insert" mode.
^]c                               Escape back to Kermit-MS.
Kermit-MS>transmit foo.txt        Upload the file a line at a time.
 ...                              The lines are displayed on your screen.
Kermit-MS>connect                 When done, connect back to the VAX.
^Z                                Type Ctrl-Z to exit EDT insert mode.
*exit                             Exit from EDT to save the file.
$
```

If transmission appears to be stuck, you can wake it up by typing a carriage return on the keyboard. You can cancel the TRANSMIT command by typing a Control-C. Control-Z's or other control characters in the file may have adverse effects on the host. For this reason, you should use TRANSMIT only for files that contain 7-bit printing ASCII characters, spaces, tabs, carriage returns, linefeeds, and possibly formfeeds.

The LOG SESSION Command

Syntax: `LOG SESSION` [*filespec*]

The LOG SESSION command lets you copy the characters that appear on your screen during CONNECT into the specified file on the PC. You can use this command to download files by displaying (usually with a command like TYPE) the file on the remote system while logging is in effect. Example:

```
Kermit-MS>set flow xon/xoff       Set flow control to match VAX/VMS.
Kermit-MS>connect                 Connect to the VAX.
$ type foo.bar                    Give this command, but don't type CR yet.
^]c                               Escape back.
Kermit-MS>log session foo.bar     Start logging.
Kermit-MS>connect                 Connect back.
                                  Now type the carriage return.
This is the file FOO.BAR.         The file is displayed on your screen
Blah blah ...                     and captured into PC file FOO.BAR.
$                                 The prompt is captured too.
^]c                               When done, escape back
Kermit-MS>close session           and close the log file.
```

The PC file `FOO.BAR` now contains a (possibly mutilated) copy of the remote computer's `FOO.BAR` file. It probably has the remote system's prompt at the end, which you can edit out. The session log can also be used to record typescripts, editing sessions, Tektronix graphics output, or any other output from, or dialog with, the remote computer.

During terminal emulation, the LOG command records all the characters that arrive from the remote host in the specified file, including escape sequences, with any input character translations applied according to SET TRANSLATION INPUT. If you have SET LOCAL-ECHO ON, the characters you type will also be recorded. Logging may be suspended and resumed within a terminal session with the CONNECT escape-level commands Q and R. The log file will be composed of 7-bit ASCII bytes if (a) PARITY is other than NONE, or (b) DISPLAY is SET to 7. If DISPLAY is 8 and PARITY is NONE, or if DEBUG is ON, then the log will contain 8-bit bytes.

You may LOG SESSION PRN to cause the logging information to be printed directly on your printer. Any escape sequences that are sent to the screen are also sent to the printer.

If you want to record information without imbedded escape sequences, use the screen dump feature, invoked by the CONNECT escape-level command F, which is described under the CONNECT command.

A session log cannot be played back directly on the PC from the log file. To relive the session, you must transfer it to the remote system and display it in "binary mode" (e.g. cat in Unix) while CONNECTed.

5.6.7. Kermit Server Commands

Kermit-MS can act as a Kermit server, and can also interact with other Kermit servers. Normally, the remote Kermit is put into server mode. Then the local Kermit becomes a "client", and may issue repeated commands to the server without having to connect and escape back repeatedly. Servers can not only transfer files, but can also provide a variety of file management functions. The SERVER command puts MS-Kermit into server mode, and the DISABLE and ENABLE commands modify the behavior of the server.

Kermit servers respond only to information sent as Kermit protocol packets and not to ordinary CONNECT-mode commands. When MS-Kermit is the client, it uses the SEND command (described above) to send files to a server, the GET command (*not* RECEIVE) to get files from a server, the REMOTE commands to invoke the file management functions of the server, and the BYE, FINISH, or LOGOUT commands to shut down the server. The MS-Kermit server can also be returned to interactive mode by typing Ctrl-C or Ctrl-Break on the PC's console keyboard.

The SERVER Command

Syntax: SERVER [timeout]

Kermit-MS is capable of acting as a full-fledged Kermit server for users coming in through one of the communication ports or a local area network. To put Kermit-MS into server mode, first issue any desired SET commands to select and configure the desired port, then DISABLE any undesired functions, and then type the SERVER command. Kermit-MS will await all further instructions from the client Kermit on the other end of the connection, which may be hardwired, or connected through a network or autoanswer modem.

In the following example, a Kermit server is set up for dialing in:

```
Kermit-MS>set port 1
Kermit-MS>set baud 1200
Kermit-MS>hangup
Kermit-MS>connect
ATS0=1
OK
^]c
Kermit-MS>set timer off
Kermit-MS>set warning on
Kermit-MS>disable all
Kermit-MS>server
```

Before putting Kermit in server mode in this case it was necessary to connect to the modem (in this example, a Hayes) and put it into autoanswer mode by typing the ATS0=1 command. Since Kermit packets typically start with a Control-A character check the modem's manual to ensure that character is not a modem command signal; some brands regard Control-A as a hangup request!

Note the command SET TIMER OFF. This disables MS-Kermit's ability to time out when waiting for a packet, but it may be necessary in some cases. For example, certain modems or PBX's will be taken out of answer mode if they receive any characters from the PC before a call is received. SET TIMER OFF prevents the MS-Kermit server from transmitting the periodic NAK packets that would cause this problem, but you should ensure that the calling Kermit has its timer ON to avoid protocol deadlocks. Version 2.31A and later have a special command to disable server timeouts, SET SERVER TIMEOUT 0, while leaving regular protocol timeouts operational.

An optional timeout value can be specified to exit server mode automatically at a certain time. The timeout can be expressed as a number, meaning seconds from now, or as the `hh:mm:ss` form, in 24-hour time of day. Both forms recognize times greater than 12 hours from now as being in the past. For instance, if you want to run a Kermit server for an hour, and then have it exit so that another program can run, use a command file like:

```
set port 1        ; Use COM1
set speed 2400    ; at 2400 bps.
disable all       ; Only allow file transfers in current directory.
server 3600       ; Be a server for 3600 seconds = 1 hour.
exit              ; Exit when done.
```

MS-Kermit `2.31` server mode supports the following requests:

SEND	REMOTE CWD (CD)	REMOTE MESSAGE
GET	REMOTE DELETE	REMOTE SEND
FINISH	REMOTE DIRECTORY	REMOTE SPACE
BYE	REMOTE HELP	REMOTE TYPE
LOGO	REMOTE HOST	REMOTE WHO

REMOTE CWD (CD) can be used to change both directories and devices. The REMOTE MESSAGE command accepts a one line message on the command line which will be displayed on the operator's console. An MS-Kermit Server can DISABLE recognition of selected REMOTE commands to help reduce accidents.

> *CAUTION:* The method used for most of the REMOTE commands is to invoke a task with the user's command line, redirect standard output to a temporary file, `$KERMIT$.TMP`, send that file back to the remote end, and then delete the file. Sufficient space must be available to store this file. To service DOS commands or user tasks `COMMAND.COM` must be located on the DOS PATH.
>
> *FURTHER CAUTION:* Any of these DOS tasks or programs may encounter an error, and in that case, DOS will generally put the familiar "Abort, Retry, Ignore?" message on the server's screen, and will wait for an answer from the keyboard. This will hang the server until a human comes to the keyboard and gives a response. The same thing will happen when any program is invoked that interacts with the real console. DISABLE ALL seems to avoid most unpleasant situations of this kind.

For local network operation with NetBios, the SET PORT NET command (with no node name) must be issued before the SERVER command. MS-Kermit then becomes a network-wide server, and other client Kermits can start a network session with it by using the name of the Kermit Server, which is shown on the server's screen when SET PORT NET is given. The Kermit Server accepts connections from other Kermits, but only one at a time. There may be many Kermit Servers active on the network simultaneously because each has a unique node name. Operations are exactly the same as with serial port usage and the session (equivalent to a dialed phone connection) is maintained between the pair until too many timeouts occur, or the client Kermit issues a HANGUP command, exits to DOS, or SETs PORT NET to another node. In the latter cases, the server remains available for use by other client Kermits. If a client Kermit issues the BYE or FINISH command, the network server is shut down (unless it was started with FIN disabled).

The DISABLE and ENABLE Commands

For security purposes, it may be desirable to leave your PC in Kermit server mode so that it can be dialed in to, but with certain functions unavailable to those who dial in. The DISABLE and ENABLE commands provide this control.

The DISABLE and ENABLE commands affect the following functions, with the effect of DISABLEs noted:

CWD	(CD) Changing of directories, disabled entirely.
DEL	Deletion of files confined to current directory.
DIR	Production of directory listings confined to current directory.
FIN	Shutting down the server (applies also to BYE) disabled entirely.
GET	Getting files from the server confined to current directory.
HOST	Execution of all REMOTE HOST (DOS) commands disabled entirely.
SEND	Forces files sent to server into current directory.
SPACE	Asking the server for a disk space report, disabled.
TYPE	REMOTE TYPE files confined to current directory.
ALL	All of the above.

TEK Automatic invocation of Tektronix graphics mode by host commands. This function is not related to server mode, and is not included in the ALL term.

For reasons which should be obvious, the Kermit server does not provide a REMOTE ENABLE command!

The GET Command

Syntax: `GET` *remote-filespec*

The GET command requests a Kermit server to send the file or file group specified by *remote-filespec*. This command can be used only when Kermit-MS has a Kermit server active on the other end of the connection. This usually means that you have CONNECTed to the other system, logged in, run Kermit there, issued the SERVER command, and escaped back (e.g. ‘‘`^]C`’’) to the local Kermit-MS. In the case of LAN operation, a Kermit server must be running somewhere on the network. If the remote Kermit does not have a SERVER command, then you should use SEND and RECEIVE as described above.

You may use the GET command in a special way to specify a different name for storing the incoming file. Just type GET alone on a line, and you will be prompted separately for the remote filespec and the local filespec:

```
Kermit-MS>get
 Remote Source File: com1 txt
 Local Destination File: a:xcom1.txt
```

The local file name may contain a device field, and/or a directory specification. Device and directory specifications in the local destination file name work the same way as in the RECEIVE command. The multiline GET command is provided so that the distinction between the two files is always clear, which would not otherwise be the case if the foreign filename had spaces in it.

The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. It can contain whatever wildcard or file-group notation is valid on the remote system, including spaces. If the string needs to begin with a question mark (?) then use a sharp sign (#) instead to avoid Kermit’s help message; it will be transmitted as a question mark.

Once the file transfer begins, the GET command behaves exactly like the RECEIVE command.

Warning: If the remote filespec is to contain a semicolon, *and* the GET command is being issued from a TAKE command file, you must prefix the semicolon with a backslash. Otherwise, all characters beginning with the semicolon will be ignored:

```
get me.home\;2
```

5.6.8. Commands for Controlling Remote Kermit Servers

The BYE, FINISH, and LOGOUT commands allow you to shut down a remote Kermit server:

`BYE` When communicating with a remote Kermit server, use the BYE command to shut down the server, log out its job, and exit locally from Kermit-MS to DOS. On local area networks, BYE also terminates the network session.

`FINISH` Like BYE, FINISH shuts down the remote server. However, FINISH does not log out the server’s job. You are left at Kermit-MS prompt level so that you can connect back to the job on the remote system. On local area nets, FINISH shuts down the MS-Kermit server, but in a way that allows it to be restarted as if no interruption had occurred.

`LOGOUT` The LOGOUT command is identical to the BYE command, except you will remain at Kermit-MS prompt level, rather than exit to DOS, so that you can establish or use another connection without having to restart MS-Kermit.

The REMOTE Commands

The REMOTE keyword is a prefix for a number of commands. It indicates that the command is to be performed by a remote Kermit server. Not all Kermit servers are capable of executing all of these commands, and some Kermit servers may be able to perform functions for which Kermit-MS does not yet have the corresponding commands. In case you send a command the server cannot execute, it will send back a message stating that the command is unknown to it. If the remote server can execute the command, it will send the results, if any, to your screen.

Here are the REMOTE commands that Kermit-MS may issue:

`REMOTE CWD` [*directory*]
: (Also REMOTE CD) Ask the server to Change your Working Directory on the remote host, that is, the default source and destination area for file transfer and management. You will be prompted for a password, which will not echo as you type it. If you do not supply a password (i.e. you type only a carriage return), the server will attempt to access the specified directory without a password. If you do not supply a directory name, your default or login directory on the remote system will be assumed and you will not be prompted for a password.

`REMOTE DELETE` *filespec*
: Ask the server to delete the specified file or files on the remote system. In response, the server may display a list of the files that were or were not successfully deleted.

`REMOTE DIRECTORY` [*filespec*]
: Ask the server to display a directory listing of the specified files. If no files are specified, then the list should include all files in the current working directory.

`REMOTE HELP`
: Ask the server to list the services it provides.

`REMOTE HOST` [*command*]
: Ask the server to send the command to the remote system's command processor for execution.

`REMOTE KERMIT` *command*
: Send the command to the remote Kermit for interpretation as a Kermit command in the remote Kermit server's own command syntax.

`REMOTE MESSAGE` *text*
: Send the one line text message to be displayed on the Server's screen.

`REMOTE SPACE` [*directory*]
: Ask the server to provide a brief summary of disk usage in the specified area on the remote host or, if none specified, the default or current area.

`REMOTE TYPE` *filespec*
: Ask the server to display the contents of the specified remote file or files on your screen.

`REMOTE WHO [`*who-spec*`]`
: Ask the server to list actively logged on users; optional who-spec qualifies the list and uses the syntax of the server system.

The Mail Command

Syntax: `MAIL` *filespec address*

The MAIL command is a very close relative of Kermit's SEND command. Mail sends a file, or file group, to a Kermit server with instructions (in an Attribute packet) to submit the file(s) to the host's Mailer utility rather than store them on disk. To round out a mail request a field following the filename is required, and into it we place the address to which the files are to be mailed. Mail addresses vary substantially, but several common forms are "username", "username@host", and "host`::`username". The MAIL command will work only if the Kermit server understands it, otherwise the mail request will be rejected before any files are sent. Kermit-MS can send mail but it cannot receive it, because MS-DOS does not have a mail facility. When sending, there is no way to transmit any fields other than the recipient's address and the message body; fields like subject and cc are not supported.

5.6.9. The LOG and CLOSE Commands

Syntax: `LOG {PACKET, SESSION, TRANSACTION}` [*filespec*]
`CLOSE {PACKET, SESSION, TRANSACTION}`

The LOG command tells MS-Kermit to record the terminal session, file transfer transactions, or the file transfer protocol packets themselves in a log file. If the log file already exists then new material is appended to it. Open log files may be closed (and the associated logging disabled) using the CLOSE command. Open log files are also closed when you EXIT from Kermit.

LOG SESSION is used to record your terminal emulation typescript. It was described above, in the section on file transfer.

The LOG TRANSACTION Command

Syntax: `LOG TRANSACTION` [*filespec*]

The Transaction log is a file recording a pair of text lines describing each file transfer (SEND, GET, RECEIVE, or some REMOTE commands). The lines indicate the local filename (and remote name if different), the time and date of the start of the transfer, the number of bytes transferred, and the status of the transfer. New entries are always appended to old to prevent loss of records. The default filename is `TRANSACT.LOG`. The command SHOW LOGGING displays the current names and which logs are active. The command CLOSE TRANSACTION will voluntarily terminate this class of log; otherwise, it will be closed automatically when Kermit exits.

The LOG PACKETS Command

Syntax: `LOG PACKETS` [*filespec*]

The packet log is for diagnostic purposes and records each Kermit protocol packet sent and received in printable format. Control characters are written as caret-letter and characters with the high bit set are shown as their 7-bit part preceeded by a tilde. The default filename is `PACKET.LOG`. If you experience difficulty with file transfers the packet log is valuable in discovering who said what to whom, even though a copy of the Kermit book is needed to unravel the meaning of each character in a packet.

5.6.10. The SET Command

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

The SET command establishes or modifies parameters for file transfer or terminal connection. You can examine their values with the SHOW or STATUS commands. The following SET commands are available in Kermit-MS:

ALARM	Set alarm clock time, for IF ALARM testing
ATTRIBUTES	Controls whether MS-Kermit uses Attribute packets
BAUD	Communications port line speed (synonym for SPEED)
BELL	Whether to beep at the end of a transaction
BLOCK-CHECK-TYPE	Level of error checking for file transfer
COUNT	Variable for TAKE file and macro IF COUNT testing
DEBUG	Display packet contents during file transfer
DEFAULT-DISK	Default disk drive for file i/o
DELAY	Wait number seconds before Sending a file
DESTINATION	Default destination device for incoming files
DISPLAY	For selecting the type of file transfer display
DUMP	Screen dump file (or device) name
END-OF-LINE	Packet termination character
EOF	Method for determining or marking end of file
ERRORLEVEL	Value returned to DOS Batch files
ESCAPE	Escape character for CONNECT
FLOW-CONTROL	Enable or disable XON/XOFF
HANDSHAKE	Half-duplex line turnaround option
INCOMPLETE	What to do with an incompletely received file
INPUT	Behavior of INPUT command for scripts
KEY	Specify key redefinitions
LOCAL-ECHO	Specify which computer does the echoing during CONNECT
MODE-LINE	Whether to display a mode line during terminal emulation
PARITY	Character parity to use
PORT	Select a communications port
PROMPT	Change the "`Kermit-MS>`" prompt to something else
RECEIVE	Request remote Kermit to use specified parameters
REMOTE	For running Kermit-MS interactively from back port
RETRY	Packet retransmission threshold
SEND	Use the specified parameters during file transfer
SPEED	Communications port line speed (synonym for BAUD)
TAKE-ECHO	Control echoing of commands from TAKE files
TERMINAL	Emulation and parameters
TIMER	Enable/disable timeouts during file transfer
TRANSLATION	Enable/disable/specify conversion of arriving characters
WARNING	Specify how to handle filename collisions

The SET commands are now described in detail, in alphabetical order.

SET ALARM

Syntax: `SET ALARM` {*seconds, hh:mm:ss*}

The alarm is a timer, like an alarm clock, available for testing by IF ALARM statements. The alarm time is given as seconds from the present or as a 24-hour specific time of day. Both need to be within 12 hours of the present to avoid being mistaken for times in the past. SHOW SCRIPT displays the current alarm setting.

SET ATTRIBUTES

Syntax: `SET ATTRIBUTES {ON, OFF}`

Disables or enables use of Kermit file Attribute protocol packets, which contain the size, time, and date of files transferred with packets. This command is a safety feature so that a small misunderstanding with another Kermit cannot block transfers. SHOW FILE tells whether attributes are on or off; they are normally ON.

SET BAUD

Syntax: `SET BAUD` *number*

Synonym for SET SPEED (q.v.).

SET BELL

Syntax: `SET BELL {ON, OFF}`

Specifies whether the bell (beeper) should sound upon completion of a file transfer operation. Normally ON.

SET BLOCK-CHECK-TYPE

Syntax: `SET BLOCK-CHECK-TYPE {1, 2, 3}`

Selects the error detection method: a 1-character 6-bit checksum (the normal case), a 2-character 12-bit checksum, or a 3-character 16-bit cyclic redundancy check (CRC). If the other Kermit program is not capable of type 2 or 3 checking methods, automatic fallback to type 1 will occur. The more secure type 2 and 3 block checks take essentially no more execution time than the simple 1 character checksum. SET BLOCK 3 is a stronger check than SET BLOCK 2. SET BLOCK 2 or 3 is recommended for use with long packets (see below), noisy communication lines, binary (8-bit data) files, and text files containing critical data (budgets, grades, etc).

SET COUNT

Syntax: `SET COUNT` *number*

Set the value of the script COUNT variable to be between 0 and 65535. COUNT is used with IF COUNT to construct counted loops in script TAKE files and macros. Each active TAKE file or macro uses a private version of COUNT. The default value is zero, and the SHOW SCRIPT command displays the current value (meaningful only when given within a TAKE file or macro).

SET DEBUG

Syntax: `SET DEBUG {PACKET, SESSION, ON, OFF}`

With DEBUG PACKET, Kermit will display the actual packets on your screen during file transfer. With the normal file transfer display, regular-length packets sent and received are displayed in fixed-size slots. The display of extended-length packets, however (see SET RECEIVE PACKET-LENGTH), tends to overlap. If this bothers you, then also SET DISPLAY SERIAL, or LOG the packets rather than displaying them.

With DEBUG SESSION, during terminal emulation (on the IBM PC, Rainbow, and a few others), control characters are displayed in uparrow (‘‘^’’) notation and characters with the 8th bit set are preceded by the tilde (‘‘~’’) sign, and your session log (if any) will record 8-bit bytes, rather than 7-bit ASCII, regardless of SET DISPLAY or SET PARITY. Character translation (SET TRANSLATION INPUT) is not done during session debugging. The effect of SET DEBUG SESSION during terminal connection can be disconcerting, but it gives you a convenient line monitor equivalent to a specialized device that costs several thousand dollars, and it can prove very handy for tracking down data communication problems.

SET DEBUG ON turns on both SESSION and PACKET debugging, and SET DEBUG OFF turns them both off.

SET DEFAULT-DISK

Syntax: `SET DEFAULT-DISK` *x*`:`[*directory*]

Specify the default disk drive to use for file transfer, directory listings, and so forth. Equivalent to typing the DOS command for changing disks (`A:`, `B:`, etc). Affects Kermit and all inferior processes, but when you exit from Kermit, you will still have the same default disk as when you entered. As a convenience, a directory may be specified with or without the drive to change one or the other or both. This command is a synonym for CWD (CD).

SET DELAY

Syntax: `SET DELAY` *number*

Wait the specified number of seconds before starting a file transfer. Intended for use when the other side needs appreciable time to become ready, such as rearranging cables, changing programs, etc., or when MS-DOS Kermit is the remote Kermit (e.g. after CTTY COM1, SET REMOTE ON). The *number* is 0 to 63 seconds, normally 0.

SET DESTINATION

Syntax: `SET DESTINATION {DISK, PRINTER, SCREEN}`

SET DESTINATION PRINTER will cause incoming files to be sent directly to the printer; SCREEN will send output normally destined for the disk to the screen. The normal destination is DISK. SET DESTINATION affects only files transferred with SEND, GET, or RECEIVE; it cannot be used to reroute the output from REMOTE server commands.

SET DISPLAY

Syntax: `SET DISPLAY {QUIET, REGULAR, SERIAL, 7-BIT, 8-BIT}`

During file transfer, MS-DOS Kermit's regular display is a formatted screen whose fields are randomly updated with file names, packet numbers, error counts, percent done, error messages, and so forth, as shown in Figure 5-1.

If you wish to run Kermit-MS interactively through the back port, for instance after the operator has done CTTY COM1, you must give the command SET REMOTE ON (which, currently at least, is equivalent to SET DISPLAY QUIET); this suppresses the file transfer display screen, so that the display won't interfere with the file transfer itself. You can also use this command to suppress the display in local mode, in case you are using a system that allows you to do other work while file transfer proceeds in the background.

If you have your PC connected to a speaking device (a common practice for visually impaired people), or you are logging the display screen to a printer (using DOS `^P` or `kermit > prn`), the random nature of the regular display will make the results of little use. SET DISPLAY SERIAL is provided for this purpose; it causes the program to report progress "serially" on the screen. In serial mode, error messages are preceeded with the word "Error" and repeat messages with the word "Retry". Packets are numbered as dots with every tenth being a plus sign. The packet display is automatically broken across lines at every 70th packet. The serial display makes much more sense when spoken than does the regular display.

The serial display does not show the percent and kilobytes transferred. It is the default display style for generic MS-DOS Kermit; REGULAR is the default for all others.

The last two parameters, 7-BIT and 8-BIT, control the size of characters sent to the screen during terminal emulation. 7-BIT is the default and includes all ASCII characters. 8-BIT is useful with national and line drawing characters.

SET DUMP

Syntax: `SET DUMP` *filespec*

On those systems that support this feature, change the file or device name of the screen dump file. The normal file name is `KERMIT.SCN`. See the section on terminal emulation for details about screen dumps. If the specified file already exists then new material is appended to old. If you want to start a new screen dump file, delete the old one first.

SET END-OF-LINE

Syntax: `SET END-OF-LINE` *number*

If the remote system needs packets to be terminated by anything other than carriage return, specify the decimal value, 0-31, of the desired ASCII character. Equivalent to SET SEND END-OF-LINE (SET END-OF-LINE is kept only for historical reasons, and the parameter really should be called END-OF-PACKET anyway.)

SET EOF

Syntax: `SET EOF {CTRL-Z, NOCTRL-Z}`

Controls how the end of file is handled. CTRL-Z specifies a Control-Z character should be appended to the end of an incoming file. Certain MS-DOS text editors and other applications require files to be in this format. For outbound files, treat the first Control-Z as the end of the local file, and do not send it or any subsequent characters. NOCTRL-Z is the default; incoming files are stored, and MS-DOS files are sent, exactly as is, in their entirety. Use SHOW FILE to see the current SET EOF status.

SET ERRORLEVEL

Syntax: `SET ERRORLEVEL` *number*

Forces the DOS "errorlevel" variable to a given value. This is used mainly in scripts when other controls or tests determine that the cumulative errorlevel reported to DOS Batch when Kermit exits needs to be modified. The number can be 0 to 255 decimal.

SET ESCAPE

Syntax: `SET ESCAPE` *character*

Specify the control character you want to use to "escape" from remote connections back to Kermit-MS. On most systems the default is ‘‘`^]`’’ (Control-Rightbracket), which was chosen because it is a character you would otherwise rarely type.

The *character* is entered literally after SET ESCAPE or in backslash number form (`\29`), and should be chosen from the ASCII control range. It is not possible to use non-ASCII characters (like function keys) for this purpose (but see SET KEY for a way around this restriction).

SET FLOW-CONTROL

Syntax: `SET FLOW-CONTROL {XON/XOFF, NONE}`

Specify the full duplex flow control to be done on the currently selected port. The options are XON/XOFF and NONE. The specified type of flow control will be done during both terminal emulation and file transfer. By default, XON/XOFF flow control is selected. XON/XOFF should not be used on half-duplex (local echo) connections, or when the other system does not support it. If XON/XOFF is used, HANDSHAKE should be set to NONE.

SET HANDSHAKE

Syntax: `SET HANDSHAKE {CODE` *number*`, BELL, CR, LF, NONE, XOFF, XON}`

Specify any half-duplex line turnaround handshake character to be used during file transfer on the currently selected port. The CODE *number* form allows any ASCII character to be specified by its decimal ASCII code. Handshake is NONE by default; if set to other than NONE, then FLOW-CONTROL should be set to NONE. In operation the handshake character is sought at the end of each received packet, following the normal END-OF-LINE character, but is not sent for outgoing packets.

SET INCOMPLETE

Syntax: `SET INCOMPLETE {DISCARD, KEEP}`

Specifies what to do with files that arrive incompletely: discard them or keep them. They are normally discarded.

SET INPUT

Syntax: `SET INPUT {CASE, DEFAULT-TIMEOUT, ECHO, TIMEOUT-ACTION}`

This command is described in Section 5.8, SCRIPTS.

SET KEY

Syntax: `SET KEY` *key-specifier* `[`*key-definition*`]`
Also: `SET KEY` {ON, OFF, CLEAR}

> *WARNING:* The format and functions of this command have changed substantially since version `2.29B` and earlier. The changes were made in order to allow key redefinition to work on a wider variety of systems and keyboards without customization of the program source code for each configuration. See section 5.11 for further details.

Typical uses of SET KEY:

- You're used to having the ESC key in the upper left corner of the keyboard, but your new PC keyboard has an accent grave (‘‘`’’) there. You can use SET KEY to make the accent key transmit an ESC, and you can assign accent grave to some other key.

- You send a lot of electronic mail, and always sign it the same way. You can put your "signature" on a single key to save yourself a lot of repetitive typing.

- You must set up your PC's function keys or numeric keypad to work properly with a host application.

- You have trouble with Kermit's 2-character escape sequences (like Ctrl-`]` C), and you want to assign these functions to single keys, like F10.

The SET KEY command does these things and more, and SHOW KEY gives us assistance. A key can be defined to:

- send a single character other than what it would normally send,
- send a string of multiple characters,
- invoke a CONNECT-mode Kermit action verb,
- send itself again.

SET KEY specifies that when the designated key is struck during terminal emulation, the specified character or string is sent or the specified Kermit action verb is performed. Key definitions operate only during CONNECT, not at `Kermit-MS>` or DOS command level.

The key-specifier is the identification of the key expressed in system-dependent terms. This can be a letter, such as Q for the key which produces an uppercase Q, or the numeric ASCII value of the letter in backslash notation (e.g.

‘‘\81’’), or else the numerical "scan code" observed by the system when the key is pressed (e.g. "\3856" for Ctrl-Alt-Shift-Q on an IBM PC). Material printed on keycaps is not necessarily a guide to what the key-specifier should be. When the word CLEAR is used in place of a key-specifier, all key definitions are cleared and then any built-in definitions are restored.

A string definition is one or more characters, including 8-bit values expressed in backslash form, such as

`SET KEY \315 directory\13`	IBM F1 key sends ‘‘`directory<cr>`’’
`SET KEY S X`	S key sends upper case X (a mean trick)
`SET KEY T \27[m`	T key sends three bytes: `ESC [ m`
`SET KEY \23336 {del }xxx`	Alt-D sends "`del `"
`SET KEY \324 \Kexit`	F10 escapes back to `Kermit-MS>` prompt.

The string begins with the first non-spacing character following the key identification and continues until the end of line, exclusive of any trailing spaces. If a semicolon comment is used and the definition is given in a TAKE file, the line ends at the last non-spacing character before the semicolon. Curly braces, `{...}`, can be use to delimit the string in case you want the definition to include trailing spaces. All text after the closing bracket is ignored.

This manual does not contain a list of all the scan codes for all the keys on all the keyboards on all the PCs supported by MS-Kermit -- that would be a manual in itself. Rather, in order to obtain the key-specifier for the SET KEY command, you must type a SHOW KEY command and then press the desired key or key combination. This will report a scan code that you can use as the key specifier in a SET KEY command. To do this for many keys is a laborious process, so you should collect all your SET KEY commands into a file, which you can TAKE, or put them in your `MSKERMIT.INI` file.

If you enter SET KEY by itself, with no key specifier, the command will prompt you to press the selected key and again for the definition string. Certain characters, like ESC and CR, may not be entered literally into the string, but can be included by inserting escape codes of the form `\nnn`, a backslash followed by a 1- to 4-digit number corresponding to the ASCII value of the desired character. Where an ASCII digit follows directly after a backslash number, confusion can be avoided by placing curly braces `{}` around the backslashed number; thus, `\{27}5` represents the two ASCII characters ESC and 5.

Here is an example of the use of SET KEY to assign ESC (ASCII 27) to the accent grave key. First the user gets the key-specifier for the key:

```
Kermit-MS>show key
  Push key to be shown (? shows all): `
  ASCII char: ` \96 decimal is defined as
  Self, no translation.
Free space: 129 key and 100 string definitions, 837 string characters.
```

The free space report says that 129 more keys may be redefined, and up to 100 of them may have multi-character strings assigned to them (as opposed to single characters), and that there are 837 bytes left for these strings, in total. Confident that there is enough space left for a new key definition, the user proceeds:

```
Kermit-MS>set key
  Push key to be defined: `
  Enter new definition: \27
```

Once a key definition is constructed and tested, it may be entered on a single line in a command file (such as `MSKERMIT.INI`):

```
set key \96 \27
```

To prevent accidents, SET KEY shows the current definition before asking for a new one; enter a Control-C to keep the current definition, or a carriage return to undefine the key, or a query mark `(?)` to see available choices.

The keyboard can be restored to its startup state, that is all redefinitions removed and all built-in defitions restored, by using the keyword CLEAR in place of the key identification:

```
SET KEY CLEAR
```

Undefined keys which do not send ASCII characters are trapped by the keyboard translator and are rejected; a beep results from using an undefined non-ASCII key.

SET KEY OFF directs MS-Kermit to read keycodes from DOS, rather than BIOS, so that console drivers like ANSI.SYS that operate at the DOS level may be used during Kermit CONNECT sessions. This would also apply to any special keyboard replacements that come with DOS-level drivers. SET KEY ON turns key definition back on, and returns Kermit to processing keystrokes at the BIOS level.

Kermit Action Verbs

An action verb is the shorthand expression for a named Kermit procedure, such as "generate the proper sequence for a left arrow," "show status," "send a BREAK," and others; verbs are complex actions and each verb has a name. In a key definition the verb name is preceeded by backslash K (\K) to avoid being confused with a string. Verbs and strings cannot be used together on a key.

```
SET KEY \331 \Klfarr
SET KEY \2349 \Kexit
```

makes the IBM keyboard left arrow key execute the verb named lfarr which sends the proper escape sequence for a VT102 left arrow key (which changes depending on the internal state of the VT102). The leading \K identifies the definition as a Kermit verb, so no string can start as \K or as \{K in upper or lower case (use \92K). The second example has Alt-X invoking the Leave-Connect-Mode verb "exit" (same as Kermit escape character ‘‘^]’’ followed by C).

Each system has its own list of verbs and predefined keys. Table 5-6 shows those available for the IBM PC family (there are also some additional verbs for reassigning Heath or VT100 function keys, see section 5.16.2). The SET KEY command shows the list of available verbs when a query mark (?) is given as a definition. SHOW KEY displays all currently defined keys or individually selected ones; SHOW KEY can be executed only interactively.

Some systems have preset key definitions when Kermit first begins (those for the IBM PC are shown in section 5.16.2). You can find out what they are on your system by typing SHOW KEY, and then question mark on the next line. You may supplement or change the predefined keys with SET KEY commands typed interactively or in MSKERMIT.INI or other command files.

The MS-Kermit CONNECT command may be used in conjunction with certain console drivers that do their own key redefinitions. Since MS-Kermit intercepts keystrokes at the BIOS level, drivers like ANSI.SYS which work at the DOS level will have no effect during CONNECT, even though they work at MS-Kermit command level. Other drivers, like SuperKey and ProKey, work at the BIOS level, and their key assignments will remain effective during Kermit terminal sessions, and additional Kermit SET KEY assignments may be made "on top" of them.

SET LOCAL-ECHO

Syntax: SET LOCAL-ECHO {ON, OFF}

Specify how characters are echoed during terminal emulation on the currently selected port. ON specifies that characters are to be echoed by Kermit-MS (because neither the remote computer nor the communications circuitry has been requested to echo), and is appropriate for half-duplex connections. LOCAL-ECHO is OFF by default, for full-duplex, remote echo operation.

Verb	Meaning
\Kupscn	Roll up (back) to previous screen
\Kdnscn	Roll down (forward) to next screen
\Khomscn	Roll up to top of screen memory
\Kendscn	Roll down to end of screen memory (current position)
\Kupone	Roll screen up one line
\Kdnone	Roll screen down one line
\Kprtscn	Print the current screen
\Kdump	Append the current screen to dump file
\Kholdscrn	Toggle hold screen mode
\Klogoff	Turn off session logging
\Klogon	Turn on session logging
\Ktermtype	Toggle terminal type
\Kreset	Reset terminal emulator to initial state
\Kmodeline	Toggle modeline off/on
\Kbreak	Send a BREAK signal
\Klbreak	Send a "long BREAK" signal
\Khangup	Drop DTR so modem will hang up phone
\Knull	Send a null (ASCII 0)
\Kdos	"Push" to DOS
\Khelp	Display CONNECT help message
\Kstatus	Display STATUS message
\Kexit	Escape back from CONNECT mode
\Kgold,\Kpf1	VT102 keypad function key PF1
\Kpf2..\Kpf4	VT102 keypad function keys
\Kkp0..\Kkp9	VT102 keypad numeric keys
\Kkpdot,\Kkpminus,\Kkpcoma,\Kkpenter	Other VT102 keypad keys
\Kuparr,\Kdnarr,\Klfarr,\Krtarr	VT102 cursor (arrow) keys

Table 5-6: Kermit-MS Verbs for the IBM PC Family

SET MODE-LINE

Syntax: `SET MODE-LINE {ON, OFF}`

On systems, like the IBM PC family, which are capable of displaying a status, or "mode" line on the 25th (or bottom) line during terminal connection, disable or enable this function. This command has no effect on systems that do not display a mode line during connect.

The mode line shows several important facts about the connection, like which port is being used, the transmission speed and parity, the current escape character, etc. When the mode line is enabled, it may be turned on and off using the CONNECT escape-level command M or the Kermit verb "modeline".

The mode line occupies the 25th line of those systems that have such a thing, and is not affected by scrolling (on some systems that have large screens, the mode line should appear on whatever the bottom line is, e.g. the 43rd). When emulating a VT102 or Heath-19, Kermit will allow the host to address the 25th line directly using cursor positioning commands. If this happens, Kermit will remove its mode line and relinquish control of the 25th line to the host (as if you had typed SET MODE OFF). When the Tektronix, or no terminal at all, is being emulated, the 25th line (if any) is available for scrolling. If the mode line is disabled by an application or by the command SET MODE OFF then the only way to revive Kermit's mode line display is to give the command SET MODE ON.

SET PARITY

Syntax: `SET PARITY {EVEN, ODD, MARK, SPACE, NONE}`

Specify the character parity to be used on the currently selected port. You will need to SET PARITY to ODD, EVEN, MARK, or possibly SPACE when communicating with a system, or over a network, or through modems, concentrators, multiplexers, or front ends that require or impose character parity on the communication line. For instance, most IBM mainframe computers use EVEN or MARK parity; Telenet normally uses MARK parity. If you neglect to SET PARITY when the communications equipment requires it, the symptom may be that terminal emulation works (well or maybe only partially), but file transfer or script INPUT commands do not work at all.

NONE means that no parity processing is done, and the 8th bit of each character can be used for data when transmitting binary files. This is the normal case. If parity is other than none, then there will be 7 data bits (use of parity with 8 data bits is not supported).

If you have set parity to ODD, EVEN, MARK, or SPACE, then Kermit-MS will request that binary files be transferred using 8th-bit-prefixing. If the other Kermit knows how to do 8th-bit-prefixing (this is an optional feature of the Kermit protocol, and some implementations of Kermit don't have it), then 8-bit binary files can be transmitted successfully. If NONE is specified, 8th-bit-prefixing will not be requested. Note that there is no advantage to using parity. It reduces Kermit's file transfer efficiency without providing additional error detection. The SET PARITY command is provided only to allow Kermit to adapt to conditions where parity is required, or 8-bit transmission is otherwise thwarted.

If parity is in use, then the display during terminal emulation, as well as any session log, will be 7-bit ASCII, unless you have SET DEBUG ON (q.v.).

There may be situations in which you require 7-bit ASCII with no parity during terminal emulation, but still want to force 8th bit prefixing during file transfer. To accomplish this, SET PARITY SPACE.

The INPUT and TRANSMIT commands use 7 or 8 bits if parity is NONE, according to the SET DISPLAY command, and this may upset recognition of received characters when the host unexpectedly sends them with its own parity.

SET PORT

Syntax: `SET PORT {`*number*`, COM`*n*`, BIOS`*n*`, NET [`*nodename*`], UB-NET1 [`*nodename*`]}`

On machines with more than one communications port, select the port to use for file transfer and CONNECT. This command lets you use a different asynchronous adapter, or switch between two or more simultaneous remote sessions. Subsequent SET SPEED, PARITY, HANDSHAKE, FLOW, and LOCAL-ECHO commands will apply to this port only -- each port remembers its own parameters, so that you may set them for each port and then switch between ports conveniently with the SET PORT command.

SET PORT 1 selects COM1, SET PORT 2 selects COM2. All versions default to port 1, except for the IBM PCjr, which uses port 2 if its internal modem is installed. Additionally, COM3 and COM4 are supported for IBM PC/AT's and PS/2's, as explained in Section 5.17.3.

SET PORT BIOS*n*, on machines which support it, instructs Kermit to do serial port input and output by Bios calls rather than going directly to the hardware (*n* is a digit between 1 and 4). The most important use is allowing selected network packages to intercept such Bios calls and relay the characters across the network.

In "generic" MS-DOS Kermit, the following alternate forms allow you to experiment with device names or numbers until you find the communication port:

```
SET PORT {DEVICE, FILE-HANDLE}
```

Just type a carriage return after either of these commands, and you will be prompted for a device name or a numeric port-handle. Keep trying till you find one that works. File-handle 3, the system auxillary device, is conventional on many machines, as are device names COM1, COM2, and AUX.

MS-Kermit for the IBM PC family is able to operate over local area networks through the NetBIOS interface. The command

    SET PORT NET [*nodename*]

redirects communications the LAN board installed in the local computer and the associated NetBIOS emulator software, if active, rather than the serial port or the COM device driver. It installs a unique Kermit node name in the local LAN, so that other nodes can refer to it when files are transferred or terminal emulation is done. This name is displayed when you give the SET PORT NET command. The server should use SET PORT NET, and the client should use SET PORT NAME *nodename*, specifying the server's name, e.g. `mskermit.K`. Note that alphabetic case is significant in node names!

Both the regular serial port and a network connection can be kept alive simultaneously; clearly, only one can be used at a time under MS-DOS. MS-DOS 3.x is not required for Kermit network usage, but most LANS do need DOS 3.1 or later for conventional file server work. Kermit needs only the NetBIOS emulator network software.

SET PORT UB-NET1 is implemented on the IBM PC version of Kermit to allow connection to Ungermann-Bass Net One LAN NETCI interface and behaves similarly to the NetBIOS method.

SET PROMPT

Syntax: `SET PROMPT [`*string*`]`

This command allows you to change the MS-DOS Kermit program's prompt. The string may be enclosed in curly braces. Control characters like ESC can be included as backslashed numbers like ''`\27`''. `ANSI.SYS` and similar console drivers can be programmed through this command to get a boldface, inverse, and/or blinking prompt. The prompt string must be less than 128 characters. If the string is omitted (missing) Kermit's original prompt of ''`Kermit-MS>`'' is restored.

SET RECEIVE

Syntax: `SET RECEIVE` *parameter value*

This command lets you modify the ways in which MS-Kermit asks the other Kermit to behave. That is, it controls the file transfer protocol options for packets sent to MS-Kermit by the other Kermit. The parameters and values you specify in the SET RECEIVE command are sent to the other Kermit during initial negotiations. Numbers may be specified as ordinary decimal numbers (74), or in backslash notation (\x03F).

`END-OF-LINE` *number*
: The ASCII value of terminating character to look for on incoming packets. Normally carriage return. Use this command if the other Kermit is terminating its packets with some other control character.

`PACKET-LENGTH` *number*
: Ask the remote Kermit to use the specified maximum length for packets that it sends to Kermit-MS. The normal length is 94 bytes. Use this command to shorten packets if the communication line is noisy or terminal buffers somewhere along the path are too small. Shorter packets decrease the probability that a particular packet will be corrupted, and will reduce the retransmission overhead when corruption occurs, but will increase the file transfer throughput.

  If a length greater than 94 is specified, a protocol option called "long packets" will be used, provided the other Kermit also supports it. Kermit-MS can receive extended-length packets up to 1000 bytes long. Long Packets can improve efficiency by reducing the per-packet overhead for a file, but they will not be used unless you issue this command. Before using this option, ensure that the equipment on the communications pathway can absorb a long packet, and that the connection is clean (retransmission of long packets is expensive!). You should also SET BLOCK-CHECK 2 or 3 for more reliable error checking.

`PADCHAR` *number*
: Ask the remote Kermit to use the given control character (expressed as a decimal number 0-31, or 127) for interpacket padding. Kermit-MS should never require any padding.

`PADDING` *number*
: Ask the remote Kermit to insert the given number of padding characters before each packet it sends. MS-Kermit never needs padding, but this mechanism might be required to keep some intervening communication equipment happy.

`START-OF-PACKET` *number*
: If the remote Kermit will be marking the beginning of packets with a control character other than Control-A, use this command to tell Kermit-MS about it (the number should be the decimal ASCII value of a control character). This will be necessary only if the hosts or communication equipment involved cannot pass a Control-A through as data, or if some piece of communication equipment is echoing packets back at you.

`TIMEOUT` *number*
: Ask the remote Kermit to time out and retransmit after the given number of seconds if a packet expected from Kermit-MS has not arrived. Use this command to change the other Kermit's normal timeout interval.

SET REMOTE

Syntax: `SET REMOTE {ON, OFF}`

SET REMOTE ON removes the file transfer display (as if you had given the command SET DISPLAY QUIET). It should be used when you are running Kermit-MS in remote mode when coming in from another PC through the Kermit-MS's "back port", to which the console has been reassigned using the DOS CTTY command, e.g.

```
CTTY COM1
```

It is necessary to issue the SET REMOTE ON command because (a) Kermit-MS has no way of knowing that its console has been redirected, and (b) when the console is the same as the port, the file transfer display will interfere with the file transfer itself. SET REMOTE OFF returns the file transfer display to its preferred style (REGULAR or SERIAL). When you SET REMOTE ON, you might also want to SET DELAY 5 or thereabouts, to allow yourself time to escape back to the local system before MS-Kermit starts sending packets.

On the IBM PC, CTTY CON returns control to the normal keyboard and screen (other systems may use other device names). See section 5.17.4 for further details about remote operation.

WARNING: During CTTY console redirection, many programs still output to the real screen and require input from the real keyboard and will hang the system until keyboard requests are satisfied.

SET RETRY

Syntax: `SET RETRY` *number*

Sets the number of times a packet is retransmitted before the protocol gives up. The number of retries can be between 1 and 63, and is 5 by default. This is an especially useful parameter when the communications line is noisy or the remote host is very busy. The initial packet of a file exchange is given three times as many retries to allow both systems to become ready.

SET SEND

Syntax: `SET SEND` *parameter value*

The SET SEND command is used primarily to override negotiated protocol options, or to establish them before they are negotiated.

`END-OF-LINE` *number*
ASCII value of packet terminator to put on outbound packets. Normally carriage return. Use this command if the other Kermit needs its packets terminated with a nonstandard control character.

`PACKET-LENGTH` *number*
Use this as the maximum length for outbound packets, regardless of what the other Kermit asks for. Normally, you would use this command only to send shorter packets than the other Kermit requests, because you know something the other Kermit doesn't know, e.g. there's a device on the communication path with small buffers.

`PADCHAR` *number*
Use the specified control character for interpacket padding. Some hosts may require some padding characters (normally NUL or DEL) before a packet, and certain front ends or other communication equipment may need certain control characters to put them in the right modes. The number is the ASCII decimal value of the padding character, (0 - 31, or 127).

`PADDING` *number*
How many copies of the pad character to send before each packet, normally zero.

`PAUSE` *number*
How many milliseconds to pause before sending each packet, 0-127, normally zero. This may help half-duplex or slow systems prepare for reception of our packet. Padding characters are sent only after the time limit expires.

`QUOTE` *number*
Use the indicated printable character for prefixing (quoting) control characters and other prefix characters. The only reason to change this would be for sending a very long file that contains very many ''#'' characters (the normal control prefix) as data.

`START-OF-PACKET` *number*
Mark the beginning of outbound packets with some control character other than Control-A. This will be necessary if the remote host or the communication channel cannot accept a Control-A as data, or if it echoes back your packets. The remote host must have been given the corresponding SET RECEIVE START-OF-PACKET command.

`TIMEOUT` *number*
Change Kermit-MS's normal timeout interval; this command is effective only if TIMER is set to be ON; it is normally ON, with a default interval of 13 seconds.

SET SPEED

Syntax: `SET SPEED` *rate*

Set the transmission speed (in bits per second, commonly called *baud*) of the currently selected terminal communications port to 300, 1200, 1800, 2400, 4800, 9600, or other common speed, and on the IBM PC family, higher speeds including 19200, 38400, 57600, and 115200. Both connected systems, as well as any intervening communication equipment, must be able to support the specified transmission speed, and both systems should be set to the same speed.

Some implementations do not support the SET SPEED command. But Kermit-MS leaves the current communication port settings alone unless you issue explicit SET commands to change them, so you may use MODE or other DOS programs to establish the desired settings before running Kermit.

On certain systems, when you first run Kermit after powering the system up, you may get a message "Unrecognized baud rate". This means that Kermit tried to read the baud rate from the port and none was set. Simply use SET SPEED (if available) or the DOS MODE command to set the desired baud rate.

SET BAUD is a synonym for SET SPEED.

SET TAKE-ECHO

Syntax: `SET TAKE-ECHO {ON, OFF}`

Specifies whether screen display should occur during implicit or explicit TAKE operations on `MSKERMIT.INI` or other Kermit-MS command files, and during evaluation of macro definitions by the DO command. Handy for finding errors in TAKE files or macro definitions.

SET TERMINAL

Syntax: `SET TERMINAL {`*type*`,` *parameter* `[`*value*`]}`

This command controls most aspects of terminal emulation. Most of the parameters are only settable (or meaningful) on the IBM PC family and compatibles. (Programmers who are proficient on other MS-DOS systems are invited to fill in these functions for those systems and send the results back to Columbia.) On other systems, built-in setup modes or DOS commands can be used to accomplish the same functions.

The first group of parameters tells which kind of terminal to emulate. When Kermit-MS uses its built-in software for emulation, incoming characters are examined for screen control commands (escape sequences) specific to that terminal, and if encountered, the commands are executed on the PC screen.

`NONE` Act as a dumb terminal. All incoming characters will be sent to the screen "bare", as-is, through DOS. If you have loaded a device driver into DOS for the `CON` device, such as `ANSI.SYS`, then that driver will be able to interpret the codes itself. Many non-IBM systems have their own screen control code interpreter built into DOS or firmware, or available as a loadable device driver.

`VT52` The DEC VT-52 terminal.

`HEATH` The Heath/Zenith-19 terminal (H19), which supports all the VT52 commands, plus line and character insert/delete editing functions, an ANSI mode, and a 25th line.

`VT102` The DEC VT102 (ANSI) terminal, which is the same as the VT100 but also supports line/character insert/delete editing functions and ANSI printer controls.

`TEK4010`
A Tektronix 4010 graphics terminal. Currently only available on IBM, TI, and Victor PCs. On the IBM family, Kermit automatically senses and adapts to the CGA, EGA, Monochrome, Hercules, or ATT style board.

On the IBM family, you may "toggle" among the supported terminal emulations by typing Alt-Minus.

The specific escape sequences supported by Kermit for each of these terminal types are listed in section 5.16.1. Note that when a Kermit program includes Tektronix emulation, this can be invoked automatically while in character mode (VT102, VT52, or Heath emulation) when the emulator receives certain escape sequences. This can be turned off using the DISABLE TEK command.

The remaining SET TERMINAL commands specify setup options for the selected terminal:

`CHARACTER-SET {UK, US}`
UK displays ‘‘#’’ (ASCII 35, number sign) as a pound sterling sign, US displays ‘‘#’’ as ‘‘#’’. This command applies during VT100/102 emulation.

`COLOR` *number* `[,` *number* `[,` *number*`]]`
Several numbers, applied in left to right sequence, separated by commas or spaces:

- 0 Reset the colors to normal intensity white characters on a black background and use the "no-snow" mode on the IBM Color Graphics Adapter (CGA).
- 1 High intensity foreground
- 10 Request fast screen updating for use on the IBM Mono, EGA, or VGA (usually sensed and set internally by Kermit), and some non-IBM CGAs.
- 3*x* Foreground color

4*x* Background color

where *x* is a single digit from 0 to 7, which is the sum of the desired colors:

1 Red
2 Green
4 Blue

Example: "SET TERMINAL COLOR 0 1 37 44" on an IBM CGA would produce bold white characters on a blue field with no snow. The snow removal business has to do with whether the program should synchronize with vertical retrace when updating screen memory. This is necessary with certain color adaptors (like the CGA) and unnecessary for others (like the EGA).

`CURSOR-STYLE {BLOCK, UNDERLINE}`
Sets the cursor rendition to your preference. Note that on some early IBM PCs and compatibles, the cursor may not be restored correctly after escaping back from CONNECT because of a bug in the early IBM BIOS.

`KEYCLICK {ON, OFF}`
Turns electronic keyclick ON or OFF. If your keyboard has a mechanical clicker (as IBM boards do), you may not notice the effect of this command.

`GRAPHICS {AUTO-SENSING, CGA, EGA, VGA, HERCULES, ATT}`
Manually selects the kind of display adapter for Tektronix graphics. AUTO-SENSING is the default, VGA means 640x480x16 colors, and ATT encompasses the ATT 6300 series, Olivetti M24/M28, DEC VAXmate II, and the Toshiba T3100 in 640x400 b/w (see Table 5-5).

`MARGIN-BELL {ON, OFF}`
Controls whether the bell should be sounded when the cursor passes column 72 near the right screen margin; wider displays set the bell 8 columns from the right edge.

`NEWLINE-MODE {ON, OFF}`
ON sends a carriage-return-linefeed combination (CRLF) when you type carriage return (CR) during terminal emulation. OFF (default) just sends a CR when you type CR. Useful in conjunction with SET LOCAL-ECHO ON when CONNECTing two PC's back-to-back.

`ROLL {ON, OFF}`
ON unrolls the screen to the bottom before adding new material if the screen had been rolled back, e.g. by Ctrl-PgUp. ROLL OFF (the default) displays new material on the current screen, possibly overwriting old material.

`SCREEN-BACKGROUND {NORMAL, REVERSE}`
NORMAL means dark background, light characters. REVERSE means light background, dark characters.

`TAB {AT` *n*`, CLEAR AT` *n*`, CLEAR ALL}`
Sets tab stops or clears one or all tab stops; *n* is the numeric position of the tab to be set or cleared. By default, tabs are every 8 spaces, at positions 9, 17, 25, etc. Only meaningful when emulating a terminal that has settable tabs (the VT52 doesn't really but the emulator can set them anyway). More than one tabstop may be specified by separating column numbers with commas, spaces, or tabs. 132 columns are supported.

`WRAP {ON, OFF}`
ON automatically breaks screen lines (by inserting a CRLF) when they reach the right margin. OFF disables wrapping -- if a line is too long, the excess characters go off the screen. WRAP is OFF by default, since most hosts format lines to fit on your screen.

SET TIMER

Syntax: `SET TIMER {ON, OFF}`

This command enables or disables the timer that is used during file transfer to break deadlocks that occur when expected packets do not arrive. By default, the timer is ON. If the other Kermit is providing timeouts, you can safely turn the timer OFF to avoid unnecessary retransmissions that occur when two timers go off simultaneously.

SET TRANSLATION

Syntax: `SET TRANSLATION INPUT {ON, OFF,` *char1 char2*`}`

This command provides multi-language support (and perhaps other special effects) during CONNECT, and during execution of the INPUT, OUTPUT, PAUSE, and TRANSMIT script commands, but not during file transfer or at MS-Kermit command level. A character that arrives at the communication port (*char1*) will be translated to another character (*char2*) before display on the screen. As many as 256 characters may have translations specified concurrently. But to see characters with ASCII values higher than 127, you must also SET DISPLAY 8 and SET PARITY NONE.

SET TRANSLATION INPUT ON enables translation (the keyword INPUT is required to allow future translation mechanisms). OFF disables the translation and is the default. So even if you have set up a translation table, you must SET TRANSLATION INPUT ON before it will take effect. SHOW TRANSLATION tells whether translation is OFF or ON, and displays any current table entries.

Translation table entries are made by specifying byte pairs in ASCII or numeric backslash form:

```
SET TRANS INPUT \3 \13
```

converts incoming ASCII ETX characters (decimal 3) to ASCII CR (decimal 13). 8-bit values are allowed, and refer to characters in the "upper half" of the PC's character set, either the ROM characters supplied with the PC or else substitutions provided by a special device driver.

A more practical example shows how the user of a German PC could use the SET TRANSLATION and SET KEY commands to make the PC's umlaut-a key (key code 132) send a left curly brace (‘‘`{`’’, ASCII 123), and to display incoming curly braces as umlaut-a's:

```
SET KEY \d132 \d123
SET TRANS INP { \d132
```

(This example applies to the IBM PC German keyboard, and assumes the German keyboard driver, KEYBGR, has been loaded. This is usually done in `AUTOEXEC.BAT`.)

SET WARNING

Syntax: `SET WARNING {ON, OFF}`

Specify what to do when an incoming file is about to be stored under the same name as an existing file in the target device and directory. If ON, Kermit will warn you when an incoming file has the same name as an existing file, and automatically rename the incoming file (as indicated in the warning message) so as not to destroy (overwrite) any existing one. If OFF, the pre-existing file is destroyed, even if the incoming file does not arrive completely. WARNING is ON by default as a safety measure, and the current setting may be observed in the SHOW FILE display.

The new name is formed by adding numbers to the part of the name before the dot. For instance, `ABC.TXT` becomes `ABC00001.TXT`, `ABC00001.TXT` becomes `ABC00002.TXT`, etc. If the name already has eight characters, then digits replace the rightmost characters.

5.6.11. The STATUS and SHOW Commands

The values of MS-Kermit options that can be SET, DEFINEd, ENABLEd, or DISABLEd can be displayed using the STATUS or SHOW commands.

The STATUS Command

Syntax: `STATUS`

The STATUS command displays the values of all the current SET options on a single screen. There are currently no operands for the STATUS command. Use the SHOW command to see logically-grouped settings, e.g. SHOW COMMUNICATIONS, SHOW TERMINAL.

The SHOW Command

Syntax: `SHOW` *option*

The SHOW command is used for displaying communication parameters, protocol settings, macro definitions, key redefinitions, file transfer statistics, translations, and other common groupings.

`SHOW COMMUNICATIONS`
: displays the settings of the current serial port (port, speed, parity, echo, etc) and the status of modem signals Carrier Detect, Data Set (modem) Ready, and Clear To Send.

`SHOW FILE`
: displays the file transfer control settings, such as the current path, file discard, attributes packets on/off, warning, end-of-file convention, etc.

`SHOW KEY`
: allows you to determine a key's identification code and what it will send in CONNECT mode, most useful for obtaining the identification of a key when SET KEY commands will be placed in a TAKE file. This command can be done only interactively (use a `?` to see all defined keys). Refer to the SET KEY description for details.

`SHOW LOGGING`
: Displays the names of the session, packet, and transaction logs, and tells whether logging is in effect.

`SHOW MACROS [macroname]`
: displays the definitions of all currently defined macros, as well as the amount of space left for new macro definitions. A macro name, or abbreviation, can be included to restrict the list, e.g. SHOW MACRO IBM will display the definition of the IBM macro, and SHOW MACRO X will list the definitions of all macros whose names begin with X.

`SHOW MODEM`
: displays the status of the modem signals DSR (dataset ready, modem tells the PC that it is turned on and in data mode), CTS (clear to send, modem grants the PC permission to send data), and CD (carrier detect, local modem tells the PC that it is connected to the remote modem). The results may be misleading if your asynchronous adapter, or the connector or cable that is attached to it, is strapped to supply these modem signals itself.

`SHOW PROTOCOL`
: displays the values of the Kermit protocol-related parameters, including all the SET SEND and SET RECEIVE parameters, plus whether the timer, attribute packets, and logging are enabled.

`SHOW SCRIPTS`
: displays the script-related variables.

`SHOW SERVER`
: displays which server functions are enabled and disabled.

`SHOW STATISTICS`
: displays counts of characters sent and received during file transfers, for both the most recent transfer and the entire session, and an estimate of the average baud rate while sending and listening.

`SHOW TERMINAL`
: displays the terminal settings, which terminal is being emulated, the tab stops, etc.

`SHOW TRANSLATION`
: displays the entries in the 256 byte input translation table. Values are expressed numerically to avoid confusion with different display adapters, and the command shows only entries for which input and output codes differ.

5.7. Macros

Like TAKE files, macros provide a way of collecting many commands into a single command. The difference between a macro and a TAKE file is that Kermit keeps all its macro definitions in memory, and can execute them as many times as you like, without having to look them up on disk, whereas every time you issue a TAKE command, Kermit has to access a disk. But . . . you can have as many TAKE command files as you like, and they can be as long as you want, whereas MS-Kermit's memory for storing macro definitions is limited. You can put macro definitions and DO commands for them in TAKE files, or for that matter, you can put TAKE commands in macro definitions. There is a limit of 25 simultaneously active TAKE files plus active macros; a TAKE file or macro remains active if the last item invokes another TAKE or macro command. Active here means Kermit is reading commands from them, not just storing them for later.

The DEFINE Command

Syntax: `DEFINE` *macro-name* `[`*command* `[,` *command* `[, ...]]]`

Kermit-MS command macros are constructed with the DEFINE command. Any Kermit-MS commands may be included. Example:

```
define telenet set parity mark, set baud 1200, connect
```

A macro can be undefined by typing an empty DEFINE command for it, like

```
define telenet
```

A macro definition may be up to 255 character long. This example shows a long definition in which lines are continued with hyphenation:

```
define setup set port 1, set speed 19200, set parity even,-
set flow none, set handshake xon, set local-echo on,-
set timer on, set terminal color 1 31 45,-
set warning on, set incomplete keep, connect
```

Longer definitions can be accomplished by "chaining." Example (even though this one isn't really longer):

```
define setup set port 1, set speed 19200, set par even, do setup2
define setup2 set flo no, set handsh xon, set local on, do setup3
define setup3 set timer on, set terminal color 1 31 45, do setup4
define setup4 set warning on, set incomplete keep, connect
```

DO SETUP or just SETUP will invoke all of these commands. Commas are used to separate commands in macro definitions; carriage returns (`\13`) cannot be used. When control or other special characters are needed in a macro they may be expressed in backslash number form, `\`*nnn*.

The SHOW MACROS command displays the values of currently defined macros, and tells how much space is left for further definitions.

The DO Command

Syntax: `DO` *macro-name* `[`*parameters...*`]`

A Kermit-MS macro is invoked using the DO command. For instance, Kermit-MS comes with a predefined macro to allow convenient setup for IBM mainframe line-mode communications; to invoke it, you would type DO IBM. The IBM macro is defined as "set timer on, set local-echo on, set parity mark, handshake xon, set flow none". You can use the DEFINE command to redefine this macro or remove the definition altogether.

There is no automatic way to undo the effect of a macro. If you need to accomplish this effect, you should define another macro for that purpose. For instance, to undo the effect of "do ibm" so that you could connect to, say, a DEC VAX, you could:

```
def vax set parity none, set handshake none, set flow xon/xoff,-
```

```
set timer off, set local-echo off
```

Then you can "do ibm" whenever you want to use the IBM system, and "do vax" whenever you want to use the VAX.

If you wish to view the macro expansion whenever you issue a DO command, you can SET TAKE-ECHO ON.

As a convenience the word DO may be omitted. However, when question-mark help is sought at the Kermit prompt, only the main keyword help table will be shown. If you want to see the available macros, type "do ?" or SHOW MACROS. Use of DO is recommended for overall clarity unless a favorite macro is executed frequently.

Variables

Syntax: `\%x` *text*

Both TAKE files and Macros can use substitution variables similar to those of DOS Batch. The name of a substitution variable is of the form "\%character" where the single character is a digit or a letter or other 8-bit character whose ASCII value is 48 decimal or larger; upper and lower case letters are considered to be the same character. A substitution variable is defined as a string of text by the DEFINE command (the variables are in fact macros) and Kermit replaces occurrences of the variable name with that text, hence the word "substitution". For example,

```
Kermit-MS>define \%a this is substituted material
Kermit-MS>echo I wonder if \%a or not.
```

yields the display:

```
I wonder if this is substituted material or not.
```

Another example:

```
Kermit-MS>define \%c set port 1,set speed 9600,set parity even,connect
```

Then

```
Kermit-MS>\%c
```

is equivalent to

```
Kermit-MS>set port com1
Kermit-MS>set speed 9600
Kermit-MS>set parity even
Kermit-MS>connect
```

The special subset of substitution variables, `\%1 .. \%9`, is similar to the DOS Batch variable set `%1 .. %9`. The DO command can accept arguments after the macro name and the individual words in the arguments become the definitions of `\%1`, etc, for up to nine words, in order. For example, given the following definition:

```
def dial ATDT\%1\13,input 30 CONNECT,connect,in Login:,out \%2\13
```

the following command can be used to dial any phone number:

```
Kermit-MS>do dial 555-1212 myname
```

The word DO may be omitted, as in:

```
Kermit-MS>dial 555-1212 myname
```

This command automatically assigns the value "555-1212" to variable the `\%1` and "myname" to `\%2`, and uses these values while dialing the phone and logging into the host system. If fewer than nine words are seen the remaining variables are not changed. For example, if the line above was busy, you could dial a different number and omit the username because it will be remembered from last time.

Substitution variables can reference other substitution variables in their definitions. Care is needed to prevent circular definitions, but even those are detected by Kermit. Subtle circular executions could cause Kermit to go into an endless loop; if you think this is happening, type a Control-C to interrupt the process. To clarify matters, the definition string of a variable is substituted for the variable's name when the name is observed in a left to right scan of a command. For example,

```
Kermit-MS>define \%a echo This is \%b example: \%b.
Kermit-MS>define \%b a mac\%c expansion
Kermit-MS>define \%c ro string
Kermit-MS>\%a
```

displays:

```
This is a macro string expansion example: a macro string expansion.
```

If this example is entered manually then when the final `\%a` is typed the command line is immediately replaced with the fully expanded command and more input is solicited (such as a carriage return). Try it. Check the variable definitions with the SHOW MACRO command.

A variable can be undefined (deleted) by defining it as an empty string:

```
Kermit-MS>define \%c
```

Finally, macros may contain labels, GOTO label commands, IF statements, and other script control features discussed below the same as if the macro were a TAKE file. Macros may also invoke TAKE files, and vice versa.

5.8. SCRIPTS

A script is a file or a macro containing Kermit commands to be executed. What distinguishes a script from ordinary TAKE files or macros is the presence of INPUT, REINPUT, OUTPUT, PAUSE, ECHO, CLEAR, IF, GOTO, and WAIT commands to automatically detect and respond to information flowing though the serial port, actions which otherwise would be performed by the user during CONNECT. The login sequence of a host computer is a classical example.

It is a common, but incorrect, assumption that text to be sent to the remote computer can be included in a TAKE file after the CONNECT command:

```
set speed 9600        ; MS-Kermit command
connect               ; MS-Kermit command
run kermit            ; Text to be sent to other system
send foo.bar          ; Text to be sent to other system
^]c                   ; Escape sequence to get back to MS-Kermit
receive               ; MS-Kermit command
```

The reason this doesn't work is that during CONNECT, MS-Kermit always reads from the real keyboard, and not from the take file. Even if this technique did work, it would still run into synchronization problems. But these can be avoided when there is a way to coordinate the commands that we send with the remote system's responses. Kermit's script commands provide this ability. They may be freely intermixed in a TAKE file or macro with any other Kermit commands to achieve any desired effect. The OUTPUT command sends the specified characters as if the user had typed them; the INPUT command reads the responses and compares them with specified character strings, just as the user would do.

The script commands include INPUT, REINPUT, OUTPUT, PAUSE, WAIT, ECHO, IF, and GOTO. These commands may be interrupted by typing Ctrl-C at the keyboard. The INPUT, REINPUT, PAUSE, and WAIT commands accept a following number as a timeout value. The number is interpreted as seconds from the present or, if given in `hh:mm:ss` form, as a specific time of day. In either case, the timeout interval must be within 12 hours of the present to avoid it being considered as in the past (expired).

The CLEAR Command

Syntax: `CLEAR`

The CLEAR command empties the buffers of the serial port to forget any earlier material. This gets the INPUT command off to a clean start. (This command was called CLRINP in 2.29B and earlier, and CLEAR was used to erase macro and key definition memory).

The ECHO Command

Syntax: `ECHO` *text*

The ECHO command is useful for reporting progress of a script, or prompting the user for interactive input. The text is displayed on the screen, and may include backslash notation for control or 8-bit characters. An implied linefeed is included at the beginning of the text.

SET INPUT

Syntax: `SET INPUT {CASE, DEFAULT-TIMEOUT, ECHO, TIMEOUT-ACTION}`

The SET INPUT command controls the behavior of the script INPUT command:

`SET INPUT CASE {IGNORE, OBSERVE}`
: Says whether or not to distinguish upper and lower case letters when doing a matchup in the INPUT command. OBSERVE causes upper and lower case letters to be distinguished. The default is to IGNORE case distinctions.

`SET INPUT DEFAULT-TIMEOUT` *seconds*
: Changes the default waiting time from one second to this new value. The value is used when an INPUT command has no timeout specified.

`SET INPUT ECHO {ON, OFF}`
: Show on the screen characters read from the serial port during the script operation, or not. Default is ON, show them.

`SET INPUT TIMEOUT-ACTION {PROCEED, QUIT}`
: Determines whether or not the current macro or TAKE command file is to be continued or exited if a timeout occurs. PROCEED is the default and means that timeouts are ignored. QUIT causes the current script file to be exited and control passed to either the next higher level script file (if there is one) or to Kermit’s main prompt.

The SHOW SCRIPTS command displays the SET INPUT values.

The INPUT command

Syntax: `INPUT [`*timeout*`] {`*search-string*`, @`*filespec*`}`

INPUT is the most powerful of the script commands. It reads characters from the serial port continuously until one of two things occurs: the received characters match the search string or the time limit expires. Matching strings is the normal use, as in:

```
Kermit-MS>input 5 Login please:
```

to recognize the phrase "Login please:", or else time out after waiting for 5 seconds. A special binary character `\255` or `\o377` or `\xFF` stands for the combination carriage return and a line feed, in either order, to simplify pattern matching. The command reports a testable status of SUCCESS or FAILURE and sets the DOS ERRORLEVEL parameter to 2 if it fails to match within the timeout interval. Characters are stored in a 128 byte buffer for later examination by REINPUT, discussed below.

Beware of characters arriving with parity set because the pattern matching considers all 8 bits of a byte unless the local parity is other than NONE and SET DISPLAY is 7-BITS. Arriving characters are modified by first removing the parity bit, if parity is other than NONE, then they are passed through the SET TRANSLATION INPUT converter, the high bit is again suppressed if SET DISPLAY is 7-BITs, the result is logged and stored for pattern matching.

The REINPUT command

Syntax: `REINPUT [`*timeout*`] {`*search-string*`, @`*filespec*`}`

The REINPUT command is like INPUT except that characters are read from the 128 byte serial port history buffer rather than always seeking fresh input from the port. The purpose is to permit the current text to be examined several times, looking for different match strings. A common case is reading the results of a connection message from a modem which might be "CONNECT 1200" or "CONNECT 2400", depending on the other modem. If the history buffer has less than 128 bytes then fresh input may be requested while seeking a match, until the buffer is full. REINPUT match searches begin at the start of the buffer whereas INPUT searches never go back over examined characters. REINPUT sets the testable status of SUCCESS or FAILURE and DOS ERRORLEVEL, just as for INPUT.

The INPUT, REINPUT, and OUTPUT commands have a special syntax to replace the normal string with text obtained from a file or device:

```
OUTPUT @filespec
INPUT @filespec
```

Both forms read one line of text from the file or device and use it as the desired string. A common use is to wait for a password prompt and then read the password from the console keyboard. A string starts with the first non-spacing character and ends at either the end of line or, if executed within a TAKE file, at a semicolon. Indirectly obtained strings, the *@filespec* form, read the first line of the file up to but not including the explicit carriage return. Note if a trailing carriage return is needed it must be expressed numerically, such as `\13` decimal. Example:

```
input 7 Password:
echo Please type your password:
output @con
output \13
echo \13\10Thank you!
```

In this example, a TAKE file requests the user to type in the password interactively, so that it does not have to be stored on disk as part of the TAKE file.

When a script fails because an INPUT or REINPUT command did not encounter the desired string within the timeout interval the message "?Timeout" is displayed.

The OUTPUT command

Syntax: `OUTPUT {`*string*`, @`*filespec*`}`

The OUTPUT command writes the indicated character string to the serial port as ordinary text. The string may contain control or other special binary characters by representing them in backslash form. Carriage Return `(CR)`, for example, is `\13 decimal`, `\o15` octal, or `\x0D` hexadecimal. The string may use 8-bit characters if the communications parity is type NONE. A special notation is also provided, `\b` or `\B`, which causes a BREAK signal to be transmitted.

The string to be transmitted starts with the first non-spacing character after the OUTPUT command and ends at either the end of line or, if executed within a TAKE file, at a semicolon (if you need to output a semicolon from within a TAKE file, use backslash notation, e.g. ‘‘`\59`’’). Indirectly obtained strings, the *@filespec* form, read the first line of the file up to but not including the explicit carriage return.

As a convenience, text arriving at the serial port during the OUTPUT command is shown on the screen if SET INPUT-ECHO is ON, and stored in a 128-byte internal buffer for rereading by subsequent (RE)INPUT commands.

The PAUSE command

Syntax: `PAUSE [{`*number*`,` *hh:mm:ss*`}]`

PAUSE simply waits one or more seconds, or until the specified time of day, before Kermit executes the next script command. Pauses are frequently necessary to avoid overdriving the host and to let a modem proceed through a dialing sequence without interruptions from Kermit. The default waiting time is set by SET INPUT DEFAULT-TIMEOUT and is normally one second. The optional integer number selects the number of seconds to pause for this command, and the *hh:mm:ss* selects a specific time of day. An explicit value of zero produces a pause of just a few milliseconds which can be useful in some situations.

Text arriving during the PAUSE interval is shown on the screen, if SET INPUT-ECHO is ON, and stored in a 128-byte internal buffer for rereading by a following INPUT command.

PAUSE is interrupted if there is any activity on the keyboard. Thus PAUSE can be useful for operations like:

```
echo "Type any key when ready..."
pause 9999
```

PAUSE is useful in scripts that are to be executed at some future time. For instance, if you want your PC to dial up another computer and transfer some files at 9:30pm, when the phone rates are lower, you can put the command

```
PAUSE 21:30:00
```

in your script file. Note that you cannot specify a time more than 12 hours in the future. If you need to pause until a specific time that is more than 12 hours away, you can use multiple PAUSE statements:

```
PAUSE 21:30:00  ; Pause until 9:30pm tonight
PAUSE  9:30:00  ; Pause until 9:30am tomorrow morning
```

The WAIT Command

Syntax: `WAIT [{`*number*`,` *hh:mm:ss*`}] [\CD] [\CTS] [\DSR]`

WAIT performs a timed PAUSE, as above, but also examines the optional modem control signals Carrier Detect (`\CD`), Clear To Send (`\CTS`), and Data Set (modem) Ready (`\DSR`). If all of the specified signals are ON, or become ON before the timeout interval, the wait operation ceases with an indication of SUCCESS. If the time interval expires without all of the specified signals on, the status is FAILURE. Example:

```
Kermit-MS> wait 12:45:00 \cd \dsr
```

This waits until both CD and DSR asserted or until 45 minutes past noon, whichever happens first, returning SUCCESS or FAILURE respectively.

If no modem signals are specified, then WAIT is the same as PAUSE.

Labels and the GOTO Command

Labels and the GOTO command work together in the same fashion as in DOS Batch files. A label is a line which starts with a colon (:) in the leftmost column followed immediately by a word of text (no intervening spaces); material on the line after the label is ignored. The GOTO command is followed by a label, the leading colon is optional in the GOTO command. The label may be located either before or after the GOTO command and is found by searching the TAKE file or macro from the beginning. Thus, duplicated labels will always use the first occurrence. The target label must be in the current TAKE file or macro; one may not GOTO a label in another TAKE file or macro. Example:

```
:LOOP
echo again and\13
goto loop
```

will print "again and again and again and..." forever (until you type Ctrl-C). As a macro:

```
define test :loop,echo again and\13,goto loop
do test
```

Note that if a label follows a comma in a macro definition, there must be no intervening spaces:

```
define test ..., :top, ..., goto top  ; bad, space before colon.
define best ...,:top, ..., goto top   ; good, no space.
```

In this example, the best macro will work, the test macro won't.

MS-Kermit searches for the target of a GOTO starting from the top of the TAKE file, or the beginning of the macro, in which it occurs. Labels are strictly local to the current TAKE file or macro. You cannot GOTO a label outside of the current TAKE file or macro.

The IF Command

Syntax: `IF` *test-condition MS-Kermit Command*

The IF command gives MS-Kermit scripts the ability to make a decision based upon the criterion specified as the *test-condition*. If the test condition is true, then the command is executed. Otherwise, it is skipped. The test conditions are:

NOT Modifier for other conditions below.

ALARM True if the current time of day is at or later than the alarm clock time. The alarm clock time is set by the command SET ALARM time. IF ALARM distinguishes early from late with a 12 hour field of view.

COUNT True if the current COUNT variable is greater than zero. COUNT is a special Kermit variable for each active TAKE file or macro. It is set by the command SET COUNT and it is both tested and modified by the IF COUNT command. The intent is to construct simple script loops where the IF COUNT command first decreases COUNT by one (but never below zero) and then if COUNT is greater than zero the following Kermit command is executed. Because COUNT exists only for TAKE files and macros it cannot be used interactively. Each TAKE file or macro has its own distinct copy of COUNT, and nested TAKE files or macros do not interact through their COUNTs. Initially COUNT is zero.

DEFINED *symbol*
True if the named macro or variable is defined. You can use this feature to remember things for future reference.

ERRORLEVEL *number*
True if the DOS errorlevel number matches or exceeds the given (decimal) number.

EXIST *filespec*
True if the specified file exists.

FAILURE
True if the previous status-returning Kermit command reported failure.

SUCCESS
True if the previous status-returning Kermit command reported success. When using IF SUCCESS and IF FAILURE, it is important to SET INPUT TIMEOUT PROCEED, otherwise the script will quit immediately upon a failing INPUT or REINPUT, before getting to the IF statement.

IF commands are closely modeled on those of DOS Batch files, for familiarity. They consist of a test condition, perhaps modified by the leading word NOT, and then any legal Kermit command. GOTO is an especially useful command here to branch in the TAKE file or macro.

The "object" of an IF command is a Kermit command, which can be:

- A regular, predefined Kermit command, like `SEND FOO.BAR` or `SET SPEED 1200`.

- A GOTO, allowing subsequent statements to be skipped.

- Another IF command, as in `IF DEFINED \%3 IF EXIST FOO.BAR SEND FOO.BAR`. The SEND command is executed only if both IF conditions are true.

- A macro. This allows a semblence of structured programming, with an implied "begin" and "end" around the commands that compose the macro. For instance:

```
define giveup echo I give up!, hangup, stop
input 10 Login:
if failure giveup
output myusername
```

The Kermit commands which yield SUCCESS or FAILURE conditions are: GET, SEND, RECEIVE, the REMOTE commands, INPUT, REINPUT, BYE, FINISH, LOGOUT, and WAIT.

The POP and STOP Commands

Use these commands for terminating execution of a TAKE file or macro. POP terminates the current level and returns to the previous level. For example, if you gave the command "take shower", and the SHOWER file contained a command "take bath", and the BATH file contained a command "take hike", and a POP command was encountered in the HIKE file, then the next command executed would be the one following the "take hike" command in the BATH file. If a STOP command was encountered in any of these files, MS-Kermit would return immediately to interactive command level. POP and STOP work in similar fashion with nested macro invocations: POP returns to the invoking macro, STOP returns to command level.

Script Examples

A counting loop. This TAKE file excerpt says hello three times, then says goodbye:

```
set count 3             ; Prime the loop counter for three passes
:TOP                    ; A label for GOTO
echo Hello\13           ; Something to see, with carriage return
if count goto top       ; Loop if COUNT is greater than zero
echo Goodbye!\13
```

Figure 5-2 shows a simple script file that logs in to a computer, prompting the user for her password using the `@con` construction, and then connects as a terminal. Notice the semicolons used to indicate comments in TAKE files. If

```
define ermsg echo %\1\13, stop       ; Define an error handling macro.
clear                                ; Clear the input buffer.
set speed 9600                       ; Set the transmission speed.
output \13                           ; Carriage return to awaken host.
input 15 Login:                      ; Wait up to 15 secs for prompt.
if failure ermsg No_login_prompt!    ; Give up if none.
output Sari\13                       ; Send username and CR.
set input echo off                   ; Privacy, please.
input 5 Password:                    ; Quietly wait for this.
if failure ermsg No_password_prompt! ; Give up if it doesn't come.
echo Type your password now...       ; Make our own prompt.
output @CON                          ; Send console keystrokes.
output \13                           ; Add a real carriage return.
input 30 $                           ; Wait for system prompt.
if failure ermsg No_system_prompt!   ; Give up if none.
connect                              ; Start terminal emulation.
```

Figure 5-2: MS-Kermit Script for Logging In

these same commands were typed by hand at the Kermit prompt the semicolon material would be considered part of a string! Typing a Control-C will interrupt and terminate any of the commands.

Figure 5-3 illustrates some detailed control of the Hayes modem. Some understanding of the Hayes dialing language is helpful for deciphering this script (consult your Hayes modem manual). If the script is stored in a file called HAYES.SCR, then a DIAL macro can be defined like this:

```
define dial take hayes.scr
```

The trick here is that any invocation of the "dial" or "do dial" command with an operand will set the variable \%1, which is used in the TAKE file, for instance:

```
dial 765-4321
```

will set \%1 to "765-4321", the number to be dialed.

A combination of DOS Batch and Kermit Script files is shown in Figures 5-4 and 5-5 (see your DOS manual for an explanation of the batch file syntax). The purpose is to allow a user to say ‘‘SEND *filename*’’ at the DOS prompt. The DOS batch shell, SEND.BAT, and the login script, KX, are combined to login to a VAX through a data switch, run VMS Kermit in server mode, transfer the file, submit it to VMS Mail, delete the disk file, shut down the server and logout from the VAX, and report the overall transfer status. The user is asked to provide a password interactively.

5.9. Initialization Files Revisited

At Columbia University, we have IBM 370-series mainframes running VM/CMS, DECSYSTEM-20 mainframes running TOPS-20, a VAX 8700 running Unix. All of these systems are accessible through a Gandalf PACX port contention unit. The IBM systems have two different kinds of front ends, a COMTEN 3695 (similar to IBM 3705) for linemode half-duplex connections, and various Series/1-style protocol converters (including the 7171 and 4994) for full-screen, full-duplex 3270 emulation, all of which use various combinations of parity and other settings. Figure 5-6 shows an MSKERMIT.INI file composed by Vace Kundakci of Columbia to automate the task of switching his PC/AT among all these systems. It illustrates the creative use of macros and scripts. Numerous site- and system-dependent key definitions have been omitted.

A bit of explanation might clarify some of this. PACX is our port contention unit. Its output appears in even parity. It prompts us to "Enter node name =>", and we respond with the name of one of our systems or front ends, like CU20B or CU20D (DEC-20s), SIMA or SIMB (7171s), CUVMA or CUVMB (IBM mainframes front ended by COMTEN), or CUNIXC (a VAX). To connect to a given system, Vace types "do cu20b" or "do sima" to invoke a "connecting" macro. Each of these, in turn, invokes the PACX macro to navigate through the PACX to the desired system, and then invokes the appropriate macro (3695, 7171, etc) to get past any associated front end (e.g. to tell the COMTEN which IBM mainframe is wanted, or to tell the protocol converter what terminal to emulate), and then to login on the desired system, prompting on the screen for user ID and password. Finally, a macro like "vml" (VM linemode), "xed" (XEDIT, i.e. VM full screen), or "dec" (DEC-20 or VAX) is executed to set the communication parameters for the system just logged in to. The key definitions that are shown in the "vml", "xed", and "dec" macros assign the host’s character deletion code (backspace or rubout) to the AT’s backarrow key.

5.10. MS-Kermit Features for Different Systems

As noted early on, MS-Kermit was designed primarily for the IBM PC family, and later adapted to various non-IBM-compatible MS-DOS (and even non-MS-DOS) systems. Some of these adaptations provide all the features of the IBM PC version, others provide only a subset, and still others may include features not available on the IBM family. These features are all of the system-dependent variety; the Kermit file transfer protocol should be implemented identically on all versions of MS-Kermit. The most obvious differences are in the terminal emulation options and the keyboards. Table 5-7 shows the terminal emulation options for the systems presently supported by Kermit-MS, and Table 5-8, shows which keys are used for screen rollback on the various systems supported by MS-Kermit.

Another difference is the default communication port, the number of communication ports supported, and the names

```
if defined \%1 goto start            ; Make sure number specified.
echo Please supply a phone number!\13
stop
:START
clear                                ; Clear the input buffer.
set speed 2400                       ; Dial at high speed.
wait 10 \dsr                         ; Is modem turned on?
if success goto init
echo Please turn on your modem.\13   ; It’s not, complain,
stop                                 ; and exit the script.
:INIT
echo Initializing modem...\13\10     ; Modem is turned on.
output ATZ F1 Q0 V1 X4 S0=0\13       ; Initialize the modem.
input 5 OK                           ; Get its response.
if success goto dial                 ; If OK, go ahead and dial
echo Can’t initialize the modem!\13  ; Not OK, give up.
stop
:DIAL                                ; Ready to dial.
set count 5                          ; Set the redial limit.
define \%d \13Dialing                ; Initial dial message.
:REDIAL
echo \%d \%1...\13                   ; Tell them we’re dialing.
output ATDT \%1\13                   ; Dial the phone number.
clear                                ; Clear away the command echo.
input 30 CONNECT                     ; Wait for CONNECT message.
if success goto speed                ; Got it, go check speed.
define \%m No dialtone or no answer. ; Make this the error message.
reinput BUSY                         ; Didn’t connect.  Was it busy?
if failure goto later                ; No, something else.
Echo \13Busy...                      ; It’s busy, let them know.
pause 60                             ; Wait one minute.
define \%d \13Redialing              ; Change message to "Redialing".
if count goto redial                 ; Then go redial.
define \%m \13Line busy.             ; After 5 tries set this message.
:LATER                               ; Get here upon giving up.
echo \%m\10\13Try again later.\13    ; Issue error message.
stop                                 ; Exit from the script.
:SPEED                               ; Connected!
pause 1                              ; Wait for text after CONNECT.
define \%s 2400                      ; Assume speed is 2400.
reinput 1 2400                       ; Rescan current text for "2400"
if success goto done                 ; It is.
define \%s 1200                      ; It isn’t, so assume 1200.
reinput 1 1200                       ; Is it?
if failure define \%s 300            ; It isn’t, so it must be 300.
:DONE                                ; We know the speed.
set speed \%s                        ; So set it.
echo Connecting at \%s bps...\13     ; Tell the user.
connect                              ; And start terminal emulation.
```

Figure 5-3: MS-Kermit Script for More Control of a Hayes 2400 bps Modem

given to them. For instance, the IBM PC family supports COM1 and COM2, and uses COM1 by default. MS-Kermit may be persuaded to support higher-numbered IBM ports using the method outlined in section 5.17.3. For remote operation, IBM’s name for the console is CON, so if you CTTY COM1, you do CTTY CON to put the PC back to normal.

File SEND.BAT, DOS batch program:

```
echo off
Rem Kermit, one-line file mailer, by Joe Doupnik.
Rem Logon to VAX, run Kermit, Send user’s file,
Rem  post via MAIL, logout from VAX.
if ".%2" == "." goto usage
if exist %1 goto proceed
echo No file to send!
:usage
echo Usage is SEND filename username
goto done
:proceed
echo Logging onto the Vax ...
kermit set disp q,take kx,send %1,pau,rem host mail %1 %2,pau 2,bye,
if errorlevel 3 goto badrem
if errorlevel 2 goto badrcv
if errorlevel 1 goto badsnd
echo File(s) "%1" has been mailed to %2.
goto done
:badrem
echo Mail did not cooperate!
:badrcv
echo Receive failed!
goto done
:badsnd
echo Send failed!
goto done
:done
echo on
```

Figure 5-4: MS-DOS Batch File Invoking Kermit to Send VAX Mail

The DEC Rainbow

The DEC Rainbow version of MS-Kermit 2.31 uses the built-in VT102 terminal firmware and setup modes, and can operate at speeds up to 9600 baud. It has no 25th screen line, and therefore no Kermit mode line during CONNECT. It supports only the Rainbow’s single communication port, and not the printer port, so SET PORT for the Rainbow is not implemented (but of course the printer may be used for printing.) The Rainbow may be put in remote mode by CTTY AUX, and returned to normal with CTTY SCRN. The Rainbow supports several SET TERMINAL commands: VT102, VT52, and ROLL.

The keypad and cursor keys all work properly in VT102 and VT52 modes and in application as well as native states (they never had in previous versions). Newline mode is activated for received characters (LF ==> CR/LF). Screen roll back is almost 11 screenfuls. Table 5-9 shows the verb names and default key assignments for the Rainbow. On the main typewriter keyboard the shifted comma and period are converted to special keys available for Set Key assignment without impacting the normal unshifted ASCII actions; Shift Lock has no effect on these keys.

The DECmate II

MS-Kermit for the DECmate II with the XPU option is somewhat similar to Rainbow Kermit. It uses built-in terminal VT100 firmware and setup modes and baud rates up to 9600 on the single communication port. The printer port is not available for communications in this version. There is no mode line, but other connect-mode escapes are supported, including sending BREAK. Disks A through I are supported, and the floppy disk format is compatible with the Rainbow. DEC utilities are available for file conversion between DOS and WPS-8 files.

File KX, Kermit script:

```
Comment Login script for VAXA via Micom data PBX Switch.
set input timeout quit
set input echo off
set display quiet
output \13
comment - "slowly." and "CLASS" are part of the switch's prompt.
input 10 slowly.
input 10 CLASS
pause
comment - Slowly tell switch "vaxa", wait for beep.
output v
output a
output x
output a
output \13
pause
input 5 \7
comment - Done with Switch, wake up the VAX and log in.
pause
output \13
pause
input 5 Username:
set input timeout proceed
output MYNAME\13
input 2 Password:
comment - Prompt ourselves, then get password from console.
echo Enter password:
output @con
comment - Send a carriage return at the end of the password.
output \13
comment - Expect ESC Z from the VAX's Set Term/Inquire...
comment - Respond ESC [ <query symbol> 6 c   (say we are VT102).
comment - Note syntax for including question mark!
input 15 \27Z
output \27[\{63}6c
comment Look for VMS dollar sign prompt
input 15 $
comment Start VMS Kermit and place it in server mode
output kermit server\13
comment - allow server's message to finish, "machine." appears twice.
input 10 machine.
input 10 machine.
pause
```

Figure 5-5: MS-Kermit Script for Logging into VAX and Sending Mail

The NEC APC3

The NEC APC3 version of MS-Kermit assumes that the ANSI.SYS driver has been installed and that a color monitor is being used; the color graphics option is not used by Kermit. Although the display should be entirely sensible with a monochrome system, it has not been tested. Differences from the IBM PC version include:

SET BAUD: The useful baud rates supported range from 300 to 9600.

SET PORT: The available ports are 1, 2, 3, or their equivalents AUX, AUX2, AUX3.

```
; MSKERMIT.INI for IBM PC/AT Kermit 2.31, by Vace Kundakci

COMMENT - INPUT command defaults for scripts
set inp tim quit
set inp echo off
set inp case observe

COMMENT - Macros for connecting to PACX and selecting various systems
def cu20b do pacx,o cu20b\13,do 2065
def cu20d do pacx,o cu20d\13,do 2065
def sima do pacx,o sima\13,do 7171
def simb do pacx,o simb\13,do 4994
def cunixc do pacx,o cunixc\13,do 8700
def cuvma do pacx,o cuvm\13,do 3695,o vma\13,do 3083
def cuvmb do pacx,o cuvm\13,do 3695,o vmb\13,do 3083

COMMENT - Macros for logging in to various systems.
def pacx cle,set par e,o \13,i 5 Enter node name =>\32,pau
def 3695 i 5 SWITCHING CHARACTERS:\32\32
def 3083 i 5 ONLINE,o L\32,do pwd,do vml,c
def 8700 i 5 login:\32,do pwd,do dec,c
def 2065 i 5 \13\10\64,o ter vt102\13,do pwd,do dec,c
def 7171 pau,cle,o \13,i 5 TERMINAL TYPE:\32,o vt-100\13,do 3270
def 4994 pau,cle,o \13,i 5 terminal type:\32,pau,o vt100\13,do 3270
def 3270 pau,cle,o \13,o L\32,do pwd,do xed,c

COMMENT - Macros for communicating with various systems
def vml set par m,set k \270 \8,set k \3 \Kbreak,do tty
def xed set par e,set k \270 \8,set k \3,do def
def dec set par n,set k \270 \127,set k \3,do def
def def set tim of,set loc of,set ter wr of,set han non,set flo xon
def tty set tim on,set loc on,set ter wr on,set han xon,set flo non

COMMENT - Macro for obtaining user ID and password
def pwd echo user:,o @con,o \13, echo Password:,o @con,o \13
```

Figure 5-6: An Advanced MS-Kermit Initialization File

SET TERMINAL COLOR: Instead of specifying colors by number, the words BLUE, RED, MAGENTA, GREEN, CYAN, YELLOW, or WHITE are appropriate. This is the color of the text in connect mode; background colors are not available. Monochrome monitors will respond with display changing from most dim to most bright if the colors are specified in the order given.

SET TERMINAL KEYCLICK: Not implemented in Kermit; use the NEC provided command.

SET TERMINAL SCREEN-BACKGROUND: Not implemented.

During terminal emulation, screen scroll is handled by the PgUp and PgDn keys. If used in combination with the Ctrl key, the display moves but one line. If used in combination with the Fnc key, the display scrolls to the end of the buffer. The Fnc-INS combination toggles the mode line on/off. The Fnc-DEL combination toggles the terminal emulation type. The Fnc-Break combination resets the emulator. The Help key pulls down the connect mode menu. The ANSI escape sequence for disable/enable cursor is implemented.

System	EscChar	Capabilities	Terminal Service
ACT Apricot	`^]`	`K`	VT52 ???
DEC Rainbow	`^]`	`R P K D`	VT102 firmware
DECmate/DOS	`^]`	`K`	VT100
Generic DOS	`^]`	`K`	Depends on system
Grid Compass	`^]`	`K`	???
HP-110	`^]`	`K`	Dumb terminal
HP-150	`^]`	`R K`	HP-2623 firmware
IBM PC family	`^]`	`R M P K D`	H19,VT52,VT102,Tek emulation
Intel 3xx	`^]`	`K`	Uses real terminal
NEC 9801	`^]`	`M P K D`	VT102, Tektronix emulation
NEC APC3	`^]`	`R M P K D`	H19,VT52,VT102 emulation
NEC APC	`^]`	`R P K`	VT100, ADM3A firmware
Olivetti M24	`^]`	`R M P K D`	Same as IBM PC
Sanyo MBC55x	`^]`	`R M P K D`	H19,VT52,VT102 emulation
Wang PC	`^A`	`K`	Wang firmware
TI Pro	`^]`	`M P K`	VT100/Tektronix
Victor 9000	`Alt-]`	`M P K D`	H19,VT52,VT102 and/or Tek4010
Zenith Z100	`^]`	`K`	Heath-19 emulation

`R`=Rollback, `M`=Modeline, `P`=Printer control, `K`=Key redefinition, `D`=screen Dump

Table 5-7: Kermit-MS Terminal Emulation Options

System	Screen Down	Line Down	Screen Up	Line Up
IBM PC	PgUp	Ctrl-PgUp	PgDn	Ctrl-PgDn
Rainbow	PrevScreen	Ctrl-PrevScreen	NextScreen	Ctrl-NextScreen
HP-150	Prev	Shift-UpArrow	Next	Shift-DownArrow
NEC APC	Uparrow	Ctrl-UpArrow	DownArrow	Ctrl-DownArrow
NEC APC3	PgUp	Ctrl-PgUp	PgDn	Ctrl-PgDn
Sanyo 55x	PgUp	Ctrl-RtArrow	PgDn	Ctrl-PgDn

The IBM PC also allows use of the Home key to get to the top of its display memory and End key to get to the bottom, and the keypad minus (`-`) key to toggle the mode line on and off. The Rainbow uses Shift-Next-Screen to get to the bottom of its display memory, but provides no key for moving directly to the top.

Table 5-8: Kermit-MS Screen Scroll Keys

5.11. Compatibility with Older Versions of MS-DOS Kermit

The last monolithic (single source file) release of MS-DOS Kermit was 1.20. Meanwhile, implementations based on versions of that vintage will have at least the following incompatibilies from the version described here:

- "RECEIVE filespec" is used instead of "GET filespec". There is no GET command in older versions, and no way to specify a new name for an incoming file.
- No LOCAL or REMOTE commands.
- No 8th-bit prefixing, repeat counts, CRCs or 2-character checksums.
- No TAKE or initialization files.
- No command macros or command line arguments.
- No terminal session logging.

and others, depending on the specific version.

```
Rainbow Key           Verb Name      Operation

PF1                   \Kpf1,\Kgold   Keypad function key
PF2..PF4              \Kpf2..\Kpf4   Keypad function keys
keypad 0..9           \Kkp0..\Kkp9   Keypad digit keys
keypad -              \Kkpminus      Keypad minus key
keypad ,              \Kkpcoma       Keypad commma
keypad .              \Kkpdot        Keypad dot (period) key
keypad Enter          \Kkpenter      Keypad Enter key

up arrow              \Kuparr        Cursor keys
down arrow            \Kdnarr
left arrow            \Klfarr
right arrow           \Krtarr
Shift Prev Screen     \Khome         Rewind to start of screen buffer
Shift Next Screen     \Kend          Unwind to end of screen buffer
Ctrl Prev screen      \Kupone        Backup one screen line
Ctrl Next screen      \Kdnone        Advance one screen line
Prev screen           \Kupscn        Backup one screen
Next screen           \Kdnscn        Advance one screen
Print Screen          \Kprtscr       Copy screen to printer
Ctrl Print Screen     \Ktoggle_prn   Toggle echoing screen to printer
                                      (printer failure resets toggle)
Do                    \Kdump         Copy screen to file (KERMIT.SCN)
Break                 \Kbreak        Send a BREAK
Shift Break           \Klbreak       Send a Long BREAK
Main Screen           \KDOS          Push to DOS
Help                  \Khelp         Show Connect mode help menu
Exit                  \Kexit         Exit Connect mode
*                     \Knull         send a null out the serial port
*                     \Khangup       hangup phone by dropping DTR, RTS
*                     \Klogon        resume logging, if active
*                     \Klogof        suspend logging
*                     \Kstatus       display status table

* (verbs not pre-assigned to keys)
```

Table 5-9: Kermit-MS Verbs for the DEC Rainbow

Incompatibilities between 2.29 and both 2.30 and 2.31 include:

- LOCAL command has been removed from 2.30 and 2.31.
- CLEAR command now means clear serial port buffer rather than key and macro definitions. Key and macro definition string space is now garbage collected, so a CLEAR command for them is no longer necessary.
- CLRINP command is gone (replaced by CLEAR).
- Numbers of the form *nnn* default to decimal rather than octal.
- Status of Default Disk is now shown as default disk and path.
- LOG *filespec* replaced by LOG SESSION *filespec* and LOG PACKET *filespec*.
- SET KEY and SHOW KEY commands use different key identifications and syntax:

MS-Kermit no longer understands keycap names such as F1 and BACKSPACE because the codes are now highly dependent on individual keyboards, software, and computers. Also, not every key press combination is supported by the system software and key codes do depend on the keyboard in use. Thus, the SHOW KEY command is normally used to obtain codes for keys on your system. In most cases, defining one key also redefines all other keys sending the same character. This is a side effect of not knowing the physical details of every keyboard. However, efforts have been made to recognize many such "aliased" keys and to generate unique identifications for each. Special keys, such as F1, F2 and others which do not send an ASCII code are usually unique and are identified by scan codes.

Previous versions of MS Kermit used a different key coding algorithm and not all old codes map to the expected keys. However, Kermit does attempt to use the older SET KEY syntax properly as much as possible. The older syntax required the keyword SCAN followed by a number WITHOUT the BACKSLASH. The current MS Kermit uses decimal as the default number base and previous versions used octal in certain commands. So, when Kermit senses an old style SET KEY command it converts the number, displays the new format and gives a warning message. It is best to make a new style SET KEY file.

5.12. What's Missing

Kermit-MS has plenty of room for improvement. Missing features (which may be added in future releases) include:

- Sliding window transport protocol.
- Default filetype for TAKE command files.
- Passing paramaters in TAKE command, like in DO command.
- A way to send files with their full path names.
- A way to play back session logs directly from disk to screen.
- Trapping of carrier loss during CONNECT or file transfer.
- Pause at end of screen during local TYPE.
- A better built-in help facility.
- A byte-stuffing mechanism during file transfer to get past devices that are not transparent to all printable ASCII characters.

Future releases of MS-Kermit will probably have major portions of the program (now written entirely in assembler) replaced by C-language code. This would include the file transfer portions, the command parser, etc.

5.13. Installation of Kermit-MS

If you already have Kermit on your PC, you can use it to obtain new versions of Kermit-MS when they appear on the central system at your site. If you do not have Kermit or any other reliable file capture facility on your PC, you can order a Kermit diskette from Columbia (write to Kermit Distribution, Columbia University Center for Computing Activities, 612 West 115th Street, New York, NY 10025, USA, for information), or from any of a number of user groups or diskette services. If you don't have Kermit already, and absolutely can't get a Kermit diskette, but have access to another computer that has a copy of the MS-DOS Kermit program (usually in ``.BOO'' format, explained below), there are two recommended methods for getting it onto your PC:

1. Use another file capture facility to get it.
2. Type in and run the "baby Kermit" program (72 lines) from chapter 7 of the Kermit book.

The first method involves either "raw capture" (no error checking), or else use of (gasp!) another protocol, such as Xmodem, which, like Kermit, requires a program to execute the same protocol on both ends of the connection.

Raw capture generally involves "typing" the file on the other computer, with your PC taking the place of the terminal, and rather than displaying the file on the screen as it's being typed, your PC is storing it on the disk. This is a tricky process, however, because data can easily be lost or corrupted. For instance, you could write a very short BASIC program to capture a file in this way, but it could probably not keep up -- even at low baud rates -- with the transmission speed unless you included the tricky serial port BASIC commands. The DOS command COPY COM1 *filename* command has the same speed problem, and it stops only when it receives a Control-Z character from the other computer.

If the other computer has Kermit on it -- which is likely, since this is probably the reason you want to get Kermit onto your PC -- you should type in the receive-only BASIC Kermit program listed on pp.186-188 of the Kermit book, and then use it in conjunction with the other computer's Kermit to transfer the file. Make sure to set a long enough delay on the other computer to give yourself time to escape back to the PC and start up the "baby Kermit" before packets start to arrive, otherwise you'll probably get fatal DOS i/o errors.

Note that Kermit programs are often distributed under names other than "Kermit". The Columbia Kermit program library contains hundreds of Kermit programs, which must be given unique names. MS-DOS Kermit for the IBM PC, for instance, is called `MSVIBM`. Once you have this program in `.EXE` format on your disk, you probably should rename it to `KERMIT.EXE`, because the distribution name is harder to remember (and type).

You will probably also want to create an MS-Kermit initialization file. A sample is distributed with MS-Kermit as `MSVIBM.INI`. This should be tailored to your requirements, and then renamed to `MSKERMIT.INI`, and stored where Kermit can find it (in the current directory or any directory in your DOS PATH).

".BOO Files"

MS-Kermit (and many other Kermit programs) are often distributed using a special encoding called "boo" (short for "bootstrap") format, developed especially for distribution of MS-Kermit over networks and communication lines. MS-Kermit has grown to have so many features that the binary program image (the `.EXE` file) has become quite large. But binary files are generally not compatible with the common labeled tape formats (e.g. ANSI D), electronic mail, or raw downloading -- the methods most commonly used for Kermit distribution.

A common practice is to encode `.EXE` and other binary files into printable characters, such as hexadecimal digits, for transportability. A simple "hex" encoding results in two characters per 8-bit binary byte, plus CRLFs added every 80 (or less) hex characters to allow the file to pass through card-oriented links. A hex file is therefore more than twice as large as the original binary file.

A `.BOO` file is a more compact, but somewhat more complicated, encoding. Every three binary bytes (24 bits) are split up into four 6-bit bytes with 48 (ASCII character ‘‘`0`’’) added to each, resulting in four ASCII characters ranging from ‘‘`0`’’ (ASCII 48) to ‘‘`o`’’ (ASCII 111), with CRLFs added at or near "column 76". The resulting file size would therefore be about 4/3 the `.EXE` file size. This is still quite large, so `.BOO` files also compress consecutive null (zero) bytes. Up to 78 consecutive nulls are compressed into two characters. Tilde (‘‘~’’) is the null-compression lead-in, and the following character indicates how many nulls are represented (subtract 48 from this character's ASCII value). For instance ‘‘`~A`’’ means 17 consecutive nulls; ‘‘~~’’ means 78 of them. Repeated nulls are very common in `.EXE` files.

4-for-3 encoding combined with null compression reduces the size of the encoded file to approximately the same size as the original `.EXE file`, and sometimes even smaller. The first line of a `.BOO` file is the name (in plain text) of the original file. Here's what the first few lines of a typical `.BOO` file look like:

```
MSVIBM.EXE
CEYP0Id05@0P~3oomo2Y01FWeP8@007P000040HB4001‘W~28bL005\W~2JBP00722V0ZHPYP:
\8:H2]R2V0[‘PYP:68>H2S23V0YHPiP:Xg800;Qd~2UWD006Yg~2Ogl009]o~2L8000;20~~~~
~~~~~~~:R2H008TV?P761T410<H6@P40j4l6RRH0083l17@PP?‘1M@?YSP20o0Ee0nUD0h3l
1WD3jO@3]0VjW03=8L?X4‘N0o01h1\H6~20l>0i7n0o1]e7[@2\PO=8LH60@00Raj>04^97Xh0
```

Programs for Handling .BOO Files

Kermit Distribution includes several useful `.BOO`-file programs:

`MSBPCT.BAS` This Microsoft BASIC program can be used on any PC that has BASIC to decode a `.BOO` file into an `.EXE` file. It's about 50 lines line, so it can be typed in.

`MSBPCT.BOO` BASIC programs run rather slowly, so `.BOO`-file decoders have also been written in high-level languages like C. The `MSBPCT.EXE` file that was produced by compiling `MSBPCT.C` is encoded into `MSBPCT.BOO`, which can be decoded back into `MSBPCT.EXE` using `MSBPCT.BAS`. Once you've done that, you don't need to run the slow BASIC version any more, which is a blessing, because the MS-Kermit `.BOO` file takes up to half an hour to decode using the BASIC version (depending on the system), but only seconds using `MSBPCT.EXE`.

`MSBPCT.*`	There are `.BOO`-file decoders written in other languages too, like assembler, Turbo Pascal, Fortran, etc. Take your pick. They all do the same thing.
`MSBMKB.*`	This is the program for encoding an `.EXE` file into a `.BOO` file. It is written in C, compiled, and translated (by itself) into `.BOO` format, suitable for decoding back into `.EXE` form by any of the MSBPCT programs. Also in other languages, including Fortran and Turbo Pascal.

`MSBHEX.*` are C programs for producing and decoding straight hex files.

5.14. Program Organization

Kermit-MS version 2 is composed of separate assembler source files, assembled separately, and linked together. The modules are:

System/Device Independent:

`MSSKER.ASM`	Main program
`MSSSEN.ASM`	File sender
`MSSRCV.ASM`	File receiver
`MSSSER.ASM`	Server operation
`MSSFIL.ASM`	File i/o
`MSSCMD.ASM`	Command parser
`MSSTER.ASM`	CONNECT command
`MSSCOM.ASM`	Packet reader and sender
`MSSSET.ASM`	SET, SHOW, and STATUS commands
`MSSSCP.ASM`	Script CLEAR, ECHO, INPUT, OUTPUT, PAUSE, TRANSMIT commands
`MSSFIN.ASM`	Dummy module for the end of the data segment; must be linked LAST.
`MSSDEF.H`	Data structure definitions and equates

System/Device Dependent:

MSG*xxx*.ASM	System-dependent graphics terminal for system *xxx*
MSU*xxx*.ASM	System-dependent keyboard translator for system *xxx*
MSX*xxx*.ASM	System-dependent code for system *xxx*
MSY*xxx*.ASM	Terminal emulation for system *xxx*
MSZ*xxx*.ASM	More terminal emulation for system *xxx*

The *xxx* is replaced by a 3-letter code for the particular system, e.g. IBM for the IBM PC family, RB1 for the Rainbow-100, etc.

The modular organization allows easier modification of the program, quicker transfer of modified portions from system-to-system. The modules are designed to be well-defined and self-contained, such that they can be easily replaced. For instance, someone who prefers windows and mice to typing commands should be able to replace the command parsing module without having to worry about the effect on the other modules.

To assemble any of the Kermit modules, file `MSSDEF.H` must be on the default disk.

All the Kermit implementations require the modules MSSCMD, MSSCOM, MSSFIL, MSSKER, MSSRCV, MSSSCP, MSSSEN, MSSSER, MSSSET, MSSTER, MSSFIN. MSSFIN *must* be linked last.

Each particular implementation requires at least an MSX*xxx* module, usually an MSU*xxx* module, and, if it is doing terminal emulation in software, also an MSY*xxx* and possible also an MSZ*xxx* module, and for graphics terminal emulation, also an MSG*xxx* module. See the batch or make files from the source distribution for details of exactly which modules are required for a particular implementation.

Once all the required object modules exist, they may be linked together to produce a Kermit program. For example, on the IBM PC:

```
A>link

   Microsoft Object Linker V2.00
(C) Copyright 1982 by Microsoft Inc.

Object Modules [.OBJ]: msscmd+msscom+mssfil+mssker+mssrcv+mssscp+msssen+
mssser+mssset+msster+msgibm+msuibm+msxibm+msyibm+mszibm+mssfin
Run File [MSSCMD.EXE]: kermit
List File [NUL.MAP]:;

    A>
```

Warning: old versions of MASM may not be able to assemble several of the large files now present in Kermit-MS. The solution is to acquire Microsoft MASM 4.0 or later.

5.15. Bringing Kermit to New Systems

You can bring Kermit-MS to MS-DOS systems that are not explicitly supported in one of two ways -- attempt to run the "generic" MS-DOS Kermit on it, or add explicit code to support your system.

To get started with Kermit on a new system, try running "generic" MS-DOS Kermit; in many cases, it will run as is. The generic version accomplishes all its port and console i/o through DOS calls, and during terminal connection does not attempt to emulate any particular kind of terminal. In some cases, the generic version may still require some fiddling to run on a new system; for instance, different systems refer to their communication ports in different ways -- COM1, J1, AUX, etc. The SET PORT command allows you to specify the port using any of these device names, or using DOS file handles -- keep trying until you find the one that works. Generic MS-DOS Kermit will probably run no faster than 1200 baud, and it only works with DOS 2.0 or later.

If you want to write code to explicitly support a new system, first call or write Kermit Distribution at Columbia to make sure no one else is already doing the same work. If you're the first, then begin by reading the file `MSXAAA.DOC`, provided with the MS-DOS Kermit sources in the Kermit distribution, which is a guide to the system dependent modules of Kermit-MS. Then create new `MSU`*xxx*`.ASM` and `MSX`*xxx*`.ASM` modules, and, if your version is also doing terminal emulation in software, also an `MSY` and possibly an `MSZ` module patterned after those that have been written for other systems.

5.16. Kermit-MS VT102 Terminal Emulator Technical Summary

5.16.1. Treatment of Inbound Characters During Terminal Emulation

Many things can happen to a character that arrives at the communication port before you see it. The actual sequence of events is:

1. Obtain character from serial port.
2. Remove high bit if parity is other than none.
3. Detect and remove xon/xoff if FLOW is XON/XOFF.
4. If DEBUG is active (ON or SESSION) then put character to debug style display, otherwise:
5. If transparent printing is active (for VT102 emulators) then print the character but do not show it on the display, otherwise:
6. Remove high-order bit if DISPLAY is 7-bit
7. If an escape sequence is not in progress and TRANSLATE INPUT is ON, translate.
8. If LOG SESSION is active then copy character to the log file
9. Pass the character to the terminal emulator for interpretation or display.

The following sections summarize the Kermit-MS keyboard and screen operation during emulation of H19, VT52, and VT102 terminals, prinicipally for the IBM PC but also used by the NEC APC3, Victor 9000, and Sanyo 55x systems.

5.16.2. Keyboard Layout and Characters Sent

Here is how the keypad functions are assigned to the IBM keyboard function keys. You may change them by using the SET KEY command to define a desired key as the appropriate Kermit action verb; use SET KEY without a definition to undefine a key. Names of appropriate verbs are also shown for use in the Set Key command, such as

```
Set Key \2352 \Kbreak
```
(IBM Alt-B assigned to verb BREAK)

Verb names are system dependent, use ? in the Set Key definition part for a list of local verbs. IBM PC verbs are listed in Table 5-6; IBM key values are either straight ASCII or the IBM Bios scan code, plus 256, plus 512 for Shift key held down, plus 1024 for Control key held down, plus 2048 for Alt key held down; non-ASCII keys are always 256 decimal or greater. Keys particular to the Enhanced Keyboard have 4096 added to the result.

```
    Heath-19 and VT52 Keypads                 VT102 keypad
       IBM Keys                                 IBM keys
+------+------+-------+----------+   +------+------+------+------+
| Blue |  Red |  Grey | up arrow |   |  PF1 |  PF2 |  PF3 |  PF4 |
|  F1  |  F2  |  F3   | up arrow |   |  F1  |  F2  |  F3  |  F4  |
+------+------+-------+----------+   +------+------+------+------+
|  7   |  8   |  9    |down arrow|   |  7   |  8   |  9   |  -   |
|  F5  |  F6  |  F7   |down arrow|   |  F5  |  F6  |  F7  |  F8  |
+------+------+-------+----------+   +------+------+------+------+
|  4   |  5   |  6    | rgt arrow|   |  4   |  5   |  6   |  ,   |
|  F9  |  F10 |  SF1  | rgt arrow|   |  F9  |  F10 |  SF1 |  SF2 |
+------+------+-------+----------+   +------+------+------+------+
|  1   |  2   |  3    |left arrow|   |  1   |  2   |  3   |  E   |
|  SF3 |  SF4 |  SF5  |left arrow|   |  SF3 |  SF4 |  SF5 |  n  S|
+------+------+-------+----------+   +------+------+------+  t  F|
|  0------0   |  .    |  Enter   |   |  0------0   |  .   |  e  6|
|  SF7        |  SF8  |  SF6     |   |   SF7       |  SF8 |  r   |
+-------------+-------+----------+   +-------------+------+------+
```

SF1 means push Shift and F1 keys simultaneously

CURSOR KEYS:

VT52/H19 key	IBM Verb	IBM key	H-19 & VT52 All Modes	VT102 Numeric	VT102 Application
up arrow	UPARR	up arrow	ESC A	ESC [A	ESC O A
down arrow	DNARR	down arrow	ESC B	ESC [B	ESC O B
right arrow	RTARR	right arrow	ESC C	ESC [C	ESC O C
left arrow	LFARR	left arrow	ESC D	ESC [D	ESC O D

AUXILIARY KEYPAD:

VT52/H19 key	IBM Verb	IBM key	Heath-19 & VT52 Numeric	Heath-19 & VT52 Applic.	VT102 Numeric	VT102 Applic.
PF1/HF7/Blue	GOLD,PF1	F1	ESC P	ESC P	ESC O P	ESC O P
PF2/HF8/Red	PF2	F2	ESC Q	ESC Q	ESC O Q	ESC O Q
PF3/HF9/Grey	PF3	F3	ESC R	ESC R	ESC O R	ESC O R
PF4/HF1	PF4	F4	ESC S	ESC S	ESC O S	ESC O S
0	KP0	SF7	0	ESC ? p	0	ESC O p
1	KP1	SF3	1	ESC ? q	1	ESC O q
2	KP2	SF4	2	ESC ? r	2	ESC O r
3	KP3	SF5	3	ESC ? s	3	ESC O s
4	KP4	F9	4	ESC ? t	4	ESC O t
5	KP5	F10	5	ESC ? u	5	ESC O u
6	KP6	SF1	6	ESC ? v	6	ESC O v
7	KP7	F5	7	ESC ? w	7	ESC O w
8	KP8	F6	8	ESC ? x	8	ESC O x

```
9              KP9        F7         9       ESC ? y   9        ESC O y
comma  (,)     KPCOMA     SF2        ,       ESC ? l   ,        ESC O l
minus  (-)     KPMINUS    F8         -       ESC ? m   -        ESC O m
period (.)     KPDOT      SF8        .       ESC ? n   .        ESC O n
Enter          KPENTER    SF6        ^M(cr)  ESC ? M   ^M       ESC O M
```

(SFn means hold down Shift key while pressing Function key n.)

An often confusing item is knowing the mode of the auxillary keypad: numeric or application. Digital Equipment Corporation designed the terminal to change modes only under command from the remote computer and not at all from the keyboard. So the startup state is numeric/cursor mode, and reception of escape sequences ‘‘`ESC [ ? 1 h`’’ or ‘‘`l`’’ changes the mode. Kermit verbs for the keypad and cursor keys generate the correct escape sequences appropriate to the current mode and terminal type.

A best attempt is made to safely test for the 101/102 key Enhanced keyboard and use it if present. If it is present then the keyboard translator separates the individual arrow keys from those on the numeric keypad and also separates the asterisk and forward slash keys on the keypad from those on the regular typewriter keyboard. These special Enhanced keyboard keys are reported as scan codes with 4096 added to the base scan code.

OTHER IBM KEYS OPERATIONAL IN CONNECT MODE:

```
IBM key          IBM Verb      Action

Keypad Del                     Send ASCII Del code (rubout) \127
Backspace (<-)                 Send ASCII Del code (rubout) \127  (BS is \8)
Keypad -         MODELINE      Toggle mode line on/off (only if Mode Line is
                               enabled and not used by the host).
Alt -            TERMTYPE      Toggle among H-19, VT52, and VT100 emulations.
Alt =            RESET         Clear screen and reset terminal emulator to
                               starting (setup) state.
Alt B            BREAK         Send a BREAK signal
Alt H            HELP          Show drop down help menu (detailed below)
Alt S            STATUS        Show settings
Alt X            EXIT          Exit Connect mode, back to Kermit prompt

Home             HOMSCN        Roll screen up (text down) to beginning of
                                storage.
End              ENDSCN        Roll screen down (text up) to end of storage.
PgUp             UPSCN         Roll screen up (back, earlier) one screen.
PgDn             DNSCN         Roll screen down (forward, later) one screen.
Ctrl-PgUp        UPONE         Roll screen up one line.
Ctrl-PdDn        DNONE         Roll screen down one line.

Control PrtSc    PRTSCN        Toggle  on/off copying of received text to
                               printer, "PRN" shows on far right of mode
                               line when activated.

Control-End      DUMP          Dump image of screen to a disk file or device.
                               Default filename is KERMIT.SCN in the current
                               directory. Use command SET DUMP to change the
                               filename.  Screen images are appended to the
                               file, separated by formfeeds.

Shift-PrtSc      Standard DOS Print-screen, dump screen image to printer.

unassigned       HOLDSCRN      DEC style Holdscreen, same as typing Control-S.
```

"`Alt -`" means hold down Alt and type minus on the upper key rank. This switches among the various kinds of emulation but does not change most operating parameters of the emulator.

CONNECT ESCAPE COMMANDS:

Type the Kermit escape character (normally ‘‘^]’’), then one of the keys below:

```
                                                      (equivalent IBM Verb)
     ?   display this short list.                             HELP
     0   send a null character.                               NULL
     B   send a BREAK signal.                                 BREAK
     C   close connect session & return to Kermit prompt.     EXIT
     F   dump screen to filespec, default is KERMIT.SCN.      DUMP
     H   hangup the phone or network connection               HANGUP
     L   send a Long BREAK signal                             LBREAK
     M   toggle mode line on/off.                             MODELINE
     P   push to DOS.                                         DOS
     Q   quit (suspend) logging.                              LOGOFF
     R   resume logging.                                      LOGON
     S   show status.                                         STATUS
     Kermit escape character itself: send it to the host.
```

5.16.3. Responses To Characters Received By the Terminal Emulator

Spaces shown between characters of escape sequences are there for ease of reading. The actual sequences contain no spaces. Unknown escape sequences of the form "ESC char" are absorbed by the emulator without further effect; longer unknown escape sequences echo the extra characters.

DEC VT102 functions while in ANSI (VT102) mode, unsupported features marked by an asterisk (*):

```
 Escape Seq      Mnemonic       Description of Action

 ESC D           IND            Index, moves cursor down one line, can scroll
 ESC E           NEL            Move cursor to start of line below, can scroll
 ESC H           HTS            Set one horizontal tab at current position
 ESC M           RI             Reverse Index, cursor up one line, can scroll
 ESC Z           DECID          Identify terminal (response is ESC [ ? 6 c)
 ESC c           RIS            Reset terminal to initial state
 ESC =           DECKPAM        Enter keypad application mode
 ESC >           DECKNPNM       Enter keypad numeric mode
 ESC 7           DECSC          Save cursor position and attributes
 ESC 8           DECRC          Restore cursor from previously saved position
 ESC # 3         DECDHL         Double height and width line, top half
 ESC # 4         DECDHL         Double height and width line, bottom half
 ESC # 5         DECSWL         Single height and width line
 ESC # 6         DECDWL         Double width single height line
 ESC # 8         DECALN         Test screen alignment, fill screen with E’s
 ESC [ Pn @      ICH            ANSI insert Pn spaces at and after cursor
 ESC [ Pn A      CUU            Cursor up Pn lines, does not scroll
 ESC [ Pn B      CUD            Cursor down Pn lines, does not scroll
 ESC [ Pn C      CUF            Cursor forward, stays on same line
 ESC [ Pn D      CUB            Cursor backward, stays on same line
 ESC [ Pn; Pn H  CUP            Set cursor to row, column (same as HVP)
 ESC [ Ps J      ED             Erase in display:
                                 0 = cursor to end of screen, inclusive
                                 1 = start of screen to cursor, inclusive
                                 2 = entire screen, reset lines to single
                                     width, cursor does not move.
 ESC [ Ps K      EL             Erase in line:
                                 0 = cursor to end of line, inclusive
                                 1 = start of line to cursor, inclusive
                                 2 = entire line, cursor does not move
 ESC [ Pn L      IL             Insert Pn lines preceding current line.
 ESC [ Pn M      DL             Delete Pn lines from current downward, incl.
 ESC [ Pn P      DCH            Delete Pn chars from cursor to left, incl.
 ESC [ Pn; Pn R  CPR            Cursor report (row, column), sent by terminal
                                Example: home position yields ESC [ 1; 1 R
```

```
ESC [ Pn c      DA               Device attributes (reports ESC [ ? 6 c)
ESC [ Pn; Pn f HVP              Set cursor to row, column (same as CUP)
ESC [ Ps g      TBC              Tabs clear, 0 = at this position, 3 = all
ESC [ 4 h       IRM              Insert mode on
ESC [ 20 h      LNM              Set newline mode (cr => cr/lf)
ESC [ 4 l       IRM              Replacement mode on
ESC [ 20 l      LNM              Reset newline mode (cr => cr)
ESC [ ? Ps;...;Ps h  SM          Set mode, see table below
ESC [ ? Ps;...;Ps l  RM          Reset mode, see table below
        Ps      Mnemonic         Mode           Set (h)         Reset (l)
        0                        error (ignored)
        1       DECCKM           cursor keys    application     cursor/numeric
        2       DECANM           ANSI/VT52      ANSI/VT102      VT52
        3       DECCOLM          Columns        +132 col        80 col
        4       DECSCLM         *Scrolling      smooth          jump
        5       DECSCNM          Screen         reverse video   normal
        6       DECOM            Origin         relative        absolute
        7       DECAWM           Autowrap       on              off
        8       DECARM          *Autorepeat     on              off
        9       DECINLM         *Interlace      on              off
        18      DECPFF           Printer termination character, use FF if set
        19      DECPEX           Printer extent,set=screen,off=scrolling region
        38      n/a              Graphics (Tek) ++graphics      text
                               + See comments on EGA boards.
                              ++ Ignored if DISABLE TEK has been given.
ESC [ Pn i      MC               Printer controls (Media Copy)
        0                         Print whole Screen
        4                         Exit printer controller (transparent print)
        5                         Enter printer controller (transparent print)
ESC [ ? Pn i    MC               Printer controls (Media Copy)
        1                         Print line containing cursor
        4                         Exit auto print (stop echoing to printer)
        5                         Enter autoprint (echo screen chars to printer)
ESC [ Ps;...;Ps m  SGR           Select graphic rendition
                                  0 = all attributes off (#'s 1, 4, 5, 7)
                                  1 = bold, intensify foreground
                                  4 = underscore (reverse video on IBM CGA)
                                  5 = blink
                                  7 = reverse video
                non-DEC extensions: 30-37 = foreground color = 30 + colors
                                    40-47 = background color = 40 + colors
                                       colors: 1 = red, 2 = green, 4 = blue
ESC [ Ps n      DSR              Device Status Report.
                                 Response from VT100: 0=ready, 3=malfunction.
                                 Command to VT100: 5=report status with DSR,
                                 6=report cursor position using CPR sequence.
ESC [ Ps;...;Ps q  DECLL         Load LEDs, Ps = 0 means clear LED #1-4
                                 Ps = 1,2,3,4 sets LED # 1,2,3,4 on status line.
ESC [ Pn; Pn r DECSTBM           Set top and bottom scrolling margins, resp.
                                 ESC [ r resets margin to full screen.
ESC [ sol x     DECREQTPARM      Request terminal parameters, see table below
ESC [ sol; par; nbits; xspeed; rspeed; clkmul; flags x
                DECREPTPARM      Reports terminal parameters
                                 sol = 0  request; terminal can send unsolicited
                                   reports - supported as sol = 1 below.
                                 sol = 1, request; term reports only on request
                                 sol = 2, this is a report (DECREPTPARM)
                                 sol = 3, terminal reporting only on request
                                 par = 1 none, 2 space, 3 mark, 4 odd, 5 even
                                 nbits = 1 (8 bits/char), 2 (7 bits/char)
                                 xspeed,rspeed = transmit & receive speed index
  0,8,16,24,32,40,48,56,64,72,80,88,96,104,112,120,128 correspond to speeds of
  50,75,110,134.5,150,200,300,600,1200,1800,2000,2400,3600,4800,9600,19200,
```

and 38400 baud.		clkmul = 1 (clock rate multiplier is 16)
		flags = 0-15 (Setup Block #5), always 0 here
ESC [2; Ps y	DECST	*Confidence tests - not supported
	SCS	Select character sets.
ESC (A	SCS	G0 points to UK symbols
ESC) A	SCS	G1 points to UK symbols
ESC (B	SCS	G0 points to ASCII symbols
ESC) B	SCS	G1 points to ASCII symbols
ESC (0	SCS	G0 points to special (line drawing) graphics
ESC) 0	SCS	G1 points to special (line drawing) graphics
ESC (1	SCS	G0 points to alt char ROM - UK symbols
ESC) 1	SCS	G1 points to alt char ROM - UK symbols
ESC (2	SCS	G0 points to alt graphics ROM - as ESC (0
ESC) 2	SCS	G1 points to alt graphics ROM - as ESC) 0
		(Separate graphics used for DEC and Heath)
^E	ENQ	*Answerback message (not supported)
^G	BELL	Sound VT102 style beep
^H	BS	Backspace, move cursor left one character
^I	HT	Horizontal tab, move cursor to next tabstop
^J	LF	Linefeed, move cursor down one line
^K	VT	Vertical Tab, treated as a line feed
^L	FF	Formfeed, treated as a line feed
^M	CR	Carriage return, move cursor to col 1
^N	SO	Select usage of G1 character set
^O	SI	Select usage of G0 character set
^X	CAN	Cancel escape sequence in progress
^Z	SUB	Treated as a CAN

Other extensions:

ESC [25; Pc f	VT52/VT100 move cursor to 25th line.
ESC [25; Pc H	VT52/VT100 move cursor to 25th line.
	(These will disable Kermit’s own status line.)
ESC * char	VT200 series graphics command, ignored.
ESC ^L	Enter Tektronix sub-mode, clear Tek screen.
	(This is ignored if DISABLE TEK has been given)

5.16.4. DEC VT102 Functions While in VT52 Mode

Escape sequence	Description of action
ESC A	Cursor up
ESC B	Cursor down
ESC C	Cursor right
ESC D	Cursor left
ESC F	Enter graphics mode
ESC G	Exit graphics mode
ESC H	Cursor home
ESC I	Reverse line feed
ESC J	Erase to end of screen
ESC K	Erase to end of line
ESC V	Print cursor line
ESC X	Exit Printer Controller mode, transparent print
ESC Y row column	Direct cursor address, offset from space
ESC W	Enter Printer Controller mode,transparent print
ESC Z	Identify (response is ESC / Z)
ESC ^ (caret)	Enter autoprint mode (printer echoes screen)
ESC _ (underscore)	Exit autoprint mode
ESC]	Print Screen
ESC =	Enter alternate keypad mode
ESC >	Exit alternate keypad mode
ESC <	Enter ANSI mode (changes to VT102)

5.16.5. Heath-19 Functions While in Non-ANSI Mode

Escape seq	Mnemonic	Description of action
ESC A	HCUU	Cursor Up
ESC B	HCUD	Cursor Down
ESC C	HCUF	Cursor Forward, stays on same line
ESC D	HCUB	Cursor Backward, stays on same line
ESC E	HCD	Clear display
ESC F	HEGM	Enter Graphics mode
ESC G	HXGM	Exit Graphic mode
ESC H	HCUH	Cursor Home
ESC I	HRI	Reverse Index
ESC J	HEOP	Erase to end of page
ESC K	HEOL	Erase to end of line
ESC L	HIL	Insert line
ESC M	HDL	Delete line
ESC N	HDCH	Delete character
ESC O	HERM	Exit Insert Char mode
ESC Y row col	HDCA	Direct cursor addressing, offset from space
ESC Z	HID	Identify (response is ESC / K which is a VT52)
ESC b	HBD	Erase Beginning of display
ESC j	HSCP	Save cursor position
ESC k	HRCP	Set cursor to saved position
ESC l	HEL	Erase entire line
ESC n	HCPR	Cursor Position Report request
ESC o	HEBL	Erase beginning of line
ESC p	HERV	Enter Reverse Video mode
ESC q	HXRV	Exit Reverse Video mode
ESC r Bn	HMBR	*Modify baud rate - not supported
ESC t	HEKS	*Enter Keypad shifted mode, not supported
ESC u	HXKS	*Exit Keypad shifted mode, not supported
ESC v	HEWA	Wrap around at end of line
ESC w	HXWA	Discard at end of line
ESC x Ps	HSM	Set Mode. See table below
ESC y Ps	HRM	Reset Mode. See table below

Ps	Mnemonic	Mode	Set (x)	Reset (y)
1	HSM/HRM	25th line	enabled	+disabled
2		*keyclick	off	on
3		*holdscreen	enabled	disabled
4		cursor type	block	underline
5		cursor on/off	on	off
6		*keypad-shifted	shifted	unshifted
7		alt app keypad	enabled	disabled
8		*linefeed	lf=>cr/lf	lf=>lf
9		newline mode	cr=>cr/lf	cr=>cr

+ disabling the 25th line also clears it

Escape seq	Mnemonic	Description of action
ESC z	HRAM	Reset to power-up configuration
ESC =	HAKM	Enter Alternate Keypad mode
ESC >	HXAM	Exit Alternate Keypad mode
ESC <	HEAM	Enter ANSI mode (ESC [stuff)
ESC @	HEIM	Enter Insert Char mode
ESC [	HEHS	*Enter Hold Screen mode, not supported
ESC \	HXHS	*Exit Hold Screen mode, not supported
ESC { and }	HEK, HDK	*Keyboard enable/disable, not supported
ESC]	HX25	*Transmit 25th line, not supported
ESC #	HXMP	*Transmit page, not supported

5.16.6. Heath-19 Functions While in ANSI Mode

```
Escape Seq     Mnenonic       Description of Action

ESC [ s        PSCP           Save cursor position & attributes
ESC [ u        PRCP           Restore cursor position & attributes
ESC [ z        PRAM           Reset to power-up configuration
ESC [ 2 J      ED             Erase entire screen but do not move cursor;
                              regular Heath-19 moves cursor to Home.
ESC [ ? 2 h    PEHM           Revert to normal Heath-19 non-ANSI mode
ESC [ > Ps h   SM             Same as ESC x Ps
ESC [ > Ps l   RM             Same as ESC y Ps
```

Plus most of the ANSI escape sequences listed for the VT102.

5.16.7. Tektronix 4010/4014 Graphics Terminal Functions

MS-Kermit's Tektronix 4010 emulator responds to ordinary text, several special control codes (for drawing lines and dots), and several escape sequences, as shown in Table 5-10. The commands SET DEBUG and SET TRANSLATION INPUT are effective in Tek mode.

Control Code		Action
FS, Control-\	Backslash	draw dots
GS, Control-]	Right square bracket	draw lines
RS, Control-^	Caret	Draw dots incrementally
US, Control-_	Underscore	Display text
BEL, Control-G		Beep, make a noise
BS, Control-H		Backspace, non-destructive
HT, Control-I		Tab, convert to single space
LF, Control-J		Line feed, go down one line
VT, Control-K		Move up one text line
FF, Control-L		Clears the screen
CR, Control-M		Carriage return, start of line
CAN, Control-X		Exit Tek sub-mode, or ignore
DEL, RUBOUT		Delete code, same as BS

Escape Sequence	Action
ESC Control-E	Send a status report, turn on Bypass mode
ESC Control-L	Clear the screen (enter sub-mode from VT102)
ESC Control-X	Turn on Bypass mode
ESC Control-Z	Activate crosshairs (GIN mode) and Bypass mode
ESC Z	Send terminal identification
ESC ' (accent grave)	Use solid lines in drawing
ESC a through ESC e	Use dashed line patterns: a=fine dots, b=short dashes c=dash dot, d=long dash dot e=dash dot dot.
ESC [Pn ; Pn m	Set ANSI colors. Same as for VT102.
ESC [? 3 8 l	Exit Tek mode (become text terminal, VT102 etc)
ESC [? 3 8 h	Enter Tek mode (from VT102 mode)

Table 5-10: Response of MS-Kermit Tektronix Emulator to Received Characters

In the table, US is the name for the ASCII character Control-Underscore, 31 decimal. Text is written starting with the last drawn point being the lower left corner of the first 8 by 8 character cell. The drawing position is updated by 8 dots to the right for each character, and lines wrap at column 80 (column 90 for Hercules boards). If text extends "below the screen" the sign "`More >`" is shown at the bottom right corner and the user needs to press a key to continue. Then the screen will be cleared and the new text will start at the top of the screen (no scrolling is done in graphics mode). A real Tek 4010 begins new text at column 40 and will overwrite dots from older material. The high resolution EGA screen and the Hercules screen will hold 43 lines, the CGA and Monochome screens hold 25 lines, and the AT&T screen holds 50 lines. Hercules screens are 90 characters wide and others are 80 characters wide. Monochrome systems lack graphics so the text is the normal hardware character font placed at the nearest normal 80x25 location (similarly, "drawing" on Monochrome systems is achieved by using a text plus ("+") sign where a dot would appear). Text mode is interrupted by the drawing commands discussed below.

<u>Bypass Mode</u>:

Certain Tektronix commands turn on or off "Bypass" mode whereby incoming text is not displayed on the screen. Removal of echos of the GIN mode, discussed below, is the major use of Bypass. Bypass mode is turned on by receipt of ESC Control-E, ESC Control-X, and ESC Control-Z and it is turned off upon receipt of BEL, LF, CR, US, other escape sequences, and resetting the terminal.

<u>Drawing commands GS, FS, RS</u>:

1. Draw a line or move to a point: GS <*xy xy . . . xy*>

GS is the name for ASCII character Control-`]` (right square bracket), decimal 29. <*xy*> stands for an encoded x,y coordinate as explained below. One or more x,y coordinates may follow GS and line segments are drawn from point to point. The first point is reached without drawing so that GS and the initial <*xy*> is a simple "move-to" command rather than a "draw-to" command. Lines may be constructed from six dash patterns described in Table 5-10. <*xy*> coordinates are encoded by separating the 10 bit value of x and of y into 5 bit components and then adding two high bits to each to identify which component is being represented: high-y, low-y, high-x, or low-x. They are transmitted in that order, with the low-x byte always sent last. In fact, bytes may be omitted if they do not change from point to point, provided that low-x is always sent. These bytes range from ASCII space (32 decimal) to ASCII DEL (127 decimal). Details are given below, and summarized in Table 5-12. This mode completes when a new command or a CR LF (carriage return, line feed) arrives; escape sequences are processed transparently but other control codes are ignored. The interrupting character is accepted and processed next.

2. Draw dots at given locations: FS <*xy xy . . . xy*>

FS is the name for the ASCII character Control-\ (backslash), decimal 28. <*xy*> is in the same form as above. A dot is drawn at each x,y point. This mode completes when a new command or a CRLF character arrives; escape sequences are processed transparently but other control codes are ignored. The interrupting character is accepted and processed next.

3. Draw dots from the current location: RS <*pen*> <*direction*> <*direction*> . . . <*direction*>

RS is the name for the ASCII character Control-`^` (caret), decimal 30. *pen* is the character Space (32 decimal) to move without drawing or P (80 decimal) to draw while moving. <*direction*> is one of the letters A, E, D, F, B, J, H, I as shown in Table 5-11.

Example: `RS P J J J` (no spaces here, naturally) means draw three dots in the southwest direction, stepping to each in turn. This mode completes when a new command or a non-<*pen*> or non-<*direction*> character arrives; the interrupting character is accepted and processed next.

direction	*Move One Tek Dot This Way*				
A	East (right)				
E	East and North	`F`	`D`	`E`	
D	North (up)				
F	North and West	`B`	`*`	`A`	(`*` is current location)
B	West				
J	South and West	`J`	`H`	`I`	
H	South				
I	South and East				

Table 5-11: Tektronix Dot-Drawing Commands

Graphics INput (GIN) mode:

Graphics input mode is entered when ESC Control-Z is received. A crosshair is drawn on the screen and may be moved by the numeric keypad arrows (fine scale motion) or the Shift key and these arrows (coarse scale motion). Pressing an ASCII-producing key sends the position of the crosshairs to the host as the sequence of: pressed key, X coordinate, Y coordinate, carriage return, then removes the crosshairs, and then returns to text mode. The coordinates are encoded by splitting them into five bit fields, adding an ascii space (20H) to each, and are sent as high-y, low-y, high-x and low-x bytes. Bypass mode is active while the report is sent to supress echos of the report. One may prematurely exit GIN mode by typing Control-C or Control-Break. Shift-PrtSc (DOS screen dump) remains active, however.

Status or Position Report:

ESCAPE Control-E requests a status report from the emulator. Tek terminals have many sub-fields. Kermit-MS sends a byte of 24 hex for being in text mode or 20 hex otherwise, followed by the encoded X then Y coordinates and a carriage return. Coordinates are encoded 5 bits at a time similar to the GIN report.

Identification Report:

ESCAPE Z requests terminal identification, as for VT52 and VT102. Currently this report is the 10 character sequence `IBM_TEK ESCAPE / Z` (no spaces).

Screen Capturing:

Kermit does not implement a graphics screen capture facility. There are many such Terminate-and-Stay-Resident (TSR) programs in circulation, as either public domain offerings or parts of commercial packages (Paint programs and even `GRAPHICS.COM` from DOS). High resolution EGA screens require more than the `GRAPHICS.COM` program. MS Windows tells the program (Kermit-MS) the system is using a pure text-only monochrome adapter so dots are shown as plus signs.

Although Kermit cannot save graphics screens directly (e.g. via the `^]F` connect-mode command), the received Tektronix escape sequences can still be logged to a PC file using the LOG SESSION command. The resulting log cannot be "played back" directly on the PC, but it can be transferred to the host and run through Kermit's Tek emulator again, just like a character-mode Kermit session log.

VGA Modes:

Considerable effort went into ensuring the graphics display would work automatically and not damage monitors. Thus, Kermit-MS safely tests the active display adapter for its kind and capabilities before starting graphics mode. Recent VGA and EGA+ display boards are capable of the 640 by 480 scan-line 16-color "VGA" mode which is now available on IBM PS/2 computers. The Tek emulator will happily run with 480 scan lines, but: the normal 256KB of video memory is sufficient to save only the top 407 lines of the graphics image. So activating this higher resolution mode is accomplished by the command SET TERMINAL GRAPHICS VGA and is not done automatically (the VGA is used in EGA mode). The 320 by 200 line by 256 color MCGA mode has too coarse a resolution for graphics line drawing and is not supported by Kermit.

Coordinate Encoding:

Coordinate 0,0 is the lower left corner and the X axis is horizontal. Tektronix positions are mapped into the typically 640 dots wide by 200 or 350 dots high PC screen and thus adjacent Tek positions may yield the same PC screen dot.

4010-like devices use positions from 0 to 1023 for both X and Y, although only 0 to 779 are visible for Y due to screen geometry. The Tek screen is 10.24 by 7.80 inches and coordinates are sent as 1-4 characters.

4014-like devices use positions 0 to 4095, but each movement is a multiple of 4 positions unless the high-resolution LSBXY are sent. This makes it compatible with the 4010 in that a full sized plot fills the screen. The emulator accepts the LSBXY components but does not use them.

The various modes are summarized in Table 5-12, in which the following notation is used:

HIX, HIY = High order 5 bits of a 10 or 12 bit position.
LOX, LOY = Middle order 5 bits of position (low order of Tek 4010).
LSBXY = Low order 2 bits of X + low order 2 bits of Y (4014 mode),
recognized by the Tek emulator but not used to calculate position.

Hi Y	Lo Y	Hi X	LSBXY	Characters Sent (Lo-X Always Sent)					
Same	Same	Same	Same					Lo-X	
Same	Same	Same	Diff		LSB,	Lo-Y,		Lo-X	4014
Same	Same	Diff	Same			Lo-Y,	Hi-X,	Lo-X	
Same	Same	Diff	Diff		LSB,	Lo-Y,	Hi-X,	Lo-X	4014
Same	Diff	Same	Same			Lo-Y,		Lo-X	
Same	Diff	Same	Diff		LSB,	Lo-Y,		Lo-X	4014
Same	Diff	Diff	Same			Lo-Y,	Hi-X,	Lo-X	
Same	Diff	Diff	Diff		LSB,	Lo-Y,	Hi-X,	Lo-X	4014
Diff	Same	Same	Same	Hi-Y,				Lo-X	
Diff	Same	Same	Diff	Hi-Y,	LSB,	Lo-Y,		Lo-X	4014
Diff	Same	Diff	Same	Hi-Y,		Lo-Y,	Hi-X,	Lo-X	
Diff	Same	Diff	Diff	Hi-Y,	LSB,	Lo-Y,	Hi-X,	Lo-X	4014
Diff	Diff	Same	Same	Hi-Y,		Lo-Y,		Lo-X	
Diff	Diff	Same	Diff	Hi-Y,	LSB,	Lo-Y,		Lo-X	4014
Diff	Diff	Diff	Same	Hi-y,		Lo-Y,	Hi-X,	Lo-X	
Diff	Diff	Diff	Diff	Hi-y,	LSB,	Lo-Y,	Hi-X,	Lo-X	4014
Kind code for byte:				20h	60h	60h	20h	40h	
				(transmitted left to right)					

Table 5-12: MS-Kermit Tektronix Coordinate Interpretation

Note that LO-Y must be sent if HI-X has changed so that the Tektronix knows the HI-X byte (in the range of

20h-3Fh) is HI-X and not HI-Y. LO-Y must also be sent if LSBXY has changed, so that the 4010 will ignore LSBXY and accept LO-Y. The LSBXY byte is

$60h+(MARGIN\times 10h)+(LSBY\times 4)+LSBX$

MARGIN is 0 here and refers to splitting the screen left and right for text rollover, which the Kermit Tek emulator does not do.

Tek 4010 Example:

Suppose *<xy>* is point y = 300, x = 500 in Tektronix coordinates. Split each 10-bit coordinate into 5-bit groups, add add the Kind code to each. Send the X part last.

```
                    HI-Y     LO-Y                         HI-X     LO-X
    Y=300d=012Ch=  01001    01100   X=500d=01F4h=  01111    10100
      +Kind code +100000 +1100000     +kind code +100000 +1000000
    Binary        101001 01101100                101111  1000100
    ASCII            )       l                      /       D
```

So *<xy>* = (500,300) is sent or received in a GS command as ‘‘`)l/D`’’. An example in C (program fragments):

```
#define ESC  27
#define GS   29
#define US   31
FILE *fp;                     /* File descriptor for terminal */
 . . .
    fputc( GS, fp); coord( 75, 65);       /* Move to 75,65      */
    fputc( ESC, fp); fputs("[31m", fp);   /* Set foreground to red */
    fputc( US, fp); fputs("A House", fp); /* Annotate at 75,65  */
    fputc( ESC, fp); fputs("[33m", fp);   /* Set foreground to yellow */
    fputc( GS, fp);                       /* Now draw lines...  */
    coord( 50, 50); coord(300, 50);       /* Bottom side        */
    coord(300,200); coord( 50,200);       /* Right wall, top    */
    coord(175,250); coord(300,200);       /* Roof               */
    fputc( GS, fp);                       /* Start a new line   */
    coord( 50, 50); coord( 50,200);       /* Left wall at 50,50 */
    fputc( ESC, fp); fputs("[37m", fp);   /* Set foreground to white */
    . . .
coord(x, y) int x, y; {       /* Send x,y coordinates to Tek 4010 */
    fputc((y / 32) + 32, fp);             /* High y */
    fputc((y % 32) + 96, fp);             /* Low  y */
    fputc((x / 32) + 32, fp);             /* High x */
    fputc((x % 32) + 64, fp);             /* Low  x */
}
```

5.17. IBM PC Kermit Technical Summaries

Under normal circumstances, MS-Kermit takes advantage of the computer’s hardware, and often bypasses DOS (sometimes even BIOS) to achieve high performance, to exercise special machine features, or to produce an attractive screen display. Thus, it is not in all respects a "well behaved" DOS program.

MS-Kermit redirects interrupts 0BH (COM2/4) or 0CH (COM1/3), 14H (serial port), 23H (Control-Break), 24H (DOS Critical Error) and returns them when done. It uses the BIOS for keyboard, video display, and system information interrupts. It examines segment 40H for EGA operating modes and it does direct screen reads and writes. Memory for the screen roll backbuffer is negotiated with DOS to leave room for a second copy of `COMMAND.COM` to run tasks within Kermit; about 100KB to 148KB is needed for the entire program. Video page zero is normally used, but page one is employed to save screens with non-standard dimensions. Hercules and other graphics mode displays are supported only in Tektronix terminal mode. Kermit’s timing delays are dynamically adjusted each time the serial port is started to accomodate machines of different speeds; duration of the normal software timing loop is measured with the hardware timer chip and looping is adjusted to produce uniform delays on 8088 through 80386 machines.

5.17.1. Kermit-MS/IBM on Local Area Networks

The IBM version of Kermit-MS has support for the IBM Local Area Network NetBIOS (and emulators) interface, Interrupt 5CH, with additional support for selected vendor specific features (presently just AT&T STARLAN), activated by the SET PORT NET command, described above, direct support for the Ungermann Bass Net One proprietary Interrupt 14h interface, and via SET PORT BIOSn support for many other networks which intercept the Bios serial port interrupt 14h. Communications across a LAN occurring through the NetBIOS interface use virtual circuits (Sessions), named nodes, and conventional NetBIOS packets. Kermit-MS does not use LAN terminal interface packages nor the Redirector or similar functions.

Kermit LAN operations are harmonious with normal network activity and many pairs of Kermits can communicate simultaneously. Kermit does not use LAN File Server functions, since these are proprietary and vendor-specific. Kermit can, however, send and receive files to/from a LAN file server.

Since Kermit uses the standard NetBIOS interrupt 5CH interface, it will run on most LANS including IBM PC Net, IBM Token Ring, AT&T STARLAN, and many others, and will run with Novell NetWare software. Presently, Kermit knows some details of STARLAN and is able to send a BREAK across the net and can use ISN node names with long path parts. If STARLAN is not operating these features are not available. As more detailed information becomes available special features of other networks can be built-in.

The sequence of operations is similar for a client or server Kermit. The SET PORT NET command is issued by both. This command causes Kermit to validate the presence of the Interrupt 5CH interface, test for vendor additions, test for a session already underway, establish and display a unique Kermit node name, but not make a network session. The node name of the remote server machine follows the word NET; this is not to be confused with our own node name discussed below.

If an earlier LAN session is still active then the current remote node name field of the command is examined for presence of a name. If a name is given then Kermit asks the user whether to RESUME the session or start a NEW one. Starting a new one results in Kermit hanging up the old session (HANGUP) before proceeding; resuming an old one requires no further work at this point.

When Kermit attaches to the network for the first time it needs to select a unique local node name so that two systems can form a Session by using these names as addresses. Kermit uses a simple algorithm to make the name. Kermit probes the network adapter board/software for the name of the local system. If the name is present Kermit makes its own name by appending a dot K (.K) to the local name. If the local name is absent then Kermit first tries a standard name of "mskermit.K"; should the network report that the name is not unique (another node is using the name) then the user is asked to choose a name. This process continues until a unique name is obtained or the user decides to quit. The final Kermit node name is reported on the screen; client Kermits will need to know the name of the server Kermit.

Communication across the LAN begins differently for client and server Kermits. The server must be started first, by simply placing a Kermit in server mode. This results in a network Listen request being posted so that arriving packets with the correct node name can be delivered to the server Kermit. Next, a client Kermit tries to connect to the server by issuing a Kermit server command to the proper node name (as given in the client's SET PORT NET node command); REMOTE WHO is a satisfactory choice. The client machine actually issues a network Call to the server's node name to make a connection and then follows it with data packets holding the Kermit server request. The initial exchange of packets establishes a particular virtual circuit between the two nodes. If the connection cannot be started then the client Kermit reports this fact to the user. The most common causes of a failure at this point are:

1. The client Kermit did not specify the correct server Kermit node name (spelling errors, wrong case for

letters, missing dot K),

2. One or both machines are using a network adapter board which is not the first in the machine; Kermit uses only the first board,

3. The LAN NetBIOS emulator does not fully support IBM standard virtual circuits,

4. The server machine was not started on the network before the client.

A virtual circuit will be broken if a sender or receiver gets no response to a request within a short time interval set by the LAN hardware/software. However, the LAN procedures within Kermit automatically reestablish the circuit transparently to the user when new information is communicated; the last used remote node name is remembered internally for this purpose. This also means the server Kermit will respond to a connection from a new client Kermit if the first client is idle for say a minute or so. A session can be terminated by the user by issuing the HANGUP command or by exiting Kermit. A session will not be broken this way if the user on the client Kermit changes to a regular serial port.

Finally, when Kermit returns control to DOS, but not via the PUSH command, its unique Kermit node name is removed from the network adapter board.

During network communications Kermit uses network packets holding 256 bytes of data. If both Kermits are given the command

```
SET RECEIVE PACKET 1000
```

then the network and Kermit will be used to best efficiency. Experience has shown that the client Kermit should have its TIMER OFF because the server may be asked to do an operation via DOS which does not complete before the client side would timeout. An observation of some token passing networks indicates that Kermit packets slightly longer than 256, 512, etc bytes result in marked slowing down because the remaining small piece is not sent until a net timer expires. Carrier sense (Ethernet, STARLAN) boards seem to be more agressive and export small packets immediately.

Support for the Ungermann-Bass Net/One network, with its NET Command Interface (NETCI), was contributed by Renne Rehmann and Henrik Levkowetz. In addition to the SET PORT NET [nodename] command, which may be used to connect to other nodes on the net with the standard NetBIOS calls, NETCI provides the means to connect directly to serial ports on the Ungermann-Bass network. Use SET PORT UB-Net1 and enter Connect mode. The NETCI prompt, >>, should appear and all the usual NETCI commands (connect, get, list, resume, abandon, examine, identify, set, logout, quit) may be selected. This line is disconnected when Kermit exits. However, the line may be put on hold, exit Kermit, then later restart Kermit and give the SET PORT UB-Net1 and CONNECT commands, and Resume the line.

Some LANs intercept the normal serial port Bios interrupt 14H and masquerade as a modem. This service can be engaged within Kermit by the SET PORT BIOSn command, where n is 1, 2, 3, or 4, as appropriate for the LAN software. To work properly the LAN must support the same use of registers as the system Bios. Several X.25 and TCP/IP packages have been operated successfully with the SET PORT BIOSn command. Since this channel appears to Kermit as a simple software level serial port, Kermit provides neither interrupt driven i/o nor LAN session support.

Kermit can access files on the LAN file server via DOS even while using the LAN as a communications medium. Network administrators should note this point because a user operating Kermit in Server mode can allow his or her file server directories to be available to other network users also running Kermit, without additional security checking of the other users. The network drives visible to the Server Kermit can become devices available for Kermit-to-Kermit file transfers, etc, unless the DISABLE command is used to confine access to the current disk and directory. A corollary is when files are accessible to DOS commands they can become public.

5.17.2. Use of Kermit-MS with External Device Drivers

It is often desirable to supplement or modify the behavior of a DOS program by loading it with special external device drivers. These drivers may operate at either the DOS or BIOS level. When Kermit-MS accesses the BIOS directly, DOS-level drivers are ineffective. When Kermit accesses the hardware directly, both the DOS and the BIOS level drivers are bypassed. Kermit-MS provides several mechanisms to allow these external drivers to operate as intended.

Here are a few examples:

- IBM's `ANSI.SYS` console driver operates at the DOS level. It allows the major IBM PC keys to be redefined, and also interprets selected ANSI-format escape sequences for screen control. It works fine at Kermit-MS command level, except SHOW KEY does not recognize strings assigned to keys via `ANSI.SYS`, and fine at CONNECT level. To use `ANSI.SYS` at CONNECT level, issue the Kermit-MS commands SET KEY OFF (to read keys via DOS) and SET TERMINAL NONE (to display characters through DOS).

- Blind people often have speaking or Braille machines attached to their PCs. DOS-level device drivers are generally used to redirect screen output to these devices, which works OK at DOS or MS-Kermit command level. SET TERMINAL NONE will allow this redirection to take place during CONNECT. But these devices also need to have the computer's output appear as a coherent stream of text, so users should also take care to inform the remote host to format its output for a "dumb" or hardcopy terminal. In addition, Kermit-MS' normal file transfer display does not mesh well with these devices, but that can be remedied using SET DISPLAY SERIAL.

- People with motor impairments may be using special keyboard replacements supported by DOS-level device drivers. As with `ANSI.SYS`, Kermit-MS may be directed to use such keyboard drivers with the command SET KEY OFF.

- Other keyboard drivers are available that work, like Kermit-MS, at BIOS level. Examples include ProKey and SuperKey. These may be used at DOS or Kermit-MS command level as well as during CONNECT.

- Conceivably, drivers exist that allow DOS communication programs to emulate terminals other than ANSI. You should be able to use them, if they exist, in conjunction with Kermit-MS by telling Kermit to SET TERMINAL NONE, but the speed may not be high because of the intervening DOS calls.

5.17.3. Kermit-MS/IBM Serial Port Information

Kermit-MS for IBM PC's and compatibles does testing of serial ports before use. This section describes those tests so users may understand what Kermit does.

When a serial port is selected by the SET PORT COMx command Kermit looks at low memory addresses in segment 40H assigned to hold the base address of each COMx port; COM1 is in word 40:0H, COM2 is in word 40:2H, and so on. If the value in the appropriate word is binary zero then Kermit declares the port to be unavailable. Otherwise, Kermit runs read-only (i.e., safe) tests at the base address to validate the presence of an official 8250 UART chip. If the tests fail Kermit indicates it will do i/o through the slow Bios pathway; some PC clones need to work this way even though the Bios has speed problems even at 1200 baud. Otherwise, interrupt driven i/o will be done through the 8250 UART (that is, very fast).

There is a special case when a communications board is present, set for COM2, but a normal COM1 serial port is not. Kermit detects this situation.

Many machines now have more than two serial ports, but until recently there has been no standard about addresses for COM3 and COM4. PC DOS 3.30 does not assign them either because it is really a problem of the system ROM Bios boot code run when the power is turned on. However, Kermit will use COM3 and/or COM4 if the base address of a port is placed in low memory words 40:4H (COM3) or 40:6H (COM4); the tests described above are then carried out. One restriction is that the Interrupt ReQuest number (IRQ in the serial port board manual) must be either IRQ4 or IRQ3. Kermit attempts to locate which line is correct with a short test. If the test is not successful it uses the IRQ4 for COM3 (and for COM1) and IRQ3 for COM4 (and for COM2) on the PC/AT, and on the PS/2 it uses IRQ3 for COM2, COM3, and COM4. Check the board and its manual. DOS utility DEBUG can be used to create a short program to insert the board's addresses into the segment 40H memory locations; a sample program is given below.

Serial Port	Address	IRQ Line	Conventions
COM1	03F8H	4	IBM standard
COM2	02F8H	3	IBM standard
COM3	?	4 (3 for PS/2)	Board
COM4	?	3	Board

Table 5-13: IBM PC/XT/AT Serial Port Numbers

The addresses shown as query marks are to be found in the board's reference manual; values such as 2E8H and 2E0H would be common. However, there is no standard for anything to do with COM3 and COM4 on non-PS/2's.

Assuming that you have selected an address in harmony with the rest of the system (good luck on that part), set the board's switches or jumpers, and use DEBUG to insert the address(es) in segment 40H memory. The example below creates a small program named `SETCOM3.COM` to put address 02E8H into the memory word 40:04H for COM3 and writes the program to drive A. (Disregard the xxxx items below):

```
A> DEBUG                         don't type these comments
-n a:setcom3.com                 sets name of output file
-a                               assemble command
xxxx:100 mov ax,40               value 40h
xxxx:103 mov es,ax               put it into register es
xxxx:105 mov ah,02               the 02 part of 02E8H
xxxx:107 mov al,e8               the E8 part of same
xxxx:109 es:
xxxx:10A mov [4],ax              store in 40:4 for com3 ([6] for com4)
xxxx:10D int 20                  return to DOS
xxxx:10F                         blank line to end assemble mode
-r cx                            show contents of register cx
CX 0000
: 0f                             set register cx to write 0fh bytes
-w                               write material to the disk file
-q                               quit debug
A> DEBUG setcom3.com
-u                               unassemble to see if all is well
-q                               quit debug
```

Note, for COM4, use `[6]` above rather than `[4]`, and of course employ your board's port address in place of 02E8H (check the manual). Finally, try it:

```
A> setcom3                       run the program
A> DEBUG                         now see what's down there
-d 40:00                         display bytes in seg 40H

       ( Shows many bytes.  See yours?  Good. )

-q
A>
```

A small side effect noted in practice is the first time the extra port is used there may be garbage from it. Just return to the Kermit prompt and try again, if necessary SET PORT to the other COM lines momentarily, all should be well the second time.

More technical comments, for those with an interest. When Kermit finishes with a port it disables interrupts for that serial port and returns the IRQ signal line to its state found when Kermit started since many devices can share the same Interrupt ReQuest line but only one device at a time can be active on it. If you find that transmissions are good but there is no reception then another device has stolen the IRQ; disable it or find a guru. Kermit will work with non-standard addresses for COM1 and COM2 but the IRQ's must be as in the table above. Accessing a non-existent port produces a message and all communications are discarded safely in the bit bucket.

5.17.4. CTTY COMx for IBM Machines

The DOS command CTTY COMx redirects the standard input and output from the keyboard and screen, respectively, to the indicated communications channel. If a Kermit Server is operated this way, "through the back port", then both DOS and Kermit can access the port hardware simultaneously; a deadlock develops on IBM machines. The items below refer to only the IBM version of Kermit-MS.

Kermit-MS/IBM version 2.31 successfully resolves the deadlock in the following manner. When Kermit requires the serial port it also attaches itself to Interrupt 16H, the Bios RS232 serial port routine. Code within Kermit receives the DOS serial port requests via Interrupt 14H and either passes the request to the Bios if the COM line is not that used by Kermit or it handles the request internally for conflicting situations. When the same port is used by both DOS and Kermit, Kermit discards DOS output material (typically a prompt, but could be the dreaded Abort, Retry, Ignore message) and returns a success code to DOS, it returns an ascii Backspace code to DOS read requests (this is a key item to keep DOS complacent while Kermit communicates), and it returns reasonable status for modem status. The interception ceases when Kermit releases the port, such as when the Kermit prompt is displayed, and this lets DOS converse out the serial port.

It is worth restating that a large number of programs bypass DOS to achieve higher performance. When such programs are started through the back door they may still require input from the real keyboard and will hang, waiting for it. There is nothing to do about this situation except a) don't let it happen, b) contact the local operator to push some keys.

5.17.5. Screen Sizes and the EGA Board, IBM Versions

Support has been included for Enhanced Graphics Adapter (EGA) video display boards which can be configured for other than the standard 80 columns by 25 lines, say 132 columns or 43 lines or other. Several boards, the Tseng Labs EVA (also Orchid Designer) board with the 132 column kit installed, the ATI EGA Wonder, the Video 7 Deluxe and VGA, and the Everex EV-659 (ega) and EV-673 (vga), can be controlled directly by Kermit for 80/132 column changes. Other boards need to be placed in the desired display mode by the user. Kermit then adapts to the settings if the board obeys standard rules for using the Bios EGA memory areas in segment 40H. The Video-7 boards have been used successfully in all screen sizes, including 132 columns by 43 lines, with an NEC Multisync monitor.

The IBM EGA board has several noteworthy bugs which are now standards. One is the cursor dots are not always on the correct scan lines when the number of screen lines is other than 25. Kermit-MS attempts to compensate for this attribute. Screen roll back space is fixed in size so there are fewer pages for more dense screens; standard screens use an internal buffer, non-standard screens use a buffer plus video page 1. `ANSI.SYS` is hard coded for 25 line displays so all DOS i/o will eventually overwrite itself on line 25; the emulator does not use DOS i/o. Commercial replacements for `ANSI.SYS` should be able to use all screen lines.

Screen dumps work correctly if done with Kermit commands. DOS PrintScreen may or may not, depending on your EGA board. Graphics dumps are not managed by Kermit.

When the VT102 receives escape sequences to change between 80 and 132 column modes the screen is reset and the ATI EGA Wonder, or Everex EV-659 (ega) or EV-673 (vga), Tseng Labs (and Orchid Designer), or Video 7 Vega or VGA board is asked to change modes (but only if that board is present); other display adapters are left in their current state. The right margin is enforced strongly so a board in 132 column mode will not display material to the right of column 80 if the emulator is in 80 column mode. Similarly, material to the right of column 80 is not preserved in the emulator if the display adapter is operating in 80 column mode; real VT102s keep that invisible material in hardware memory whereas the emulator does not.

Reference is made to line 25 in the emulator; this is normally the status/mode line in Kermit. Real VT102's have only 24 line displays. If the display adapter is set for a different number of lines per screen then the 25th line is interpreted to mean the bottom display adapter line, such as line 43. Should the host access the status/mode line then the line is declared to be disabled (same as SET MODE OFF) so that Kermit's own status information does not overwrite the host's when the screen is restored. Toggling a disabled mode line has no effect; only SET MODE ON will enable it again. The Heath-19 terminal has the unusual feature that disabling the mode line `(ESC y 1)` also clears it.

5.17.6. Kermit-MS/IBM Printer Control

The IBM PC MS-Kermit VT102 terminal emulator also supports full transparent printing of 8-bit binary bytes. The escape sequence ``ESC [ 5 i'' turns on transparent printing, in which all further 8-bit characters are sent directly to the printer, bypassing the SET TRANSLATION INPUT filter, and are not shown on the screen. Escape sequence ``ESC [4 i'' turns off transparent printing and the escape sequence is not sent to the printer. Non-transparent printing is controlled by the ``ESC [ ? 5 i'' and ``ESC [? 4 i'' sequences. Such printing simply duplicates text intended for the screen, excluding escape sequences. The text also appears on the screen.

Kermit-MS accesses the system printer through DOS calls several ways; neither the Bios nor the hardware are used. Files directed to the printer by the SET DESTINATION PRINTER command are written by opening a file with the name PRN (DOS's name for the system printer) and writing to it the same as to a disk file; DOS provides limited buffering. LOGging to device PRN works the same way, as can be noticed by the last line or so not being printed until the log file is CLOSED. DOS is used again while emulating a terminal in CONNECT mode. If the VT102 emulator found in the IBM PC is used for transparent or Controller printing, single characters are written to DOS file handle 4, the DOS standard print device. If the screen is echoed to the printer via the typical Control PrtSc key combination, or equivalent, single characters are written by the DOS function 05H Printer Output call. In both cases of terminal emulation the printer's ready status is found by the DOS IOCTL 44H call. Only the Control PrtSc case results in the PRN message being displayed on the status line. Finally, the classical IBM PC Shift PrtSc command to copy the whole screen to the printer is unknown to Kermit because the system Bios traps the key combination and does not tell Kermit about it. If the Control P command is given to DOS before Kermit starts then again characters are echoed by the system Bios without Kermit's knowledge; this situation can result in lost characters.

Print spoolers generally operate by being told an existing filename and then in the background they steal cpu cycles to read from disk and write to the printer. The DOS PRINT command invokes such a spooler. Although an active Kermit does not feed these software programs directly the spooler and Kermit can compete for cpu cycles and characters can be lost. If a non-DOS resident program intercepts characters destined for the printer device and spools them Kermit does not know about it and similar competion can occur.

During file transfers printing is carefully sequenced to occur only when the local Kermit is in control of the communications line so that a small pause will not result in missing characters arriving at the serial port. When terminal emulation is active then printing competes for cpu time with the serial port routines. Generally, the serial port wins such contests if the port is interrupt driven (Generic Kermit is not interrupt driven, so beware). However, the printing itself can use enough cpu cycles to delay processing of characters to the screen and eventually the receive buffer of the serial port fills to the high water mark and an XOFF flow control character is sent to the host to suspend further transmissions until we send an XON. If FLOW is NONE then expect lost characters at the serial port. Experience with ordinary IBM PC's through 80386 machines at very high baud rates indicates no characters are lost when FLOW is XON/XOFF. However, it is possible on some machines for the printer to have priority over the serial port, and hence to have lost characters, especially if a Terminate Stay Resident program intercepts characters destined for the printer and keeps interrupts turned off too long.

6. UNIX KERMIT

Program: Frank da Cruz, Bill Catchings, Jeff Damens, Columbia University; Herm Fischer, Encino CA; contributions by many others.
Language: C
Documentation:
Christine Gianone, Frank da Cruz
Version: `4E(070)`
Date: January 24, 1988

C-Kermit is an implementation of Kermit, written modularly and transportably in C. The protocol state transition table is written in *wart*, a (non-proprietary) lex-like preprocessor for C. System-dependent primitive functions are isolated into separately compiled modules so that the program should be easily portable among Unix systems and also to non-Unix systems that have C compilers, such as VAX/VMS, Data General AOS/VS, Apollo Aegis, the Apple Macintosh, and the Commodore Amiga. This document applies to Unix implementations of C-Kermit, and in most ways also to the VMS, Data General, and other implementations.

Unix Kermit Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
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 (most versions)
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
UUCP and multiuser line locking:	Yes
Long packets:	Yes
Sliding Windows:	No
File attributes packets:	No
Command macros:	No
Raw file transmit:	No

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

C-Kermit provides traditional Unix 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.

On systems with dialout modems, C-Kermit's command file, DIAL command, and login script facilities provide a counterpart to UUCP for file transfer with non-UNIX operating systems, including the use of scheduled (e.g. late night) unattended operation.

6.1. 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 characters, with the high-order bit off (0), and lines separated by the Unix newline character, which is linefeed (LF, ASCII 10). This distinguishes Unix text files from those on most other ASCII 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.

When transferring files, C-Kermit will convert 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 must be instructed not to perform LF/CRLF conversion (`-i` on the command line or "set file type binary" interactively; see below).

6.2. File Transfer

If C-Kermit is in local mode, 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:

Control-F: Interrupt the current File, and go on to the next (if any).

Control-B: Interrupt the entire Batch of files, terminate the transaction.
Control-R: Resend the current packet
Control-A: Display a status report for the current transaction.

These interrupt characters differ from the ones used in other Kermit implementations to avoid conflict with commonly used Unix shell interrupt characters. With Version 7, System III, and System V implementations of Unix, interrupt commands must be preceeded by the 'connect' escape character (e.g. normally-\). Ctrl-F and Ctrl-B are effective only during the transfer of data (D) packets, and cannot be used to interrupt a transfer that has not yet reached that stage.

> *CAUTION:* If Control-F or Control-B 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.

EMERGENCY EXIT: When running Unix Kermit in remote mode, if you have started a protocol operation (sending or receiving a file, server command wait, etc), you will not be able to communicate with the terminal in the normal way. In particular, you cannot stop the protocol by typing the normal Unix interrupt characters, since the terminal has been put in "raw mode". If you need to regain control quickly -- for instance, because the protocol is stuck -- 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 exit and restore the terminal to normal.

6.3. 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 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 Unix 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.

`-s` *fn* Send the specified file or files. If *fn* contains wildcard (meta) characters, the Unix shell expands it into a list. 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 christin | 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. 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 sets C-Kermit's "mode":

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

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

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

`-b` *n* Baud -- Specify the 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 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 commands may be used only with a C-Kermit which is local either by default or else because the `-l` option has been specified.

`-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.

`-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.

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

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

Files are sent with their own names, except that lowercase letters are raised to upper, pathnames are stripped off, certain special characters like ('`~`') and ('`#`') are changed to '`X`', and if the file name begins with a period, an '`X`' is inserted before it. Incoming files are stored under their own names except that uppercase letters are lowered, and, 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.
- 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.

```
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 orginating 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)
```

Exit Status Codes:

Unix Kermit returns an exit status of zero, except when a fatal error is encountered, where the exit status is set to one. With background operation (e.g., '&' at end of invoking command line) driven by scripted interactive commands (redirected standard input and/or take files), any failed interactive command (such as failed dial or script attempt) causes the fatal error exit.

6.4. 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 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.
- `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.

SP (Space) -- Delimits fields (keywords, filenames, numbers) within a command. HT (Horizontal Tab) may also be used for this purpose.

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, especially in the 'script' command.

^Z (Control-Z) -- On systems (like Berkeley Unix, Ultrix) with job control, 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.

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 the file .kermrc 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 Unix 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 a Unix shell command, or start a shell.
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	Instruct a 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 a log file -- debugging, packet, session, transaction.
quit	Same as 'exit'.
receive	Passively wait for files to arrive.
remote	Issue file management commands to 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 own command language.
space	Display current disk space usage on remote system.
type	Display a remote file on your screen.
who	Display 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 Unix Kermit commands are described below.

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 be 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 Unix 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 (meta) 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. Other notations for file groups, like '`[a-z]og`', are not available in interactive commands (though of course they are available on the command line). 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. If *fn* was of the form

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

then C-Kermit's string space will fill up rapidly -- try doing a cwd (see below) to the path in question and reissuing the command.

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 on page 137, or under System III/V Unix you can use cpio.

Another Note -- The 'send' command does not skip 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. Similarly for "temporary" files whose names start with "#".

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'.

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.exec
```

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>
```

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 Unix 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 who	Shows who's logged in
remote host	Executes a Unix shell command
remote help	Lists these capabilities

The Unix Kermit server cannot always respond properly to a BYE command. It will attempt to do so using "`kill()`", but this will not work on all systems or under all conditions because of the complicated process structures that can be set up under Unix.

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 have 'set file display off'; normally the program expects the console to be used to observe file transfers and enter status queries or interruption 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 "`&`".

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 or the Berkeley C-Shell.

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*	command in remote host's own command language.
remote space	disk usage report from 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.

The 'log' and 'close' commands:

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

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`.

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.

Local File Management Commands:

Unix Kermit allows some degree of local file management from interactive command level:

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

cwd [directory-name]
: Changes Kermit's working directory to the one given, or to the default directory if the directory name is omitted. 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.

`!` [*command*]
: The command is executed by the Unix shell. If no command is specified, then an interactive shell is started; exiting from the shell, e.g. by typing Control-D or '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:

- C-Kermit attempts to use your preferred, customary (login) shell.
- At least one space must separate the '!' from the shell command.
- A 'cd' (change directory) command executed in this manner will have no effect -- use the C-Kermit 'cwd' command instead.

The 'set' and 'show' Commands:

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. Used in remote mode to give you time to escape back to your local Kermit and issue a 'receive' command. Normally 5 seconds.

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 Unix command level. The normal escape character is Control-Backslash (28). The escape character is also used in System III/V implementations to prefix interrupt commands during file transfers.

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 (stdin) 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 console terminal.

  names {converted, literal}
  : Normally converted, which means that outbound filenames have 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. Incoming files 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.

  type {binary, text} [{7, 8}]
  : The file type is normally text, which means that conversion is done between Unix newline characters and the carriage-return/linefeed sequences required by the canonical Kermit file transmission format, and in common use on non-Unix systems. Binary means to transmit file contents without conversion. Binary ('`-i`' in command line notation) is necessary for binary files, and desirable in all Unix-to-Unix transactions to cut down on overhead.

    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 `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 would be deleted from the end first, for instance, `thelongestname` on a system with a limit of 14 characters would become `thelongestn~1`.

> *CAUTION:* If Control-F or Control-B 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. *Warning:* This command may have no effect on certain Unix systems, where Kermit puts the communication line into 'rawmode', and rawmode precludes flow control.

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 Unix 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 [device-name]
: The device name for the communication line to be used for file transfer and terminal connection, e.g. `/dev/ttyi3`. 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'. If you omit the device name, Kermit will revert to its default mode of operation. If you specify `/dev/tty`, Kermit will enter remote mode (useful when logged in through the "back port" of a system normally used as a local-mode workstation). 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. The method used for locking is the "uucp lock file", explained in more detail later.

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 doublequotes, 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.

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 Unix Kermit to use when waiting for packets from the other Kermit. If you specify 0, then no timeouts will occur, and Unix Kermit will wait forever for expected packets to arrive.

speed {0, 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600}

The baud rate for the external communication line. This command cannot be used to change the speed of your own console terminal. Many Unix systems are set up in such a way that you must give this command after a ’set line’ command before you can use the line. ’set baud’ is a synomym for ’set speed’.

terminal
Used for specifying terminal parameters. Currently, 'bytesize' is the only parameter provided, and it can be set to 7 or 8. It's 7 by default.

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.

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).

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 line /dev/tty17
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. The 'echo' command should not be confused with the Unix 'echo' command, which can be used to show how meta characters would be expanded. The Kermit echo 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. Of course, each backslash must be must be entered twice in order for it to be passed along to the echo command by the Kermit command parser.

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 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.

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 ttyh7 at 4800 baud, you could create a file called `b` containing the commands

```
% C-Kermit command file to connect to System B thru /dev/ttyh7
set line /dev/ttyh7
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 `.kermrc` file when C-Kermit starts up, upon either interactive or command-line invocation. The `.kermrc` 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 `.kermrc` file. On some non-Unix systems that run C-Kermit, the initialization file might have a different name, such as `kermit.ini`.

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 shell). When kermit is executed in the background, errors during execution of a take file are fatal.

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:

```
cmdprocess | kermit
```

The 'connect' Command:

The 'connect' command ('c' is an acceptable non-unique 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' 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 include:

c	Close the connection
b	Send a BREAK signal
0	(zero) send a null
s	Give a status report about the connection
h	Hangup the phone
`^\`	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. If special terminal emulation is desired, then the 'connect' command can invoked from the Unix command line (`-c` or `-n`), piped through a terminal emulation filter, e.g.

```
kermit -l /dev/acu -b 1200 -c | tek
```

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 the following modems:

- AT&T 7300 Internal Modem
- Cermetek Info-Mate 212A
- Concord Condor CDS 220
- DEC DF03-AC
- DEC DF100 Series
- DEC DF200 Series
- General DataComm 212A/ED

- Hayes Smartmodem 1200 and compatibles
- Penril
- Racal Vadic
- 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 ?`".

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). The "dial" command calls locks the path (see the section on line locking below) and establishes 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 one:

```
kermit -l /dev/cul0 -b 1200
C-Kermit>set modem-dialer hayes       hint: abbreviate set m h
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 unlocks line.

Example two:

```
kermit
C-Kermit>set modem-dialer ventel
C-Kermit>set line /dev/cul0
C-Kermit>dial 9&5551212%
Connected!
C-Kermit> ...
```

Example three:

```
kermit
C-Kermit>take my-dial-procedure
Connected!
```

file my-dial-procedure:

```
set modem hayes
set line /dev/tty99
dial 5551212
connect
```

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.

For Hayes dialers, two important switch settings are #1 and #6. Switch #1 should be normally be UP so that the modem can act according to your computer's DTR signal. But if your computer, or particular implementation of Kermit, cannot control DTR, then switch 1 should be DOWN. Switch #6 should normally be UP so carrier-detect functions properly (but put it DOWN if you have trouble with the UP position). Switches #2 (English versus digit result codes) and #4 (Hayes echoes modem commands) may be in either position.

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 to get back to command level.

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
~*o[o[o]]* an octal character
`~d` delay approx 1/3 second during send
`~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 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~saa001122 = 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 dial or script fail.
#
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 a bye and a quit.

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.

The 'exit' and 'quit' Commands:

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

- Attempt to insure that the terminal is returned to normal.
- Relinquish access to any communication line assigned via 'set line'.
- Relinquish any uucp and multiuser locks on the communications line.
- Hang up the modem, if the communications line supports data terminal ready.
- Close any open logs or other files.

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.

6.5. UUCP Lock Files

Unix has no standard 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`, on some systems `/etc/locks`) with a name like `LCK..`*name*, where *name* is the device name, for instance `tty07`.

Unix Kermit uses UUCP lock files in order to avoid conflicts with UUCP, tip, or other programs that follow this convention. 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        4 May  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, but will allow you to proceed at your own risk (and the risk of anyone else who might also be using the same line).

If no lock file is found, Unix Kermit will attempt create one, thus preventing anyone who subsequently tries to run Kermit, UUCP, 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 a warning message but will be allowed to proceed at your -- and everyone else's -- risk. When Kermit terminates normally, your lock file is removed.

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`).

It may be seen that line locking is fraught with peril. It is included in Unix Kermit only because other Unix communication programs rely on it. While it is naturally desirable to assure exclusive access to a line, it is also undesirable to refuse access to a vacant line only because of a spurious lock file, or because the uucp directory is not appropriately protected.

6.6. 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', 'cwd', 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. In this case the terminal is not restored. You will have to type Control-J followed by "reset" followed by another Control-J to get your terminal back to normal.

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.

6.7. C-Kermit on the DEC Pro-3xx with Pro/Venix Version 1

The DEC Professional 300 series are PDP-11/23 based personal computers. Venix Version 1 is a Unix v7 derivative. It should not be confused with Venix Version 2, which is based on ATT System V; these comments apply to Venix Version 1 only. C-Kermit runs in local mode on the Pro-3*xx* when invoked from the console; the default device is `/dev/com1.dout`. When connected to a remote system (using C-Kermit's 'connect' command), Pro/Venix itself (not Kermit) provides VT52 terminal emulation. Terminal operation at high speeds (like 9600 baud) requires xon/xoff flow control, which unfortunately interferes with applications such as the EMACS that use Control-Q and Control-S as commands.

When logging in to a Pro-3xx (or any workstation) through the "back port", it may be necessary to give the command "set line /dev/tty" in order to get C-Kermit to function correctly in remote mode (on a system in which it normally expects to be operating in local mode).

6.8. C-Kermit under VAX/VMS

C-Kermit can be built using VAX-11 C to run under VMS. Most of the descriptions in this manual hold true, but it should be noted that as of this writing the VMS support is not thoroughly tested, and no explicit support exists for the various types of VMS files and their attributes.

The C-Kermit init file for VMS is called `KERMIT.INI`.

6.9. C-Kermit on the Macintosh and other Systems

The "protocol kernel" of C-Kermit is also used by Columbia's Macintosh Kermit. The user and system interface is entirely different, and is covered in a separate document.

There is also a Kermit for the Commodore Amiga based on C-Kermit, as well as versions for MS-DOS, Data General operating systems, etc.

6.10. C-Kermit Restrictions and Known Bugs

1. Editing characters: The program's interactive command interrupt, delete, and kill characters are Control-C, Delete (or Backspace), and Control-U, respectively. There is currently no way to change them to suit your taste or match those used by your shell, in case those are different.

2. Flow control: C-Kermit attempts to use XON/XOFF flow control during protocol operations, but it also puts the communication line into "rawmode". On many systems, rawmode disables flow control, so even though you may have "set flow xon/xoff", no flow control will be done. This is highly system and Unix-version dependent.

3. High baud rates: There's no way to specify baud rates higher than 9600 baud. Most Unix systems don't supply symbols for them (unless you use EXTA, EXTB), and even when they do, the program has no way of knowing whether a specific port's serial i/o controller supports those rates.

4. Modem controls: If a connection is made over a communication line (rather than on the controlling terminal line), and that line has modem controls, (e.g. data terminal ready and carrier detection implementation), returning to the shell level will disconnect the conversation. In that case, one should use interactive mode commands, and avoid use of piped shell-level operation (also see 'set modem-dialer' and 'dial' commands.)

5. 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.

6. Dial-out vs dial-in communications lines: C-Kermit requires a dial-out or dedicated line for the "set line" or "-l" options. Most systems have some lines dedicated to dial-in, which they enable "loggers" on, and some lines available for dial-out. Recent releases of Unix (ATT & Berkeley) have mechanisms for changing the directionality of a line.

7. Using C-Kermit on Local Area Networks: C-Kermit can successfully operate at speeds up to 9600 baud over LANs, provided the network buffers are big enough to accommodate Kermit packets.

   When computers are connected to LAN's through asynchronous terminal interfaces, then the connection should be configured to do XON/XOFF flow control between the network interface and the computer, rather than passing these signals through transparently. This can help prevent Kermit from overrunning the LAN's buffers if they are small (or if the LAN is congested), and will can also prevent the LAN from overrunning a slow Kermit's buffers.

   If the network hardware cannot accept 100 characters at a time, and flow control cannot be done between the network and the computer, then Kermit's "set send/receive packet-length" command can be used to shorten the packets.

8. Resetting terminal after abnormal termination or kill: When C-Kermit terminates abnormally (say, for example, by a kill command issued by the operator) the user may need to reset the terminal state. If commands do not seem to be accepted at the shell prompt, try Control-J "stty sane" Control-J (use "reset" on Berkeley Unix). That should take the terminal out of "raw mode" if it was stuck there.

9. Remote host commands may time-out on lengthy activity: Using "remote host" to instruct the C-Kermit server to invoke Unix functions (like "make") that might take a long time to produce output can cause timeout conditions.

10. XOFF deadlocks: When connecting back to C-Kermit after a transaction, or after finishing the server, it may be necessary to type a Control-Q to clear up an XOFF deadlock. There's not much the program can do about this...

6.11. How to Build C-Kermit for a Unix System

The C-Kermit files, as distributed from Columbia, all begin with the prefix "ck". You should make a directory for these files and then cd to it. A makefile is provided to build C-Kermit for various Unix systems (there are separate makefiles for VMS and the Macintosh). As distributed, the makefile has the name "`ckuker.mak`". You should rename it to "`makefile`" and then type "make xxx", where xxx is the symbol for your system, for instance "make bsd" to make C-Kermit for 4.x BSD Unix. The result will be a program called "wermit". You should test this to make sure it works; if it does, then you can rename it to "kermit" and install it for general use. See the makefile for a list of the systems supported and the corresponding "make" arguments.

6.12. Adapting C-Kermit to Other Systems

C-Kermit is designed for portability. The level of portability is indicated in parentheses after the module name: "C" means any system that has a C compiler that conforms to the description in "The C Programming Language" by Kernighan & Ritchie (Prentice-Hall, 1978). "Cf" is like "C", but also requires "standard" features like printf and fprintf, argument passing via argv/argc, and so on, as described in Kernighan & Ritchie. "Unix" means the module should be useful under any Unix implementation; it requires features such as fork() and pipes. Anything else means that the module is particular to the indicated system. C-Kermit file names are of the form:

`ck`<*system*><*what*>`.`<*type*>

where the part before the dot is no more than 6 characters long, the part after the dot no more than 3 characters long, and:

<*type*> is the file type:

c:	C language source
h:	Header file for C language source
w:	Wart preprocessor source, converted by Wart (or Lex) to a C program
nr:	Nroff/Troff text formatter source
mss:	Scribe text formatter source
doc:	Documentation
hlp:	Help text
bld:	Instructions for building the program
bwr:	A "beware" file - list of known bugs
upd:	Program update log
mak:	Makefile

<*system*> is a single character to tell what system the file applies to:

a:	Descriptive material, documentation
c:	All systems with C compilers
d:	Data General
h:	Harris computers (reserved)
i:	Commodore Amiga (Intuition)
m:	Macintosh
p:	IBM PC, PC-DOS (reserved)
u:	Unix
v:	VAX/VMS
w:	Wart

<*what*> is mnemonic (up to 3 characters) for what's in the file:

aaa:	A "read-me" file, like this one
cmd:	Command parsing
con:	Connect command
deb:	Debug/Transaction Log formats, Typedefs
dia:	Modem/Dialer control
fio:	System-depdendent File I/O
fns:	Protocol support functions
fn2:	More protocol support functions
ker:	General C-Kermit definitions, information, documentation
mai:	Main program
pro:	Protocol
scr:	Script command
tio:	System-dependent terminal i/o & control and interrupt handing
usr:	User interface
us2:	More user interface
us3:	Still more user interface

Examples:

`ckufio.c`	File i/o for Unix
`ckmtio.c`	Terminal i/o for Macintosh
`ckuker.mss`	Scribe source for for Kermit User Guide chapter
`ckuker.nr`	Nroff source file for Unix C-Kermit man page

The following material discusses each of the C-Kermit modules briefly.

`ckcmai.c, ckcker.h, ckcdeb.h (Cf)`:

This is the main program. It contains declarations for global variables and a small amount of code to initialize some variables and invoke the command parser. In its distributed form, it assumes that command line arguments are passed to it via argc and argv. Since this portion of code is only several lines long, it should be easy to replace for systems that have different styles of user interaction. The header files define symbols and macros used by the various modules of C-Kermit. `ckcdeb.h` is the only header file that is included by all the C-Kermit modules, so it contains not only the debug format definitions, but also any compiler-dependent typedefs.

`ckwart.c (Cf), ckcpro.w (C)`:
: The ckcpro module embodies the Kermit protocol state table and the code to accomplish state switching. It is written in "wart", a language which may be regarded as a subset of the Unix "lex" lexical analyzer generator. Wart implements enough of lex to allow the ckprot module to function. Lex itself was not used because it is proprietary. The protocol module `ckcpro.w` is read by wart, and a system-independent C program is produced. The syntax of a Wart program is illustrated by `ckcpro.w`, and is described in `ckwart.doc`.

`ckcfns.c (C)`:
: The module contains all the Kermit protocol support functions -- packet formation, encoding, decoding, block check calculation, filename and data conversion, protocol parameter negotiation, and high-level interaction with the communication line and file system. To accommodate small systems, this module has been split into two -- `ckcfns.c` and `ckcfn2.c`.

`ckutio.c`:
: This module contains the system-dependent primitives for communication line i/o, timers, and interrupts for the various versions of Unix. Certain important variables are defined in this module, which determine whether C-Kermit is by default remote or local, what the default communication device is, and so forth. The tio module maintains its own private database of file descriptors and modes for the console terminal and the file transfer communication line so that other modules (like ckcfns or the terminal connect module) need not be concerned with them. The variations among Unix implementations with respect to terminal control and timers are accommodated via conditional compilation.

`ckufio.c`:
: This module contains system-dependent primitives for file i/o, wildcard (meta character) expansion, file existence and access checking, and system command execution for the various versions of Unix. It maintains an internal database of i/o "channels" (file pointers in this case) for the files C-Kermit cares about -- the input file (the file which is being sent), the output file (the file being received), the various logs, the screen, and so forth. This module varies little among Unix implementations except for the wildcard expansion code; the directory structure of 4.2bsd Unix is different from that of other Unix systems. Again, variation among Unix systems is selected using conditional compilation.

`ckuusr.h, ckuusr.c, ckuus2.c, ckuus3.c` (Unix):
: This is the "user interface" for C-Kermit. It includes the command parser, the screen output functions, and console input functions. The command parser comes in two pieces -- the traditional Unix command line decoder (which is quite small and compact), and the interactive keyword parser (which is rather large). This module is fully replacable; its interface to the other modules is very simple, and is explained at the beginning of the source file. The ckuusr module also includes code to execute any commands directly which don't require the Kermit protocol -- local file management, etc. The module is rated "Unix" because it makes occasional use of the `system()` function.

  Note that while `ckuusr` is logically one module, it has been split up into three C source files, plus a header file for the symbols they share in common. This is to accommodate small systems that cannot handle big modules. `ckuusr.c` has the command line and top-level interactive command parser; `ckuus2.c` has the help command and strings; `ckuus3` has the set and remote commands along with the logging, screen, and "interrupt" functions.

`ckucmd.c, ckucmd.h` (Cf):
: This is an interactive command parsing package developed for C-Kermit. It is written portably enough to be usable on any system that has a C compiler that supports functions like printf. The file name parsing functions depend upon primitives defined in the fio module; if these primitives cannot be supplied for a certain system, then the filename parsing functions can be deleted, and the package will still be useful for parsing keywords, numbers, arbitrary text strings, and so forth. The style of interaction is the same as that found on the DECSYSTEM-20.

`ckucon.c` (Unix):
: This is the connect module. As supplied, it should operate in any Unix environment, or any C-based environment that provides the fork() function. The module requires access to global variables that specify line speed, parity, duplex, flow control, etc, and invokes functions from the tio module to accomplish the desired settings and input/output, and functions from the fio module to perform session logging. No terminal

emulation is performed, but since standard i/o is used for the console, this may be piped through a terminal emulation filter. The ckucon function may be entirely replaced, so long as the global settings are honored by its replacement. PC implementations of C-Kermit may require the ck?con module to do screen control, escape sequence interpretation, etc, and may also wish to write special code to get the best possible performance.

`ckudia.c` (Unix):
: This is the dialer module. As supplied, it handles Hayes, Ventel, Penril, Racal-Vadic, and several other modems.

`ckuscr.c` (Unix):
: This is the login script module. As supplied, it handles uucp-style scripts.

Moving C-Kermit to a new system entails:

1. Creating a new `ck?tio` module in C, assembler, or whatever language is most appropriate for system programming on the new system. If the system is Unix-like, then support may be added within the `ckutio.c` module itself using conditional compilation.

2. Creating a new `ck?fio` module, as above.

3. If the system is not Unix-like, then a new `ckuusr` module may be required, as well as a different invocation of it from `ckcmai`.

4. If the distributed connect module doesn't work or performs poorly, then it may be replaced. For instance, interrupt-driven i/o may be required, especially if the system doesn't have forks.

Those who favor a different style of user/program interaction from that provided in `ckuusr.c` may replace the entire module, for instance with one that provides a mouse/window/icon environment, a menu/function-key environment, etc.

A few guidelines should be followed to maintain portability:

- Keep variable and function names to 6 characters or less. Don't use identifiers that are distinguished from one another only by alphabetic case.

- Keep modules small. For instance, on a PDP-11 it is necessary to keep the code segment of each module below 8K in order to allow the segment mapping to occur which is necessary to run programs larger than 64K on a non-I-and-D-space machine.

- Keep strings short; many compilers have restrictive maximum lengths; 128 is the smallest maximum string constant length we've encountered so far.

- Keep (f,s)printf formats short. If these exceed some compiler dependent maximum (say, 128) memory will be overwritten and the program will probably core dump.

- Do not introduce system dependencies into `ckcpro.w` or `ckcfn*.c`.

- If a variable is a character, declare as CHAR, not int, to prevent the various sign extension and byte swapping foulups that occur when characters are placed in integer variables.

- Remember that different systems may use different length words for different things. Don't assume an integer can be used as a pointer, etc.

- Don't declare static functions; these can wreak havoc with systems that do segment mapping.

- In conditional compilations expressions, use `#ifdef` and `#ifndef` and not `#if`, which is not supported by some compilers. Also, don't use any operators in these expressions; many compilers will fail to understand expressions like `#ifdef FOO | BAR`. Also, don't put trailing tokens on `#else`'s or `#endif`'s (use `/*` comments `*/`).

- Don't define multiline macros.

In general, remember that this program will have to be compilable by old compilers and runnable on small systems.

7. MACINTOSH KERMIT

Program:	Bill Catchings, Bill Schilit, Frank da Cruz (Columbia University), Davide Cervone (University of Rochester), Matthias Aebi (ECOFIN Research and Consulting, Ltd., Zuerich), Paul Placeway (Ohio State University).
Language:	C (MPW)
Documentation:	Christine Gianone, Frank da Cruz, Paul Placeway
Version:	`0.9(40)`
Date:	May 26, 1988

MacKermit Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes (server mode only)
Login scripts:	No
Transfer text files:	Yes
Transfer binary files:	Yes
MacBinary transfers:	No
Wildcard send:	Yes (whole HFS folders)
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 (VT100,VT102)
Communication settings:	Yes (Speed, Parity, Echo)
XON/XOFF:	Yes
Transmit BREAK:	Yes
IBM mainframe communication:	Yes
Transaction logging:	Yes
Session logging:	Yes
Debug logging:	No
Packet logging:	No
Act as server:	Yes
Talk to server:	Yes
Advanced server functions:	Yes
Local file management:	Yes
Command/Init files:	No
Key redefinition/macros:	Yes
File attributes packets:	No
Command macros:	No
Raw file transmit:	No
Long packets:	Yes
Sliding windows:	No

7.1. Introduction

Macintosh Kermit, or "MacKermit", is an implementation of the Kermit file transfer protocol for the Apple Macintosh family of computers. It was developed at Columbia University, based on C-Kermit (which also forms the nucleus of Unix Kermit and many other Kermit programs). Version 0.9 of MacKermit runs on the Macintosh 512, XL (Apple Lisa running MacWorks), 512e, Plus, SE, and II, under the regular Finder and the Multifinder, with which it can transfer files in the background. MacKermit 0.9 probably does not run on a 128k (original, classic) Macintosh, due to lack of sufficient memory, but should run OK on a "fat Mac" (a 128K Mac upgraded to 512K). Version 0.8 should be retained for 128K Macs.

This manual assumes you are acquainted with your Macintosh, and that you are familiar with the general ideas of data communication and Kermit file transfer. A very brief overview is given here, but for details consult the early chapters of the *Kermit User Guide* (of which this document is a chapter), or the book Kermit, A File Transfer Protocol, by Frank da Cruz, Digital Press (1987). For further information about Kermit documentation, updates, lists of current available versions, and ordering information, write to:

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

7.2. Installation

Before you can use Macintosh Kermit or any other communication program on your Mac, you must have a way to connect it to the other computers you wish to communicate with. This means either a direct cable connection (usually using a "null modem" cable), or a modem connected to your Mac and to a telephone. The Macintosh poses two special problems at this level. First, its connectors are not the standard 25-pin RS-232 style, but either 9-pin or 8-pin special connectors which you need special adapters for. Second, the Macintosh does not supply a Data Terminal Ready (DTR) signal, which is required by most modems before they will operate. To use your Mac with a modem that is not designed specifically for the Mac, you have to either (a) configure the modem to ignore the DTR signal, or (b) feed some other active signal into the modem's DTR input. The former is usually done with DIP switches on the modem, the latter can be done in the connector that plugs into the modem by installing a jumper wire between DTR (pin 20) and DSR (pin 6), or by connecting the Mac's +12V output (pin 6 on the Mac's 9-pin connector) to DTR (pin 20) on the modem end.

If you have received a Macintosh Kermit diskette from Columbia University, there's no special software installation procedure -- just insert the diskette, double-click on the appropriate start-up file, or on MacKermit itself, and go. If all the communication and other settings agree with your requirements, there's nothing else you need to do. This process is illustrated in the next section, just below.

MacKermit is not copy-protected, and nothing out of the ordinary is required to make copies onto other diskettes, or onto your hard disk if you have one. Just use the ordinary Macintosh methods of copying files, folders, etc.

Later, you may wish to create settings files tailored to your communication environment, and you might also want to customize the keyboard configuration. Use the various Settings options for this, and then select Save Settings from the File menu. Settings and settings files are explained in Sections 7.10 and 7.11.

7.3. Getting Started

Kermit programs perform two major functions, terminal emulation and file transfer. Before transferring files between two systems you must establish a terminal connection from your system to the other one, either direct or else dialed up using a modem. Then to transfer files, login to the remote system if necessary, start up a Kermit program there, and then tell the two Kermit programs which files to transfer, and in what direction.

Most Kermit programs present you with a prompt, in response to which you type a command, repeating the process until you exit from the program. If you want to establish a terminal connection to another system, you must give the CONNECT command. Unlike these programs, MacKermit is *always* connected, and whatever keystrokes you type are sent to the other system. To give commands to MacKermit itself, you must use the mouse to pull down menus from the menu bar that overlays your terminal session, or type special Command-key equivalents.

The following example shows how to transfer a file with MacKermit. The remote computer is a Unix system, but the method is the same with most others.

- First insert the MacKermit diskette. It should appear on the screen as a diskette icon titled **Kermit**

0.9(40). Click on it twice to open if it did not open automatically when you inserted it in the drive.

- Once the disk is opened, you will see three MacKermit icons across the top of the screen. For the Unix system and most others you can use the "**Normal Settings**" icon -- to start the Kermit program click twice on it. For linemode connections to IBM mainframes, you would click twice on the "**IBM Mainframe Linemode Settings**" icon.
- You will see a white backround with menus stored under the headings **File**, **Edit**, **Settings**, **Remote**, and **Log**.
- Move the mouse pointer to the **Settings** menu and select **Communications...** by clicking on it once.
- MacKermit normally sets the communication speed to 9600 bits per second. Click on the circle in front of 1200 (or whatever speed you need to match the baud rate of your modem and/or remote system). Check to see that the other communication settings like parity are as required, and make any necessary changes.
- Click on the "`OK`" box to accept the settings.
- If you have a Hayes-like dialout modem, follow the next two steps:

  1. Type AT (uppercase) and then press the Enter key. The modem should respond with "OK" or the digit "0" (zero). If it doesn't, check the cable, the modem, etc (consult your modem manual for details).

  2. Now type ATDT 7654321 followed by Enter (replace 7654321 by the actual phone number). If the connection succeeds, you'll get a message like CONNECT (or the digit "1"), otherwise you'll see an error message like NO CARRIER, ERROR, etc, or a digit like 3 or 4 (see your modem manual).

  For non-Hayes-compatible modems, follow the instructions in your modem manual. For direct connections, skip this step altogether.

Now you should be talking to the Unix system. Type a carriage return to get its attention, and it will issue its login prompt. In the examples below, underlining is used to show the parts that you would type.

`Login: christin`	*Login to the host.*
`password:______`	*(Passwords normally don't echo.)*
`% kermit`	*Run Kermit on the host.*
`C-Kermit>receive`	*Tell it to receive a file.*

Now tell MacKermit what file to send:

- Use the mouse to point to the **File** menu and select the **Send File...** option. You can either type in the name of the file (if you know the name) or select the alternate drive to see what files are on the disk. Once you see the file you want to send, click on the filename and then click on the SEND option (or you can just click twice on the filename).
- A "File Transfer Status" box will appear to report the progress of the transer. *NOTE:* If the number of retries is increasing but not the number of packets, you should check your **Communications...** settings under the **Settings** menu.
- When the file transfer is completed, the "File Transfer Status" box should disappear and the C-Kermit prompt should reappear.

You have just transferred a file from the Macintosh to the Unix system. To transfer a file in the other direction, use the "`send` *filename*" command on Unix instead of "`receive`", and click on "**Receive File...**" from the Mac's **File** menu, instead of "**Send File...**".

After the file is transferred, your terminal connection is automatically resumed. Once your Unix session is complete, you can log out, and then exit from MacKermit:

```
C-Kermit>exit
% ^D                        Logout from Unix by typing Ctrl-D.
```

1. Select the **Quit** option in the **File** menu by clicking on it.
2. Select the **Close** option in the **File** menu by clicking on it (assuming you want to close the current folder).
3. Select the **Eject** option in the **File** menu by clicking on it (assuming you ran Kermit from a diskette that you want to eject).

That's the easiest and quickest way to use Kermit. If this simple scenario does not work for you, look for any obvious incorrect settings (speed, parity), fix them, and try again. Otherwise, read on.

7.4. The Macintosh File System

The Macintosh file system consists of one or more disks, each disk containing files. There are actually two Macintosh file systems, which work slightly differently.

Disks formatted for the older Macintosh File System (MFS) are essentially "flat". All files on one of these disks must have a unique name. Files may be collected together into "folders", but folders are not analogous to directories on other file systems, and no two folders on the same disk may contain files of the same name; the folders exist only to make things look nicer in the Finder. All Macintoshes have support for MFS.

Disks formatted with the newer Hierarchical File System (HFS) are not "flat"; each folder is a directory. There may not be more than one file with the same name in a single folder, but there may be identically named files in different folders.

Macintosh file names may contain practically any printable characters, including space and punctuation -- but colon (‘‘:’’) may not be used; it is used in device names and as the HFS path element separator.

7.5. Menus

The major menus are **Apple**, **File**, **Edit**, **Settings**, **Remote**, and **Log**. The **Apple** menu gives some information about the program, including the MacKermit version number and the C-Kermit protocol mudule version number (useful in reporting bugs). It also shows statistics about the most recent file transfer.

The **File** menu invokes Kermit's file transfer functions, **Send**, **Get**, and **Receive**. It also allows settings to be saved and restored, and like most Macintosh applications, includes a "quit" entry for leaving the program, and a "transfer" entry for transferring to another program without going through the Finder.

The **Edit** menu provides support for Macintosh desk accessories that need to have this menu to do cut and paste. This menu does not currently do anything in MacKermit.

The **Settings** menu provides dialog boxes for file, communications, and protocol settings; these will be discussed below.

The **Remote** menu has the commands that can be sent to Kermit servers, as well as an option to turn Macintosh Kermit itself into a server (also discussed below).

The **Log** menu contains commands to start and stop session and transaction logging. It also has an entry to dump the current screen image to the session log, which is only enabled when the session log is open.

7.6. Terminal Emulation

Before you can transfer files, you have to establish a terminal connection with the other computer. You don't have to give MacKermit any special command to do this, just start the program. Assuming you have a physical connection, then the software will use it. If you think you have a physical connection, but don't see any results, click on the **Settings** menu and select **Communications** to make sure you have the right speed and parity. If you have to dial out to make the connection, you must do this yourself -- Mac Kermit won't do it for you. Depending on the type of modem, you must either type dialing commands to it directly (like the Hayes ATDT command in the example in section 7.3), or else dial the phone manually, wait for an answer and a carrier tone, and then put the modem in data mode.

Once you've made the connection, you can use MacKermit's terminal emulator, which conforms to ANSI standard X3.64, providing a subset of the features of the DEC VT102 terminal (a VT100 with line and character insert and delete functions added). The functions provided are sufficient to allow MacKermit to act as a terminal for the EMACS full-screen editor as it exists on most timesharing systems, and for most host-resident display-oriented applications that expect to do cursor positioning and editing on the VT100 or VT102 screen, such as VAX TPU. MacKermit does not currently support the following VT100/102 functions:

- Double height or double width lines
- Blinking
- 132 columns
- DEC-style line wrapping
- Control characters embedded in escape sequences
- VT52 mode

(this is not an exhaustive list)

The keyboard is set up by default as follows: If your Macintosh has a Control key (ie. an SE or II), Kermit uses it, and the Command (Fan, Cloverleaf) key can be used for keyboard equivalents for menus. If your Mac does not have a Control key, then the Command key is used as the Control key. The CAPS LOCK key forces all alphabetic characters to upper case. The terminal emulator sends ESC (escape) when the ‘‘´’’ (accent grave) key is pressed unshifted (even if your keyboard has an ESC key). The character ‘‘´’’ can be sent by typing Control (or Command) and the same key. The Backspace key sends a Delete (Rubout) and Control-Backspace sends a Backspace. On the original Mac keyboards, the main keypad Enter key sends a "short" (250ms) BREAK signal. The Mac+, Mac SE, and Mac II do not have a main keypad Enter key, so the BREAK function must be reassigned to another key.

You can modify the keyboard layout any way you like, defining keyboard macros, defining or moving the Control and Escape keys, etc., using MacKermit's built-in key configuration features. Older MacKermits (version 0.8 and earlier) came with a separate key configuration program called CKMKEY. This should not be used, because it does not understand the format of the 0.9 and later keyboard configuration software.

MacKermit includes a mouse-controlled cursor positioning feature for use during terminal emulation. If the "Mouse -> Arrow Keys" feature is turned on (via the **Terminal** entry of the **Settings** menu), then when the mouse button is pressed, the program acts as if you typed the VT100 keypad arrow keys to move the terminal cursor to where the mouse cursor is. MacKermit does this by sending the absolute strings for arrow keys, independant of what is bound to the actual arrow keys of the keyboard.

MacKermit sets the Mac hardware to do 8-bit data communication with no parity, and then correctly sets the parity bit of each character itself in software, as requested in the **Communication** settings menu. This has the benefit of avoiding the problem of a machine which requires a different input parity than it sends back. MacKermit will correctly receive all of the characters sent to it, no matter which parity they are.

To allow useful coexistence of desk accessories and Kermit, the terminal emulation window may be dragged using the drag bar. If a desk accessory overlays the emulation window, the emulation window can be clicked upon to move it in front of the DA, and later dragged to reveal the hidden desk accessory so that it can be restored to the foreground. The same thing can be done with Kermit's own remote response window as well. Note that Kermit's terminal emulation window does not accept input when any other window is in the foreground.

MacKermit uses XON/XOFF (control-Q and control-S) flow control during terminal emulation and file transfer. If the other system does not understand XON/XOFF, problems may result at high speeds. The terminal emulator can normally keep up at 9600 baud, and has a very large input buffer, but after several continuous scrolling screens at this speed, some characters may be lost. When running at high baud rates on a system that does not understand XON/XOFF flow control, either keep your terminal in page mode, use a text paging program such as Unix "more", or view text with a non-scrolling screen editor. Also, don't drag the terminal emulation window while characters are arriving; if you do, the characters may be lost and the display may become confused.

During terminal emulation, the characters displayed on the screen may also be saved on the disk. This allows you to record interactions with the remote system, or to "capture" files that you can't transfer with Kermit protocol, for example when the remote system does not have a Kermit program. Use the **Log** menu, and choose session logging to activate this feature. The result goes into a file called "Kermit Session" in the current folder, which is always appended to, rather than overwritten. To create a new session log, delete or rename the old one first.

The following features are missing from the MacKermit terminal emulator, and may be added in subsequent releases:

- Restoration of character attributes such as underlining or highlighting.
- Cutting text from screen to clipboard.
- Transmission of raw text to host (e.g. pasting to screen).
- Screen rollback.
- Screen resizing.
- Explicit modem or dialer control.
- Login scripts.
- Printer support.
- Ability to use the printer port for terminal emulation.
- A way to disable XON/XOFF flow control, or select other flow controls.

7.7. File Transfer

Like most Kermit programs, MacKermit allows you to send and receive text or binary files singly or in groups. It will interact with a remote Kermit server, and it can act as a server itself. However, due to the unique nature of the Macintosh file system, there are some special considerations:

- <u>Mode</u> - Text or Binary. Binary means the data is sent or stored without modification. Text means that every carriage return character (CR) in a Macintosh file is translated to a carriage-return-linefeed (CRLF) sequence when sending, and every CRLF in an incoming file is turned into a CR when stored on the Mac disk. A text file is produced when you save a file from MacWrite or other applications using the "text only" option; text files are not associated with any particular Macintosh application and can be sent in a useful fashion to other kinds of computers.

  A word of caution about Macintosh text files: The Macintosh supports an extended version of ASCII, with characters like accented and umlauted vowels in the 128-255 range. These characters allow representation of Roman-based languages other than English, but they do not follow any of the ISO standards for extended character sets, and thus are only useful on a Mac. When transferring text files, you should ensure that either there are no extended characters in the file, or that the other system can understand the Mac's 8-bit characters.

- <u>Fork</u> - Data or Resource. Macintosh files may have two "forks". The data fork contains data for an application; the resource fork contains icons, strings, dialog boxes, and so forth. For instance, a MacWrite document contains text and formatting information in the data fork, and fonts in the resource fork. For applications, the executable code is stored in the resource fork.

File transfer is initiated when you select **Send file...**, **Receive File...**, or **Get file from server...** from MacKermit's **File** menu.

File transfers can be canceled by clicking on the Cancel File or Cancel Group buttons. These will always work when sending. When receiving, they will work if the opposite Kermit honors this (optional) feature of the protocol. There is also an "emergency exit" from any protocol operation, which can be taken at any time by typing "Command-." -- that is, hold down the Command (Fan, Cloverleaf) key and type period.

The progress of file transfer operations can be logged into a Macintosh file called a "transaction log". This log will show the names of the files transferred, the date and time, and the completion status. This feature is useful with long unattended transfers -- you can come back later and read the transaction log to find out what happened. The transaction log is called "Kermit Log".

The current version of Mac Kermit can only send one fork of a file at a time. When a file has two forks, there is no provision for sending both forks together. This restriction may be lifted in future releases of MacKermit, for example by converting applications to MacBinary format during transmission.

7.7.1. Sending Files

To send files, first put the remote Kermit in server mode, or else give it the RECEIVE command. Then use the mouse to select **Send file...** from the **File** menu. This will give you a MacKermit file-open box, which includes the standard Macintosh dialog items -- a file list, Disk and Eject buttons, etc. You can either send one file at a time, by clicking on its name in the file list, or send the entire contents of the current HFS folder (for HFS disks only, of course). Clicking the Disk button will switch the file list to another physical disk. If desired, you can type an alternate name to send the file under. When you select a file, MacKermit examines its type; if the type is APPL, then MacKermit expects to send the resource fork in binary mode, otherwise the data fork in text mode. The Mode and Fork radio buttons will display these choices; you may change them before clicking the Send button.

7.7.2. Receiving Files

You can receive or get multiple files, providing the opposite Kermit is capable of sending multiple files in a single transaction (most are). To receive files, first give the remote Kermit a SEND command and then select **Receive file...** from the **File** menu. To get files from a server, first put the remote Kermit into server mode, then select the **Get file from server...** option from the **File menu**, and type in the name of the file you want to get, or a wildcard designator for multiple files, in the remote system's filename syntax.

As each file arrives at the Mac, it will be decoded according to the current mode (text or binary), and stored in the default fork (data or resource). The file names will be either the names the files arrive with (overwriting existing files of the same names) or new unique names (when name conflicts occur), according to the current default for name collisions. You may also elect to perform an "attended" receive, in which you have an opportunity to override file defaults on a per-file basis (do this in the **Protocol** section of the **Settings** menu). But attended operation must be used with caution -- if you take too long (more than about a minute) to execute an incoming file's dialog box, the opposite Kermit could time out and terminate the transaction. If this happens, tell the opposite Kermit to send again and try again with the receive dialog.

The folder for new files is the same as the location of the settings file, or if no settings file was used then the new files appear on the desktop. If you are transferring a lot of files and want to keep them together, create a folder, drag the settings file into it, and double click on the settings file; all created files will appear in that folder.

7.8. Remote Commands

When connected to a Kermit server, MacKermit is capable of issuing special file management and other commands to it. The **Remote** menu contains these commands. You may request directory listings, you can delete files, change directories, etc, on server's machine. The response from these commands (if any) is displayed in a special pop-up window. Responses to multiple Remote commands are separated by a dashed line. The response window can be scrolled, sized, and positioned, and can be hidden by clicking the menu item "**Hide Response**" or the window's go-away box; all text remains intact and will be appended to the next time you do a Remote command; it can also be brought to the foreground by clicking the **Show Response** menu item. Note that typein to the terminal emulator will not take effect when the response window -- or any other window (such as a desk accessory) -- is up front. This is not a bug, but a feature of the Macintosh user interface guidelines.

If the response buffer gets too full (greater than 30,000 characters), MacKermit will remove enough text from the beginning of the buffer, in 512 byte chunks, to make it less than 30,000 characters again.

A Remote command can be canceled by taking the Emergency Exit (Command-`.`). To disengage from the remote Kermit server, click on **Bye** or **Finish** in the **Remote** menu.

7.9. Server Operation

MacKermit may itself act as a Kermit server. Just set the desired parameters in the **Settings** menu, then click on **Be a Server** in the **Remote** menu. The MacKermit server can respond to SEND, GET, REMOTE DIRECTORY, FINISH, and BYE commands. You can send single or multiple files to a MacKermit server, and you can get a single file from it by name. You can also get all the files in the current folder by using a colon (‘‘`:`’’) as the file specification in the GET command:

```
GET :
```

If you give the FINISH command, MacKermit will return to terminal mode. If you give the BYE command, the Macintosh will reboot itself.

You can take MacKermit out of server mode from the Mac keyboard by typing the emergency exit sequence, Command-dot.

7.10. Settings

You can change File, Communications, Protocol, Terminal, Keyboard macros, and Keyboard modifier settings by using the **Settings** pull-down menu. You can save and load these settings by invoking the appropriate selection in the **File** menu. If the "bundle bit" has been correctly set on your version of MacKermit (it should be), then you can double-click on the resulting document to start MacKermit with those settings.

The **File** settings establish the defaults for file transfer:

- Attended versus Unattended operation for incoming files.

- Naming: When doing unattended file reception, whether incoming files should supersede existing files of the same name, or a new unique name should be assigned to them. If the latter, the new name is formed by adding a dot and a number to the end. For instance, if a file called FOO exists and a file called FOO arrives, MacKermit will store the arriving file as FOO.1; if FOO.1 exists, then FOO.2, etc.

- Mode: text or binary. Used for received files only. When sending, MacKermit tries to figure out an appropriate mode for the file being sent (but then lets you override it the Send File dialog).

- Fork: which fork -- data or resource -- to send, or to store an incoming file into.

The **Communications** settings allow you to set the baud rate (anywhere between 300 baud and 57.6K baud, except 38.4K baud), and parity (odd, even, mark, space, or none). When the parity is set to none the Macintosh uses an 8-bit-wide connection. All other parity settings tell the Macintosh to use a 7-bit-wide connection, and to request 8th-bit prefixing when transferring 8-bit data. If the remote host or the communication path uses any kind of parity, then you won't be able to transfer files successfully unless you tell MacKermit (and in most cases also the Kermit on the other end) about it. Duplex is selected in the **Terminal** settings.

The **Protocol** settings allow you to set packet parameters for both incoming and outbound packets. These include the block check type (1 or 2 character checksum, 3-character 16-bit CRC-CCITT), line turnaround handshake character (for file transfer with half duplex systems), packet start and end characters, padding, packet length, timeout interval, and packet length. Characters are specified by entering their ASCII value in decimal, e.g. 1 for Control-A, 13 for Control-M (Carriage Return), etc. The RECEIVE parameters are conveyed by MacKermit to the other Kermit. For instance, if you set the receive-packet-length to 500, MacKermit will tell the other Kermit to send 500-character packets. The SEND parameters are used to override negotiated values, and need rarely be used.

Long packets are selected by setting the RECEIVING packet length between 95 and 1000. Normally, you should not change the sending length because MacKermit, and most other Kermits, will configure themselves correctly. Note also that the fastest file transfers will happen with long packets in the range of 300-500. Very long packets actually end up being much slower, because the operating systems in both the Mac and the other machine have to do more work to cope with such long inputs, and, under noisy conditions, the probability is higher that a longer packet will be struck by noise, and will take longer to retransmit.

The **Terminal** settings let you modify the characteristics of the VT102 emulator, such as auto-linefeed, autowrap, autorepeat keys, block vs underline cursor, blinking vs steady cursor, inverted screen (reverse video), and smooth scrolling. There is also a "visible bell" for those who can't hear the audible bell produced upon receipt of a Control-G, and an option to display control characters visibly by showing their numeric ASCII values (in decimal) in a single character cell. If local echo is needed, as in half-duplex connections, that must be specified here also.

7.11. Settings Files

You can start MacKermit with all its "factory settings" by double clicking on the MacKermit icon. Factory settings are designed for direct communication with most other microcomputers, DEC minis and mainframes, etc: 9600 bps, no parity, XON/XOFF, remote echo, etc. You can change the communication, protocol, file, keyboard, and terminal settings by going through the options in the **Settings** menu. Once you have set all parameters as desired, you can save your settings in a "MacKermit settings file" by selected "**Save Settings...**" from the **File** menu. A settings file is, in Macintosh terminology, a "MacKermit document". You'll recognize it because it looks like a dog-eared piece of paper with the MacKermit icon superimposed. You can have more than one settings file.

There are two ways to use a settings file. First, you can double-click on it, just as you can double-click on a MacWrite document to start up MacWrite to edit a particular file. This method starts up MacKermit with all the saved settings. The other method is to click on the "**Load Settings...**" option in the **File** menu from inside MacKermit. This lets you change settings without leaving and restarting the program. **Load Settings...** shows all MacKermit settings files in the selected folder. Opening one of them loads all its settings, removing all current settings.

You can "edit" a MacKermit settings file by loading it, going through the `Settings` menu, and then saving the settings either in a new file, or overwriting the same file.

As distributed by Columbia, Mac Kermit comes with two settings files. One is called "Normal Settings", and is pretty much identical to Mac Kermit's factory settings. The other is "IBM Mainframe Linemode Settings". It selects mark parity, local echo, XON half-duplex line turnaround handshake. You can use these files as-is, customize them for your own environment, or create new settings files for all the different kinds of systems that you use.

7.12. Reconfiguring the Keyboard

Beginning with version 0.9, MacKermit has keyboard configuration functions built in. These are accessed through the **Set Key Macros** and the **Set Modifiers** entries in the **Settings** menu.

The Macintosh keyboard is composed of normal keys and modifier keys. Modifier keys are those keys that, when held down, change the meaning of other keys. On the Mac these are: SHIFT, CAPS LOCK, OPTION, CONTROL (only on the Mac II and SE), and COMMAND (also known as APPLE, CLOVER, or FAN). Normal keys are the letters, numbers, special symbols, arrow keys, space bar, and function keys. Only one normal key can be typed at a time, but one or more modifier keys can be pressed down along with it.

When you type a key, Kermit reads both the ASCII value, and the keyboard-independent scan code for that key. Kermit looks in its table of key macros to see if there is a macro for this combination of key and modifiers, and if so sends the macro. If there is no macro, Kermit then looks in its modifier table to see if any of the modifiers do special things to the character; if so, it does these to the character. Finally, Kermit sends the character. In the normal case when there is no macro and no modifiers apply, the character sent is simply the ASCII value for that character.

It is important to keep in mind that if the parity setting is something other than <u>none</u>, the high (8th) bit will be stripped off of the characters when they are transmitted. Since most systems do not understand characters in the range 128 -- 255 (decimal), you should avoid using the Apple extended characters (accented vowels, for example) during terminal connection.

7.12.1. Defining Key Macros

To define a new key macro, select the **Key Macros** entry. A dialog window will appear, asking you to press the key to define. Type the key (including any of the modifiers). A new dialog will appear, with an editable text field in it. Enter the definition for the key here. Your definition may be up to 255 characters long, and can include all of the control characters (including NUL). Special characters can be included in the macro by entering a ‘‘`\`’’ (backslash), followed by up to 3 *octal* (base 8) digits for the value (just like in the C programming language). For example, an ASCII NUL (value 0) would be written as ‘‘`\000`’’, carriage return (ASCII 13) would be written ‘‘`\015`’’ (1 x 8 + 5 = 13). Also, control characters may be entered with a backslash, followed by a caret (or circumflex, ‘‘`^`’’), followed by the corresponding letter. Thus a Control-G (value 7) could be entered as ‘‘`\007`’’, ‘‘`\^G`’’, or ‘‘`\^g`’’. To include a literal backslash in a definition, type in two of them: ‘‘`\\`’’.

BREAK conditions are also programmable as macros. If the entire macro the string is ‘‘`\break`’’, then typing the defined key will send a short (1/4 second) break. A long (3.5 second) BREAK is defined with ‘‘`\longbreak`’’. Note that a macro can define either a BREAK, or a string of normal characters, but not both.

7.12.2. Defining Key Modifiers

Skip ahead to the next section if you already know about things like SHIFT, CAPS LOCK, CONTROL, and META.

On a typewriter the only modifier key is SHIFT. Typing a character with no modifier key depressed selects a lowercase letter or the character printed on the lower face of the keytop (say, the digit "4"). Typing a character with SHIFT depressed selects an uppercase letter or the character printed on the upper face of the keytop (say, a dollar sign). Some keyboards also have a SHIFT LOCK key, which stays down once pressed and pops up the next time it’s pressed; its operation is equivalent to holding down SHIFT. And some keyboards have a CAPS LOCK key which operates like SHIFT LOCK, but only upon letters.

Computer terminals also have a modifier key called CONTROL (or CTRL). Its function is a little less obvious: it is intended to produce one of the 33 characters in the "control range" of the ASCII alphabet. Control characters are not graphic -- they are intended for use as format effectors (like carriage return, formfeed, tab, backspace), for transmission control, or for device control. The remaining 95 characters -- letters, digits, punctuation, and space --

are the graphic characters. When a character is typed with the CONTROL modifier pressed, its "control equivalent" (if any) is transmitted. By convention, the control equivalent of A is Control-A, B is Control-B, etc, and there are also seven special control characters generally associated with punctuation characters or special keys. For the "alphabetic" control characters Control-A through Control-Z, SHIFT or CAPS LOCK modifiers are ignored; for the others, operation varies from terminal to terminal.

The SHIFT and CONTROL modifiers allow all 128 ASCII characters to be sent from a normal typewriter-like keyboard that has about 50 keys. However, certain host-resident computer applications -- notably the full screen text editor EMACS and its descendents -- can be used to greater advantage with a 256 character 8-bit alphabet (EMACS responds to single-character commands, and the more characters a terminal can send, the more commands are directly available).

For this purpose, some terminals also provide a META modifier key. This key simply causes the high-order ("8th") bit of the selected 7-bit ASCII value to be set to 1 upon transmission. This can only work when the connection is 8-data-bits-no-parity. When parity is in use, EMACS allows a sequence of two 7-bit ASCII characters to represent a single meta character. The advantage of having a real META modifier key is that it can be held down while the actual key is struck repeatedly or even autorepeats, whereas a use of a "meta prefix" such as <escape> requires much more typing. To illustrate, suppose META-F is the command to go forward one word. If you want to execute this operation repeatedly, just hold down META and F and let it autorepeat. If you don't have a META key, then you'd have to type <escape>F<escape>F<escape>F..., etc.

A common problem faced by computer users who switch from one terminal or PC to another is the placement of the modifiers and other special keys. DEC, IBM, Apple, and other manufacturers consistently move these keys around on new models of their keyboards. MacKermit allows you to assign any of various functions to any of the Mac's modifier keys, and to assign any desired character or character sequence to the regular keys, so that you can tailor the layout of your Mac's keyboard to suit your taste.

7.12.3. Modifiers Dialog

To change the action of any of the modifier keys, select **Modifiers** from the **Settings** menu. A dialog will appear that looks roughly like the one in Figure 7-1 (the ''%'' represents the Apple or Clover key).

```
         Modifier Pattern:  -->   Modification:

      Ctrl Opt Lock Shift  %      Unmodify Caps Ctrl Meta Prefix string:
                                                          _____________
      [X]  [ ] [ ]  [ ]   [ ]   |   [ ]     [ ]  [X]  [ ]  [_____________]
      [ ]  [ ] [ ]  [ ]   [X]   |   [ ]     [ ]  [X]  [ ]  [_____________]
      [ ]  [X] [ ]  [ ]   [ ]   |   [x]     [ ]  [ ]  [ ]  [\033_________]
      [ ]  [ ] [ ]  [ ]   [ ]   |   [ ]     [ ]  [ ]  [ ]  [_____________]

                    (Cancel)   (Help)   ( OK )
```

Figure 7-1: MacKermit Key Modifier Dialog

The check boxes are divided into rows, each one describing a modification. The left half of each row describes the modifier combination to look for; a checked box means that this key is down, and an unchecked box means "don't care". Note that there is no way to specify a key being up, and lines with nothing checked on the left side will be ignored; the character will be modified in the normal Macintosh way.

The right half describes what modification to do to the characters. The Unmodify modification says "make this the character that would be sent from the same key with no modifer keys pressed". In other words, un-Option, un-Caps, un-Control, and un-Shift this character. The Caps modification translates all letters to upper case, Ctrl makes the letter a contol character, Meta sets the high (8th) bit on the character, and if a Prefix string is present, it is sent before the character is.

Hints about modifiers:

- Beware of the Option key. It changes the value of any characters you use with it. If you type Option-F, the Mac will send a D, if you type Option-B, the Mac will send a ‘‘`:`’’, etc. If you want to use the option key as a modifier, be sure to check the "Unmodify" box.

- To use MacKermit with a version of EMACS that does not accept 8-bit Meta characters, define a key, like Option, to be unmodified, with a prefix string of `\033` (ASCII Escape), as in Figure 7-1. Then you can hold down Option and type F (or any other key) repeatedly, or let it autorepeat, and MacKermit will send the correct prefix-Meta sequence.

- When interpreting a keystoke, MacKermit checks the list of modifiers from top to bottom, applying the first one that matches. This means that if you want a different modifier for Command-Option and just plain Command, you must put the definition for Command-Option first in the list.

7.13. Bootstrapping

This section applies if you do not have a MacKermit diskette, but MacKermit is available for downloading from some other computer.

MacKermit is distributed in source form for building on a Macintosh, running Apple’s Macintosh Programmers Workbench (in MPW C), in `.HQX` "BinHex 4" form, and sometimes also as a binary resource file. Those who want to work from the source are referred to the file `CKMKER.BLD` for instructions.

If you’re downloading, it’s best to work with `CKMKER.HQX`, a textual encoding of the MacKermit application. Download this using any technique available to you -- an old release of Kermit, an Xmodem implementation, even raw screen capture. Then run BinHex (version 4) to convert it into a working application (select **Upload -> Application** from the **File** menu). Eveything will be set up correctly -- icons, forks, etc.

If you don’t have the `.HQX` file available, but you do have access to the binary resource file (its name will be `CKMKER.RSRC`, `ckmker.rsrc`, `CKMKER.RSR`, `ckmker.rsr`, `%ckmker` or some variation on these, depending on what system it’s stored on and how it got there), AND if you have "MacPut" on your system and MacTerminal on your Mac, AND if you have an 8-bit-wide (no parity) data path between your Mac and your system, then you can use MacPut to download the binary resource file to your Mac using MacTerminal’s "MacBinary" format (a variant of XMODEM). After doing this you must use a program such as SetFile or ResEdit on the Mac to set the author to KR09, the type to APPL, and turn on the bundle bit. Do not bother with the CKMKEY program, as it is not used with newer MacKermits. If you have an earlier release of MacKermit, you may use it in place of MacTerminal and MacPut.

7.14. Differences Between Versions 0.8 and 0.9

MacKermit 0.8(34) runs on the 128K Mac, the 512K Mac, and the Mac Plus, but not on the Macintosh II or SE. MacKermit 0.9(40) runs on all Macs except the 128K original. You should use version 0.9 unless you have a 128K Mac.

The second major difference is that the program is has been translated into Apple MPW C, so that it can be edited, compiled, and built on the Macintosh itself. This was done originally by Jim Noble of Planning Research Corporation, who converted MacKermit from SUMACC C (which had to be cross compiled on a UNIX system) to Megamax C. Jim’s version was converted to MPW C by Matthias Aebi, who also added most of the new features listed below. Paul Placeway integrated the program with the current (long packet) version of C-Kermit and added additional new features.

Besides these important differences, there were many other changes from version 0.8 to version 0.9, including:

- The Cursor with open desk accessories now works correctly
- Long packet support
- New program icon
- New settings files are no longer TEXT
- Settings can now be written back to an already existing settings file
- Key redefinition function built in to Kermit, no more CKMKEY
- Server mode directory listing feature
- Multifile (folder) send
- Server "Delete" file command
- Server "Space" command
- Get whole folder content from the server with filename ‘‘:’’
- Recognition of all the different Mac keyboards
- Support of menu command keys (key macros)
- Terminal settings dialog separated from communication settings
- Non-transparent terminal mode
- Display of statistics and protocol version to "About Kermit" dialog.
- Parity problems fixed
- Session logging
- Transaction logging
- Multifinder support
- Additions to the VT102 emulator (smooth scrolling, etc)
- Rearrangement of menus and displays
- Program no longer hangs if remote response window gets too full
- Program now works correctly on 64K ROM machines
- A new manual

This manual applies in large part to version 0.8(34), except that the older version is missing the new features listed above, and it comes in two pieces: CKMKER and CKMKEY. The CKMKEY program is used to program the keys, like the **Set Key Macros...** and **Set Modifiers** described in this manual, and creates a settings file which Kermit itself uses. The old version only works well with early Macintosh keyboards.

8. IBM 370 KERMIT

Program: John Chandler (Harvard/Smithsonian Center for Astrophysics); contributions from Vace Kundakci and Daphne Tzoar (Columbia U), Bob Shields (U. Maryland), Victor Lee (Queens U.), Gary Bjerke (U. Texas at Austin), Greg Small (UC Berkeley), Clark Frazier (Harvard Bus. Sch.), Bob Bolch and Steve Blankinship (Triangle), Ron Rusnak (U. Chicago), Roger Fajman and Dale White (NIH), Andre Pirard (U. Liege)
Language: IBM 370 Assembler
Documentation: John Chandler (CfA)
Version: 4.0
Date: 1988 February

Kermit-370 is a family of programs implementing the KERMIT file transfer protocol for IBM 370-series mainframes (System/370, 303x, 43xx, 308x, etc.) under a variety of operating systems. Kermit-370 operates over asynchronous ASCII communication lines attached to 3705-style front ends (‘‘`TTY`’’ devices), to a Series/1 or 4994 running the Yale ASCII Terminal Communication System or the IBM 7171 ASCII Device Control Unit or a 9370 with ASCII subsystem (‘‘`SERIES1`’’ devices), or to SNA-type front-ends with graphics pass-through mode, such as the MICOM 7400, Datastream/Leedata 8010, and PCI 1076 (‘‘`GRAPHICS`’’ devices), or, in some configurations, to an IBM 3708 (‘‘`FULLSCREEN`’’ device). As of this writing, the pending implementation of full 7171 compatibility in the program product SIM3278 has not been completed. For more details on front-ends, see the section SET CONTROLLER.

The source is coded in IBM 370 assembly language and is divided into sections, some generic and some specific to an individual operating system. While the details of file-system and supervisor interaction vary widely among the systems available for IBM 370's, the basic features are essentially the same. This chapter will describe the features common to all variants of Kermit-370, and a separate chapter will deal with the system-specific details for each variant.

IBM 370 systems have some peculiarities that users should be aware of. First, they are essentially half-duplex systems; the communication line must "turn around" before any data can be sent to it. The ‘‘`TTY`’’ devices are strictly half-duplex, and even the ‘‘`SERIES1`’’ and ‘‘`GRAPHICS`’’ devices, although they maintain full-duplex communication with the terminal, must transmit a block at a time to the mainframe. The fact that a packet has been received from the IBM system through a ‘‘`TTY`’’ device is no guarantee that it is ready for a reply; generally, the true indicator of readiness is the line turnaround character (XON), which the operating system sends immediately before issuing a read request. On some systems, however, it is possible for Kermit to do away with the system-supplied turnaround and schedule read requests immediately after the corresponding writes. It is up to the user to tell the other Kermit that it must conform to the requirements of the IBM mainframe.

Second, disk files are encoded using the EBCDIC character set. Consequently, there are three layers of character translation on packets exchanged on a ‘‘`TTY`’’ device. For an incoming packet, the outer layer is provided by the operating system, which translates all characters from ASCII to EBCDIC. Kermit-370 must then translate the packets back to ASCII (the middle layer) in order to calculate and verify the checksum. Data arriving through a ‘‘`SERIES1`’’ or ‘‘`GRAPHICS`’’ device are still in ASCII and therefore bypass the two outer layers. In any case, Kermit-370 translates everything finally into EBCDIC (the inner layer) before storing on disk (except BINARY files). When Kermit-370 sends a file, the opposite translations occur. In translation, EBCDIC characters not representable in ASCII are replaced by nulls. In some cases, several EBCDIC characters are mapped into a single ASCII character, but no two 7-bit ASCII characters are translated into the same EBCDIC character. The middle-layer tables used by Kermit must be the inverses of the corresponding outer-layer ones used by the host operating system if file transfers are to work at all. If necessary, the system programmer should add the appropriate SET TATOE/TETOA/TTABLE subcommands (*q.v.*) to the global ‘‘`INIT`’’ file (see the next section). Indeed, it is usually a good idea to set TTABLE ON in the global ‘‘`INIT`’’ file to force using different built-in sets of tables for the inner and middle layers, whenever the system has ‘‘`TTY`’’ devices. The standard ASCII-to-EBCDIC translations can be found in the Appendix or the IBM System/370 Reference Card.

Another distinction of IBM 370's is that they store and retrieve files as records rather than byte streams. Records may be either fixed-length with some sort of padding (as needed) or varying-length with some sort of (generally hidden) delimiters. Thus, Kermit-370 must assemble incoming data packets into records by stripping off carriage return-linefeed pairs (CRLF's) and padding with blanks or truncating as needed and must strip trailing blanks and append CRLF's to outgoing records. Further, disk files typically have the records combined into blocks for efficiency. One consequence of this form of storage is that files have attributes describing the component records: maximum record length (LRECL), record format (RECFM), and sometimes block size (BLKSIZE).

As mentioned before, Kermit-370 is a family of programs. At present, only the CMS and TSO versions are operational. Versions for DOS/VSE and MTS have at least reached the "drawing board," but no others have even been started. Volunteers are always welcome to port Kermit-370 to other operating systems or add new features to the existing family. Anyone interested should first get in touch with the Center for Computing Activities at Columbia University to find out what projects of a similar nature are already pending (and thereby prevent unnecessary duplication of effort).

8.1. Program Operation

Kermit-370 can be invoked directly or from a command procedure. In either case, it reads and executes subcommands sequentially until directed to quit and then returns. A subcommand consists of one or more fields (words) separated by spaces.

Upon initial startup, the program looks for two (optional) initialization files, one system-wide and a second specific to the user. Both *filespecs* are, of course, system-dependent. The purpose of these files is to allow Kermit to be customized for a particular system and for a user's specific settings without changing the source code. The system-wide file, which is maintained by a systems programmer, should contain Kermit subcommands that all users would need to issue in order for Kermit to run on the system, such as subcommands to modify the ASCII/EBCDIC tables used by Kermit-370. The user-specific file, if any, should contain subcommands that the user generally issues every time Kermit is run. Kermit-370 executes any subcommands found in these files as though they were typed at the terminal. Here is a sample ''`INIT`'' file:

```
* Asterisk in column one is a comment.
set debug on
set warning on
set block 3
```

During interactive execution, you may use the built-in help feature while typing Kermit-370 subcommands. A question mark ("?") typed at almost any point in a subcommand, followed by a carriage return, produces a brief description of what is expected or possible at that point. Moreover, mistyping a subcommand will generally produce a helpful error message or a list of possible options at the point of error. Keywords in such lists are displayed with the minimum-length abbreviation in upper case and the remainder, if any, in lower case. In entering Kermit subcommands, any keyword may be shortened to any substring that contains the minimum abbreviation.

8.2. Kermit-370 Subcommands

The following is a brief summary of Kermit subcommands. The starred subcommands can be issued as remote Kermit commands to Kermit-370 when it is in server mode. System-specific subcommands are omitted from this list.

BYE	logs out other Kermit server.
CWD*	establishes a new working directory.
DIRECTORY*	displays all or part of the disk directory.
ECHO	a line back to the user.
EXIT	from Kermit-370.
FINISH	other Kermit server.
GET	file(s) from a Kermit server.

HELP	about Kermit-370.
HOST*	executes a system command.
KERMIT*	executes a Kermit subcommand.
QUIT	from Kermit-370.
RECEIVE	file(s) from other Kermit.
SEND	file(s) to other Kermit.
SERVER	mode of remote operation.
SET*	various parameters.
SHOW*	various parameters.
SPACE*	displays disk storage allocation.
STATUS*	inquiry.
TAKE*	subcommands from file.
TDUMP*	dumps the contents of a table.
TYPE*	a file.
XECHO	echoes a line (transparently).
XTYPE	displays a file (transparently).

Although Kermit-370 is generally a remote Kermit, it has the capability of communicating with another Kermit in server mode. In that situation, the subcommand prefixes REMOTE and LOCAL refer to the Kermit server and Kermit-370, respectively, even when Kermit-370 is, strictly speaking, the remote Kermit. Any replies from the Kermit server are added to a disk file (whose *filespec* is, of course, system-dependent). Such a transaction can be carried out, for example, under control of a TAKE file if Kermit-370 is not operating locally. If the local Kermit has a "magic" character sequence that switches it from terminal emulation to server mode, then an entire session could be controlled from the mainframe, possibly in response to a single command issued by a naive user. For example,

```
grab
```

Kermit-370 is invoked and executes the following TAKE file

`ECHO Serve Me!`	*the local Kermit switches to server mode*
`GET file.a`	*the server uploads file.a*
`FINISH`	*the server switches back to terminal mode*

The remainder of this section concentrates on the subcommands that have special form or meaning for Kermit-370, but neglects those with highly system-dependent syntax or use.

The SEND Subcommand

Syntax: `SEND [`*filespec* `[`*foreign-filespec*`]]`

The SEND subcommand tells Kermit-370 to send a file or file group to the other Kermit. If no such file exists, Kermit-370 simply displays an error message and issues another prompt. If one or more files are sent, their names are recorded in memory and may be viewed later via the TDUMP subcommand (but with no indication of whether any of them were rejected or cancelled by the other Kermit). If this subcommand is issued without any arguments, Kermit-370 will prompt the user for both native and foreign *filespecs* (and will insist on getting the former, but will do without the latter).

When Kermit-370 sends files using long packets (longer than 96), the throughput is especially sensitive to the level of noise on the line because retries are so time-consuming. Therefore, Kermit-370 adds an extra, heuristic size limit for packets when retries have been found necessary. When that is the case, after every 20 packets, Kermit computes the packet size for maximum throughput assuming that the transmission errors were due to sparse, Poisson-distributed noise bursts. The result of this calculation is then used as another limit on the size of outgoing packets besides the one specified by the other Kermit. If no retries are required, then Kermit-370 assumes the line to be noiseless and sends packets of the maximum length the other Kermit allows. For more details on the SEND subcommand syntax and operation, see the chapter on the desired system-specific version of Kermit-370.

The RECEIVE Subcommand

Syntax: `RECEIVE [`*filespec*`]`

The RECEIVE subcommand tells Kermit-370 to accept a file or file group. The user must issue the corresponding SEND subcommand to the other Kermit. Under some circumstances, the records of the received file(s) may be truncated; when this happens, Kermit does not stop, but notes the fact as an error (unless something more serious happens later). For more details on the RECEIVE subcommand syntax and operation, see the chapter on the desired system-specific version of Kermit-370.

The GET Subcommand

Syntax: `GET [`*foreign-filespec* `[`*filespec*`]]`

The GET subcommand tells Kermit to request a file or file group from the other system, which must have a Kermit running in server mode. Provided the other Kermit complies, the effect is the same as if SEND *foreign-filespec* had been issued directly to the other Kermit and RECEIVE [*filespec*] to Kermit-370. other Kermit). If this subcommand is issued without any arguments, Kermit-370 will prompt the user for both foreign and native *filespecs* (and will insist on getting the former, but will do without the latter). See the respective SEND and RECEIVE subcommands for a description of the each *filespec*.

The TAKE Subcommand

Syntax: `TAKE` *filespec*

Execute Kermit subcommands from the specified file, usually called a TAKE file. The TAKE file may in turn include TAKE subcommands to a nesting depth of ten. If a TAKE file includes the subcommand SERVER, however, the nesting count starts over again in server mode. The user has the option of seeing the subcommands echoed from the TAKE file as they are executed and also the option of automatically exiting from a TAKE file on error. See the subcommand SET TAKE for details.

The SERVER Subcommand

Kermit-370 is capable of acting as a server. In server mode, Kermit-370 can send and receive files, execute host commands, execute a restricted set of Kermit subcommands, and perform a variety of generic Kermit functions. The following list shows the typical local Kermit commands along with the server functions they elicit. When Kermit-370 is talking to another Kermit running in server mode, these same subcommands may be used in the other direction.

BYE	log out the Kermit server.
FINISH	server mode.
GET	a file or files from the server.
REMOTE	
COPY	a file or files.
CWD	set new working directory.
DIRECTORY	display file attributes.
ERASE	a file or files.
HELP	display this command summary.
HOST	execute a system command.
KERMIT	execute a Kermit-370 subcommand.
RENAME	a file or files.
SPACE	display disk space.
TYPE	a file.

SEND	a file or files to the server.

If your local Kermit does not support the REMOTE KERMIT command, you may need to issue SET subcommands to select various options before typing the SERVER subcommand. Once in server mode, Kermit-370 will await all further instructions from the user Kermit on the other end of the connection until a FINISH or BYE command is given.

Command execution in server mode is different in some respects from normal operation. First of all, some Kermit subcommands are not allowed (see the list at the beginning of this section). Moreover, command errors always terminate any active TAKE file. Also, all commands will be run in the special environment that Kermit sets up during protocol transfers. Among other things, Kermit intercepts all terminal I/O (if possible) in this environment in order to transmit the data to the local Kermit as text packets.

Note that some operations can be requested by several different commands. If for example, the IBM 370 system has a command ‘‘PRT’’ for displaying a file, a user interacting with a Kermit-370 server can choose to display a file by issuing any of the commands: REMOTE TYPE, REMOTE HOST PRT, REMOTE KERMIT TYPE, REMOTE KERMIT HOST PRT, or (if SYSCMD has been set ON) REMOTE KERMIT PRT. The first form simply transfers the requested file as text, but the others invoke the ‘‘PRT’’ command with any specified options, intercept the terminal output, and return the results to the local Kermit.

The SET Subcommand

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

The SET subcommand establishes or modifies various parameters controlling file transfers. The values can, in turn, be examined with the SHOW subcommand. Some parameters have two levels. In particular, there are two matching lists of SEND and RECEIVE sub-parameters corresponding to the values exchanged by Kermits in the Send-Init/ACK sequence. For each of these SEND/RECEIVE pairs one element is encoded in outgoing parameter packets, and the other is decoded from incoming ones. Setting the latter by hand may be needed to establish contact and also has the effect of redefining the default value for decoding from subsequent parameter packets. Generally, the distinction between SEND and RECEIVE parameters is unambiguous, the only exception being TIMEOUT (q.v.). The following SET subcommands are available in Kermit-370:

APPEND	Append if file name collision.
ATOE	Modify the Kermit-370 ASCII-to-EBCDIC table.
BLOCK-CHECK	Level of error checking for file transfer.
CONTROLLER	Indicate type of terminal connection.
DEBUG	Log packet traffic during file transfer.
DELAY	Length of pause before a SEND subcommand.
EOF	Text file truncation at CTRL-Z.
ETOA	Modify the Kermit-370 EBCDIC-to-ASCII table.
FILE	Attributes for incoming or outgoing files...
TYPE	... text or binary.
other	... system-specific attributes.
FOREIGN	Strings added to outgoing filespec...
PREFIX	
SUFFIX	
INCOMPLETE	Determine the action on an aborted file transfer.
LINE	Specify alternate communication line.
MARGIN	for sending files...
LEFT	
RIGHT	
PARITY	Indicate if 7-bit or 8-bit data.
RETRY	Maximum retry count...
INIT	... for initial packet exchange.
PACKET	... per packet for ongoing transfer.

SYSCMD	Try apparently invalid Kermit subcommands on host system.
TABS-EXPAND	Determine tab-to-space conversion on reception.
TAKE	
ECHO	Echo subcommands read from TAKE files.
ERROR-ACTION	Exit from TAKE file on command error.
TEST	Facilitate testing of Kermit.
TATOE	Modify the Kermit-370 ASCII-to-EBCDIC table.
TETOA	Modify the Kermit-370 EBCDIC-to-ASCII table.
TTABLE	Determine which tables undo the terminal translation.
WARNING	Rename if filename collision.
8-BIT-QUOTE	Determine state of 8th-bit prefixing.
SEND or RECEIVE	
END-OF-LINE	Packet terminator.
PACKET-SIZE	Maximum packet size.
PAD-CHAR	Character to insert before each packet.
PADDING	Number of pad characters to insert.
QUOTE	Use to quote control characters in packets.
START-OF-PACKET	Packet beginning marker.
TIMEOUT	Time limit for response.

SET APPEND

Syntax: `SET APPEND ON` *or* `OFF`

ON If an incoming file has the same name as an existing one, the new file is appended to the old one. This option supersedes SET WARNING.

OFF Filename collision is handled according to the WARNING parameter. (Default.)

SET ATOE etc.

Syntax: `SET` *table* `[`*num1 num2*`]`

This modifies one of the ASCII/EBCDIC translation tables used by Kermit-370 (for example, to conform to your system). The valid table names are ATOE, ETOA, TATOE, and TETOA. The arguments are, respectively, the offset within the named table and the new value for that offset. If the arguments are omitted, the table is restored to its initial arrangement. Both *num1* and *num2* should be in the range 0-255 (decimal). For example, in ATOE or TATOE, the offset is the ASCII character code, and the new value is the new EBCDIC result code. Initially, ATOE and TATOE each contain two identical copies of the 7-bit ASCII character table. Note: the meaning of the tables depends on the TTABLE setting -- if TTABLE is OFF, the TATOE and TETOA tables are not used.

SET BLOCK-CHECK

Syntax: `SET BLOCK-CHECK` *number*

This determines the type of block check used during file transfer, provided the other Kermit agrees. Valid options for *number* are: 1 (for a one-character checksum), 2 (for a two-character checksum) and 3 (for a three-character CRC). This is one of only two Send-Init parameters that cannot be SET separately for SEND and RECEIVE.

SET CONTROLLER

Syntax: `SET CONTROLLER` *type*

The *type* may be TTY, SERIES1, GRAPHICS, or FULLSCREEN. Kermit-370 automatically determines whether you are connected via a Series/1 (or similar) emulation controller or a TTY line. This subcommand is provided, though, to allow that choice to be superseded, and because Kermit may not be able to distinguish between Series/1-type and other 3270-emulation controllers. When CONTROLLER is set to SERIES1 or GRAPHICS, Kermit disables the 3270 protocol conversion function by putting the terminal controller into "transparent mode", which allows Kermit packets to pass through intact.

SET DEBUG

Syntax: `SET DEBUG ON` *or* `RAW` *or* `OFF`

ON Keep a journal of all packets sent and received in a log file on disk. If the file already exists, it is erased and overwritten. The *filespec* of the log is, of course, system-dependent. All packets are logged in EBCDIC for legibility, even when CONTROLLER is set to SERIES1 or GRAPHICS. In addition to the packets themselves, which are labelled "S" or "R" for packets sent or received, the log includes any additional status information (labelled "A"), such as the AID returned by a SERIES1 device.

RAW The same as ON, but packets are logged in the form that is passed to or from the operating system, i.e., EBCDIC for TTY terminals, and ASCII for SERIES1 and GRAPHICS terminals.

OFF Stop logging packets and close the the log file. (Default.)

SET DELAY

Syntax: `SET DELAY` *number*

Normally, Kermit-370 waits 10 seconds after the SEND subcommand before starting the transfer, but this delay may be SET to any non-negative value. Two DELAY values have special meaning. When DELAY is 1, the usual two-line greeting displayed during protocol mode is abbreviated to a short message (the default Kermit prompt with three dots...), and when DELAY is 0, the greeting is suppressed entirely, along with the extra one-second pause for the RECEIVE and SERVER subcommands.

SET EOF

Syntax: `SET EOF ON` *or* `OFF`

ON Scan each incoming TEXT file for the first occurrence of CTRL-Z and ignore the remainder of the file (but continue decoding up to the actual end of the file). BINARY files are not affected.

OFF Accept incoming files in their entirety. (Default.)

SET FILE TYPE

Syntax: `SET FILE TYPE TEXT` *or* `BINARY` *or* `V-BINARY` *or* `D-BINARY`

TEXT Specifies ordinary text. ASCII-to-EBCDIC or EBCDIC-to-ASCII translation is performed on the data. Trailing blanks are removed, and CRLF's are appended to outgoing records. CRLF's are used, in turn, to determine the end of incoming records, which are padded with blanks if necessary to fill buffers. (Default.)

BINARY Specifies bit-stream data. No translation is performed, no CRLF's are added to outgoing records, and blanks are neither added nor removed. Incoming bytes are added successively to the current record buffer, which is written out when the current LRECL is reached. Padding, if necessary, is done with nulls.

V-BINARY Specifies varying-length-record binary data. This type is like BINARY, except that a two-byte binary prefix is added to each outgoing record giving the number of data bytes, and incoming records are set off by (and stripped of) their prefixes on receipt.

D-BINARY Is like V-BINARY except that the length prefixes are five-byte ASCII-encoded decimal (right-justified with leading zeroes).

SET FOREIGN

Syntax: `SET FOREIGN PREFIX` *string*

This defines a prefix string to be added to the outgoing *filespec* generated by the SEND subcommand. For example, the string might be set to ‘‘`B:`’’ to specify output to the B disk drive on the other Kermit's system. The default is a null string. There is also a FOREIGN SUFFIX handled in the same manner.

SET HANDSHAKE

Syntax: `SET HANDSHAKE` *number*

This defines the character, if any, that Kermit-370 should send (or cause to be sent) immediately before reading each packet. The character is given as the decimal of an ASCII control character, or as zero if no handshake is to be sent. The default is 17 (XON), and any value in the range 0-31 is valid, but 13 (CR) should not be used because it is generally the end-of-packet character. When Kermit-370 is running through a full-duplex connection (such as a ‘‘`SERIES1`’’), the traditional IBM handshaking is not necessary, and HANDSHAKE should be set to 0 (as long as the other Kermit can be instructed not to expect a handshake). Note the distinction between `SET HANDSHAKE` in Kermit-370 (where it defines a character to be sent) and in many micro Kermits (where it defines a character to be expected).

SET INCOMPLETE

Syntax: `SET INCOMPLETE DISCARD` *or* `KEEP`

DISCARD Specifies that incomplete files (that is, files partially received in a transfer cancelled by the other Kermit) are to be erased. This is the default. Note that when APPEND is ON, incomplete files are never erased, lest pre-existing data be lost.

KEEP Specifies that incomplete files are to be kept.

SET LINE

Syntax: `SET LINE [`*name*`]`

This specifies an alternate communication line for file transfers. If the *name* is omitted, the default line (the user’s terminal) is used. The format of *name* is, of course, system-dependent, and some versions of Kermit-370 do not support any alternate lines. No version currently allows Kermit-370 to CONNECT over an alternate line.

SET MARGIN

Syntax: `SET MARGIN` *side column*

When Kermit-370 sends a text file, each line may be truncated on the left or right (or both) at fixed column numbers. Only the text from the left margin to the right margin (inclusive) will be sent, and any trailing blanks in the truncated lines will be stripped. A value of zero for either margin disables truncation on that side.

SET PARITY

Syntax: `SET PARITY MARK` *or* `NONE`

Transparent-mode ASCII data received from a ‘‘`SERIES1`’’ or ‘‘`GRAPHICS`’’ device will typically have either all Mark parity (seven data bits with the eighth bit set) or no parity (eight data bits). Kermit-370 must know which kind of parity to expect in order to calculate checksums properly. Since Kermit-370 does not actually verify parity, the other possible variants (ODD, EVEN, and SPACE) are lumped together with MARK parity for the purpose of this subcommand, which merely chooses between 7-bit and 8-bit data transfer. The default is MARK.

SET PROMPT

Syntax: `SET PROMPT` *string*

This defines the character string that Kermit-370 displays when asking for a subcommand. The prompt may be any string of up to 20 characters. The default is the name of the system-specific version of Kermit-370 followed by a ">" sign, e.g., `Kermit-CMS>`.

SET RETRY

Syntax: `SET RETRY INITIAL` *or* `PACKETS` *number*

Kermit-370 resends its last packet after receiving a NAK or bad packet, but it eventually gives up after repeated failures or the same packet. The limit on retries can be set separately for the initial packet exchange (Send-Init or server-mode command) and for ordinary packets. The default for INITIAL is 16 and for PACKETS, 5. Either limit can be set to any positive value.

SET SYSCMD

Syntax: `SET SYSCMD ON` *or* `OFF`

ON If the user enters a command string which is not a valid Kermit subcommand, Kermit-370 will pass the string along to the host operating system for execution. If the string is rejected by the system as well, Kermit will report it as an invalid *Kermit* subcommand. Otherwise, Kermit will assume the string was intended as a host command and will simply report the completion code if non-zero.

OFF Invalid Kermit subcommands are simply rejected as such. System commands may be executed, of course, but only by specifying the generic prefix ‘‘`HOST`’’ or the appropriate system-specific prefix, such as CMS or TSO. (Default.)

SET TABS-EXPAND

Syntax: `SET TABS-EXPAND ON [`*list*`]` *or* `OFF`

ON Tab characters in incoming TEXT files are replaced by one or more blanks to bring the record size up to the next higher multiple of eight for each tab. If tab settings other than columns 1, 9, 17, etc. are desired, they may be specified explicitly in a list following the keyword ‘‘`ON`’’. Items in the list may be separated by spaces or commas.

OFF Incoming tabs are retained. (Default.)

SET TAKE ECHO

Syntax: `SET TAKE ECHO ON` *or* `OFF`

ON Subcommands are echoed to the terminal as they are executed from a TAKE file.

OFF Subcommands from a TAKE file are executed "silently." (Default.)

SET TAKE ERROR-ACTION

Syntax: `SET TAKE ERROR-ACTION CONTINUE` *or* `HALT`

CONTINUE Execution continues in a TAKE file regardless of illegal commands, except in server mode. This is the default.

HALT A command error in a TAKE file causes immediate exit to Kermit subcommand level.

SET TEST

Syntax: `SET TEST ON` *or* `OFF`

ON Allow setting the START-OF-PACKET and other special characters to any value, and suppress checksum testing on received packets.

OFF Normal operation. (Default.)

SET TTABLE

Syntax: `SET TTABLE ON` *or* `OFF`

ON The translation that undoes the terminal controller's ASCII/EBCDIC conversion comes from the TATOE and TETOA tables, rather than the ATOE and ETOA tables (which are used only for translating disk files). This option has no effect when there is no translation built into the controller, i.e., with SERIES1 and GRAPHICS connections.

OFF The ATOE and ETOA tables are used for all translations by Kermit-370. (Default.)

SET WARNING

Syntax: `SET WARNING ON` *or* `OFF`

ON If an incoming file has the same *filespec* as an existing file on disk, Kermit will attempt to rename the incoming file so as not to destroy (overwrite) the pre-existing one.

OFF Upon filename collision, the existing file will be erased and replaced by the incoming file. (Default.)

SET 8-BIT-QUOTE

Syntax: `SET 8-BIT-QUOTE` *char or* `ON` *or* `OFF`

This controls whether eighth-bit prefixing is done and can be used to specify the character to be used. This is one of only two Send-Init parameters that cannot be SET separately for SEND and RECEIVE.

char Eighth-bit prefixing will be done using *char*, provided the other Kermit agrees.

ON Eighth-bit prefixing will be done, provided the other Kermit explicitly requests it (and specifies the character).

OFF Eighth-bit prefixing will not be done. (Default.)

SET SEND/RECEIVE

The following parameters can be set either as SEND or RECEIVE options. As a rule, in each pair, one is the operational value, and the other is used to change the default for Send-Init packets received from the other Kermit and to set up parameter values as if the other Kermit had specified them on the previous exchange. When both values are described, the operational one will be first. After a transfer, the operational values will be unchanged, but the others (as displayed by SHOW) will reflect the parameters specified by the other Kermit. The underlying defaults established by previous SET subcommands will still be in effect. In the syntax descriptions, *mode* is SEND or RECEIVE.

END-OF-LINE

Syntax: `SET` *mode* `END-OF-LINE` *number*

RECEIVE should not be changed.

SEND may be needed to establish contact. If the other system needs packets to be terminated by anything other than carriage return, specify the decimal value of the desired ASCII character. *number* must be in the range 0-31 (decimal). The default is 13 (CR).

PACKET-SIZE

Syntax: `SET` *mode* `PACKET-SIZE` *number*

RECEIVE defines *number* as the maximum length for incoming packets. The valid range is 26-9024, but 94 is the limit for normal short-packet protocol. The default is 80. In practice, the size may be limited by hardware and programming considerations. See the system-specific chapters for details.

SEND might be needed for sending files to a minimal Kermit that neither specifies a buffer size in the Send-Init sequence nor can accept the default (80). This parameter has no other function and is meaningful only in the range 26-94.

PAD-CHAR

Syntax: `SET` *mode* `PAD-CHAR` *number*

RECEIVE defines *number* as the character to be used by the other Kermit for padding packets. The character must be an ASCII control character (in the range 0-31). The default is 0 (NULL). This option is seldom useful.

SEND may be needed to establish contact if the other Kermit (or the transmission line) needs padded packets.

PADDING

Syntax: `SET` *mode* `PADDING` *number*

RECEIVE defines the *number* of pad characters to be used for padding packets from the other Kermit. This number may be anywhere from 0 to 94. The default is 0. This option is seldom useful.

SEND may be needed to establish contact if the other Kermit (or the transmission line) needs padded packets.

QUOTE

Syntax: `SET` *mode* `QUOTE` *char*

SEND indicates a printable character for prefixing (quoting) control characters and other prefix characters. The only good reason to change this would be for sending a file that contains many ‘‘#’’ characters (the normal control prefix) as data. It must be a single character with ASCII value 33-62 or 96-126 (decimal).

RECEIVE would be needed only for talking to a crippled Kermit that uses a non-standard quoting character, but does not admit it.

START-OF-PACKET

Syntax: `SET` *mode* `START-OF-PACKET` *number*

RECEIVE defines *number* as the character to be expected to mark the start of packets from the other Kermit. The character must be an ASCII control character (in the range 0-31). The default is 1 (SOH). This may be needed to establish contact.

SEND may also be needed to establish contact. It defines *number* as the character to be used to mark outgoing packets.

TIMEOUT

Syntax: `SET` *mode* `TIMEOUT` *time*

RECEIVE defines the *time* in seconds the other Kermit is to wait for a response from Kermit-370 before resending a packet. The default is 5. A value of 0 means the other Kermit should wait indefinitely.

SEND defines the *time* in seconds Kermit-370 is to wait for a response from the other Kermit before resending a packet. The default is 0.

The SHOW Subcommand

Syntax: `SHOW [`*option*`]`

The SHOW subcommand displays the values of all parameters that can be changed with the SET subcommand, except for ATOE, ETOA, TATOE, and TETOA (for those, see the TDUMP subcommand). If specified, *option* can be a particular parameter or the keyword ‘‘`ALL`’’ (the default). Groups of parameters, such as SEND, can be displayed by requesting the group name, or individual sub-parameters can be displayed by specifying the complete name. For example,

```
SHOW RECEIVE EOL
```

will display the decimal value of the packet terminator that Kermit-370 currently expects, i.e., 13. Similarly,

```
SHOW FOREIGN
```

will display the character strings currently in use for prefix and suffix on each outgoing *filespec*.

The STATUS Subcommand

Syntax: `STATUS`

This subcommand displays information about the previously executed subcommand. The response will include either the appropriate error message or the message ‘‘`No errors`’’. The initial status is ‘‘`No file transfers yet`’’. If the status reflects an error condition, the name of the last file used (excluding TAKE files) will be displayed as well. If the error was detected by the other Kermit, the message will be ‘‘`Micro aborted`’’ followed by the text from the Error packet. Conversely, if Kermit-370 detected the error, the text of the status message will have constituted the error packet sent out. In any case, if the last file transfer was cancelled (by virtue of an attribute mismatch or manual intervention), the reason for cancellation is displayed. Also, if the error occurred in disk I/O, any available explanatory information is displayed. Normally, the error status is altered only when a transfer-initiating subcommand (SEND or RECEIVE) is executed, but in server mode *every* subcommand is received through a transfer from the other Kermit and may affect the status (except the STATUS subcommand itself, of course). When Kermit-370 has been forced to truncate one or more records in a RECEIVE operation (because of the current maximum record length), the number of records truncated is reported. The status display also includes throughput statistics for the last transfer: number of files sent, duration, number of packets, number of retries, and averages of bytes/packet and bytes/second. These last two quantities are calculated separately for bytes sent and received on the communication (including padding, if any), and the last quantity is also calculated on the basis of the number of bytes read from or written to disk.

Finally, if retries were necessary, Kermit-370 computes the optimum packet size assuming the retries to have been due to sparse, Poisson-distributed bursts of noise. This is the same heuristic optimum that Kermit-370 computes and uses as an alternative packet-size limit when sending long packets.

The TDUMP Subcommand

Syntax: `TDUMP` *table-name or* `NAMES`

This displays the contents of *table-name*. The same table can be modified using the SET subcommand. The ATOE, ETOA, TATOE, and TETOA tables can presently be displayed and changed. Alternatively, the *filespec* of each file sent in the last transfer can be displayed.

The GIVE Subcommand

Syntax: `GIVE` *table-name filespec*

This compares the named table with its default values and saves the differences in the form of a TAKE file consisting of SET subcommands that would convert the default into the current arrangement. ATOE, ETOA, TATOE, and TETOA are the available tables. The details of the *filespec* are system-dependent, but those details will, in general, be the same as for the TAKE subcommand.

The HOST Subcommand

Syntax: `HOST` *text of command*

This issues a command to the host operating system from Kermit-370. When a command returns a non-zero completion code, the code will be displayed. Generally, the name of the system (e.g., CMS) is treated as a synonym for the HOST subcommand.

The KERMIT Subcommand

Syntax: `KERMIT` *text of subcommand*

This is provided for redundancy as the counterpart of the HOST subcommand. Kermit-370 executes the specified text as a Kermit subcommand just as if the LOCAL prefix had been entered.

The ECHO and XECHO Subcommands

Syntax: `[X]ECHO` *line*

These subcommands type the *line* back at the user. The *line* may contain control characters or any desired text, including upper or lower case. These subcommands may be used, for example, to test the ASCII/EBCDIC translate tables or to issue coded commands to the user's terminal. XECHO differs from ECHO primarily in that it uses transparent mode if CONTROLLER is SERIES1 or GRAPHICS. It also offers its own brand of control-character quoting, using the ‘‘`^`’’ character to indicate that only the five low-order bits of the ASCII codes are to be used. Thus, ‘‘`^a`’’, ‘‘`^A`’’, and ‘‘`^!`’’ are all translated to SOH (CTRL-A), while ‘‘`^[`’’ becomes ESC. However, there must be one exception for ‘‘`^`’’ itself: ‘‘`^>`’’ and ‘‘`^~`’’ are both translated to RS (CTRL-^), but ‘‘`^^`’’ becomes just ‘‘`^`’’.

The TYPE and XTYPE Subcommands

Syntax: `[X]TYPE` *filespec*

These subcommands type the named file. XTYPE differs from TYPE primarily in that it uses transparent mode if CONTROLLER is SERIES1 or GRAPHICS, and sends the data in bursts no larger than the current SEND PACKET-SIZE. TYPE is effectively a synonym for (and allows the same options as) the host system command for listing files, but XTYPE merely sends the file "raw".

8.3. Before Connecting to the Mainframe

Several flags must be set on the micro version of Kermit before connecting to an IBM 370 system as a ‘‘TTY’’ device. You should set the LOCAL-ECHO flag to ON (to indicate half-duplex). This is the norm but not true in absolutely every case; if each character appears twice on your terminal screen, set the LOCAL-ECHO flag OFF. FLOW-CONTROL should be set to NONE, and on some systems HANDSHAKE should be set to XON. The parity should be set according to the system’s specifications. On some micro versions of Kermit, all of the above is done in one step using the DO IBM macro (or SET IBM ON). Set the baud rate to correspond to the line speed.

Connecting through a ‘‘SERIES1’’ or ‘‘GRAPHICS’’ device also requires that certain flags be set on the micro version of Kermit. You should set the LOCAL-ECHO flag to OFF (to indicate full-duplex). FLOW-CONTROL should be set to XON/XOFF, and HANDSHAKE should be set to OFF. For many systems, the PARITY should be set to EVEN. Set the baud rate to correspond to the line speed.

One exception to these rules is the case where the micro Kermit is attempting automated file transfer, e.g., downloading several separate files from Kermit-370 running in server mode. In fact, under those circumstances, handshaking is necessary even with ‘‘SERIES1’’ connections, and the two Kermits must be instructed to adopt a common handshake character (e.g., by SET HANDSHAKE 10 to Kermit-370 and SET HANDSHAKE LF to the micro).

In any case, you should make sure that either the micro Kermit or Kermit-370 will provide timeouts during file transfers (if not both). Some versions of Kermit-370 (notably CMS) cannot provide timeouts, and you may need to set the TIMER to ON in the micro.

8.4. After Returning from Kermit-370

When Kermit-370 receives a QUIT or EXIT subcommand or finishes the subcommand or subcommands specified in the original command string that invoked Kermit, control is returned to the caller. Before returning, Kermit-370 closes any active TAKE files (the EXIT or QUIT subcommand may be issued from a TAKE file). On return, the completion code is set from the current error status according to the codes in Table 8-0.

8.5. What’s New

Below is a list of the additions in Version 4.0 of Kermit-370:

1. Code reorganization into generic 370 and system-specific sections.
2. Optional separate translation tables for counteracting the system conversion of terminal I/O.
3. New `GIVE` subcommand for saving a modified translation table.
4. A new, RAW debug mode for recording the packet traffic as actually sent and received on ‘‘GRAPHICS’’ and ‘‘SERIES1’’ devices.
5. Preservation of the case of subcommands as typed, with uppercase conversion of only those words that must be uppercase.
6. New `SET MARGIN` subcommand for limiting the width of a file to be sent.
7. Settable tab stops for Kermit’s conversion of tabs to spaces (alternative to the default 1, 9, 17, etc.).
8. Replace `SET SERIES1` subcommand with new `SET CONTROLLER`. Support for multiple terminal controller types.
9. New `DIRECTORY` and `HOST` subcommands following Kermit standard.
10. Combination of file-attribute `SET` subcommands (FILE-TYPE, LRECL, and RECFM) into a new group `SET FILE`.
11. Separate retry limits for initial and subsequent packet exchanges.

Code	Symbol	Error Message
0	NOE	No errors
1	NFT	No file transfers yet
2	TRC	Transfer cancelled
3	USC	Invalid server command
4	TIE	Terminal I/O error
5	BPC	Bad packet count or chksum
6	IPS	Invalid packet syntax
7	IPT	Invalid packet type
8	MIS	Lost a packet
9	NAK	Micro sent a NAK
10	ABO	Micro aborted
11	FNE	Invalid file name
12	FNF	File not found
13	FUL	Disk or file is full
14	DIE	Disk I/O error
15	MOP	Missing operand
16	SYS	Illegal system command
17	KCE	Kermit command error
18	TIM	No packet received
19	RTR	Records truncated
20	COM	Bad communication line

Table 8-1: Error messages and codes for Kermit-370

12. Pad binary records on disk with nulls, rather than blanks.
13. Automatically tune packet length when sending long packets according to heuristic optimum based on sparse Poisson statistics, provided that transmission errors do occur.
14. Expand STATUS report to include the number of files in the last transfer, throughput statistics, heuristic optimum packet length (when long packets are enabled), and the reason for any file rejection based on A-packets.
15. New subcommand `TDUMP NAMES` to display the list of files sent in the last transfer.
16. Add file creation date to A-packet repertoire.
17. `REMOTE COPY` and `REMOTE RENAME` commands to a server at the other end.
18. Allow long packets through a 7171 with VTAM.
19. New type `D-BINARY` for binary files with undelimited variable-length records.
20. `SET 8-BIT-QUOTE`. Allow 8-bit data where possible via `SET PARITY`.
21. `SET SYSCMD`, so that Kermit can be told to try "illegal" subcommands as host system commands instead of just rejecting them.
22. `SET PROMPT` subcommand.
23. Do not forget parameters specified by the other Kermit in I-packets.
24. Keep track of truncated records during a RECEIVE operation and report the count in STATUS; also call truncation an error after everything is received.
25. `SET HANDSHAKE` subcommand to alter or suppress handshake character Kermit-370 sends out after each packet.

Both `SEND` and `GET` prompt the user for native and foreign *filespecs* if no arguments are entered.

8.6. What's Missing

Work on Kermit-370 will continue. Features that need to be improved or added include:

- Detect file properties from Attribute packets and allow overriding current parameter settings. Also implement file archiving.
- Add SET REPEAT subcommand.
- Improve Kermit-370 operation as a local Kermit.
- Recover from sudden line degradation by retransmitting partial packets.
- System-specific upgrades; see the respective chapters for details.

9. IBM VM/CMS KERMIT

Program:	John Chandler (Harvard/Smithsonian Center for Astrophysics); contributions from Vace Kundakci and Daphne Tzoar (Columbia U), Bob Shields (U. Maryland), Victor Lee (Queens U.), Gary Bjerke (U. Texas at Austin), Greg Small (UC Berkeley), Clark Frazier (Harvard Bus. Sch.), Bob Bolch and Steve Blankinship (Triangle), Ron Rusnak (U. Chicago), Andre Pirard (U. Liege)
Language:	IBM/370 Assembler
Documentation:	John Chandler (CfA)
Version:	4.0 (88/1/31)
Date:	1988 February

Kermit-CMS Capabilities At A Glance:

Local operation:	No
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
`^X/^Z` interruption:	Yes (through micro)
Filename collision avoidance:	Yes
Can time out:	No
8th-bit prefixing:	Yes
Repeat count prefixing:	Yes
Alternate block checks:	Yes
Terminal emulation:	No
Communication settings:	No
Transmit BREAK:	No
Transaction logging:	Yes
Session logging:	No
Raw transmit:	Yes (no prompts)
Sliding window:	No
Long packets:	Yes
Act as server:	Yes
Talk to server:	Yes
Advanced server functions:	Yes
Advanced commands for servers:	Yes
Local file management:	Yes
Handle Attribute Packets:	Yes
Command/init files:	Yes
Command macros:	No

Kermit-CMS is a member of the generic Kermit-370 family and shares most of the features and capabilities of the group. As its name implies, Kermit-CMS is the version of Kermit-370 that runs under the VM/CMS operating system. The primary documentation for Kermit-CMS is actually the chapter on Kermit-370, which describes general properties; the present chapter assumes the reader is familiar with that material. Only the details specific to CMS operation will be discussed here, e.g., command syntax relating to the CMS file system or commands not offered in general by Kermit-370.

CMS Specifics of Kermit-370:

Global INIT file:	`SYSTEM KERMINI *`
User INIT file:	`<userid> KERMINI *`
Debug packet log:	`KER LOG A1`
Server reply log:	`KER REPLY A1`
Maximum packet size:	1913
Maximum disk LRECL:	65535

9.1. The VM/CMS File System

The features of the CMS file system of greatest interest to Kermit users are the format of file specifications (or *filespecs*) and the concept of records. The latter is described in the Kermit-370 chapter.

The VM/CMS *filespec* takes the form

```
filename filetype filemode
```

(often abbreviated FN FT FM). The filename and filetype are one to eight characters each. The name field is the primary identifier for the file, and the type is an indicator which, by convention, tells what kind of file it is. For instance, `TEST FORTRAN` is the source of a Fortran program named TEST. `MODULE` is the filetype for executable programs (as distinct from object code, which has a filetype of `TEXT`!). Although some operating systems consider the filetype optional, VM/CMS requires a type for each file. Therefore, Kermit-CMS supplies a default type of ‘‘`$`’’ for any received file if no type is provided by the remote system. The same default is used for a missing filename. At the same time, the FN and FT are forced to conform to CMS rules in other respects. The FN and FT may contain, in any order, uppercase letters, digits, and the special characters ‘‘`$`’’ (dollar sign), ‘‘`#`’’ (pound sign), ‘‘`@`’’ (at sign), ‘‘`+`’’ (plus), ‘‘`-`’’ (hyphen), ‘‘`:`’’ (colon), and ‘‘`_`’’ (underscore). Other characters may be not be included. If an invalid character is found in the FN or FT field, it is replaced by an underscore (or converted to uppercase if it is a lowercase letter). Also, both FN and FT are truncated, if necessary, to eight characters.

The filemode, which consists of a letter and a number, is similar to a device specification on microcomputer systems: `FN FT FM` would translate to `FM:FN.FT` in CP/M or MS-DOS if the filemode number is ignored. Indeed, the filemode number is more properly an attribute of a file than part of its name -- no two files can co-exist with names that match all but the filemode number. Even the filemode letter is not a fixed part of the *filespec* because the same mini-disk could be accessed under a different mode letter. In some ways, the filemode letter is also like a disk directory designator, since many such mini-disks may reside on the same disk drive. For this reason, the Kermit concept of the "working directory" is equated with a particular disk mode letter under Kermit-CMS. The current "working directory" is, thus, the "home" filemode (normally ‘‘`A`’’, which is the primary user mini-disk under CMS), and file transfers take place preferentially to and from the "home" disk. If the filemode is omitted from a *filespec* when sending, the "home" disk is normally used, but there is an option for using a default of ‘‘`*`’’ instead. In this case, the user’s disks are scanned according to the search order and the first occurrence of the file is the one that is sent. If the filemode is omitted from a *filespec* when receiving, the "home" disk is used with a filemode number of ‘‘`1`’’.

To provide compatibility with other operating systems, when Kermit-CMS sends a file, it ordinarily makes a file header with only the filename and filetype. It also converts the intervening blank to a period. On the other hand, extra information may be added by way of the SET FOREIGN subcommand.

VM/CMS allows a group of files to be specified in a single *filespec* by including the special "wildcard" characters ‘‘`*`’’ and ‘‘`%`’’. A ‘‘`*`’’ matches any string of characters (even a null string) from the current position to the end of the field; a ‘‘`%`’’ matches any single character. Here are some examples:

`* COBOL A`	All files of type `COBOL` (all COBOL source files) on the A disk.
`F* * *`	All files whose names start with F.
`% * B`	All B-disk files with one-character FN’s.

CMS files, like those in other IBM 370 systems, are record-oriented (see the introduction to the Kermit-370 chapter). In particular, CMS files are characterized by record format (RECFM), which may be fixed-length or varying-length, and by maximum record length (LRECL). The size of record blocks is irrelevant, however, because CMS performs the blocking and deblocking operations automatically and transparently, including the spanning of records across block boundaries. Records in CMS files may be up to 65535 bytes long.

Another file system feature of occasional interest is the means of reporting errors. When Kermit-CMS encounters a disk error, it records the function and error code for inclusion in the `STATUS` report. The explanations can be found in the CMS reference manual under the FSREAD and FSWRITE macros (which correspond to the RDBUF and WRBUF functions).

9.2. Program Operation

At startup time, Kermit-CMS looks for two initialization files, `SYSTEM KERMINI` and <*userid*> `KERMINI` (where <*userid*> is the user's logon ID). If either of these files exists on more than one disk, it will be read and executed from the first copy in the search order. The file `SYSTEM KERMINI` should be placed on a publicly accessible disk by a systems programmer, preferably the same disk where the Kermit executable module is kept. The file <*userid*> `KERMINI` can be maintained by the user on any convenient disk.

One important distinction between Kermit-CMS and other Kermits is that a program running under VM/CMS is unable to interrupt a read on its "console". This means that the CMS version of Kermit cannot time out after sending a packet. The only way to time out is from the other side: typing a carriage return to the local Kermit causing it to retransmit its last packet, or an automatic timeout as provided by most other Kermits.

Five CP SET parameters (MSG, IMSG, WNG, ACNT, and TIMER) are set OFF during protocol mode (and restored afterwards) to prevent CP from interrupting any I/O in progress, and RUN is set ON to ensure that Kermit can recover from accidental attention interrupts. Also, on a TTY line, the TERMINAL LINESIZE is set OFF to prevent CP from inserting carriage return-linefeed pairs into packets, TERMINAL SCROLL is set to CONT to prevent CP pauses, LINEDIT is set OFF to ensure that all characters are taken literally, and the CMS user terminal translation tables (established via the CMS SET INPUT and OUTPUT commands) are temporarily suppressed for both short and long packet protocols. The settings in effect when Kermit starts up are saved as a sort of "normal" status snapshot (as opposed to the "protocol" status just described). The protocol status is selected whenever Kermit enters protocol mode and also after Kermit executes a CP command in server mode. Similarly, normal status is selected when Kermit leaves protocol mode and before Kermit executes a CP command in server mode. Note: if Kermit is interrupted in the midst of a transfer or while in server mode, these parameters will be left with peculiar settings (namely, the protocol status), and they may need to be restored by hand.

CMS is different from some other IBM mainframe systems in that allows a program to take control of prompting and synchronization on ‘‘`TTY`’’ lines. Kermit-CMS takes advantage of this option, and it is not, in general, necessary to enable handshaking on the micro Kermit before connecting to CMS. In other words, handshaking should be suppressed for both ‘‘`TTY`’’ and ‘‘`SERIES1`’’ devices (the micro Kermit should have HANDSHAKE set OFF, and Kermit-CMS should have HANDSHAKE set to 0). Since the generic Kermit-370 default handshake (XON) is retained in Kermit-CMS, the subcommand ‘‘`SET HANDSHAKE 0`’’ is a good candidate for inclusion in `SYSTEM KERMINI`.

Interactive Operation:

To run Kermit-CMS interactively, invoke the program from CMS by typing `KERMIT`. When you see the prompt,

```
Kermit-CMS>
```

you may type a Kermit subcommand. When the subcommand completes, Kermit issues another prompt. The cycle repeats until you exit from the program. For example:

```
KERMIT

Kermit-CMS Version 4.0 (88/1/31)
Enter ? for a list of valid commands

Kermit-CMS>send foo *
```

Files with fn FOO are sent

```
Kermit-CMS>receive test spss
```

File is received and called TEST SPSS A1

```
Kermit-CMS>exit
```

The prompt string under CMS is truly interactive. In other words, the string (without carriage return or linefeed) appears only when fresh input is needed from the terminal. If, for example, Kermit is invoked after several subcommands have been stacked up, the stack is read and executed before the first prompt appears.

Command Line Invocation:

Kermit-CMS may also be invoked with command line arguments from CMS. The arguments are interpreted as one or more subcommands to be executed by Kermit after completion of the initialization. For instance:

```
KERMIT send test fortran
```

or

```
KERMIT set debug on # set file binary # server
```

Kermit will exit and return to CMS after completing the specified subcommand or subcommands. Note that several commands may be given on the command line as long as they are separated by the LINEND character, which is pound sign in this case. Note that the LINEND is a concept of CP, rather than Kermit, and applies only to commands entered from the terminal and only when LINEDIT is on. A command line may contain up to 130 characters.

EXEC Operation:

Like other CMS programs, Kermit-CMS may be invoked from a CMS EXEC. Subcommands can be passed to Kermit using the program stack and/or command line arguments. For example, to start up Kermit-CMS and have it act as a server, include the line:

```
KERMIT server
```

To pass more than one subcommand, they must be stacked in the order in which they are to be executed. To start up a Kermit-CMS server with a three character CRC, include:

```
&STACK set block 3
&STACK server
KERMIT
```

Another way of setting up multiple subcommands would be to collect the subcommands into a TAKE file and then issue the TAKE subcommand via the command line or program stack. Of course, EXEC's may be executed from Kermit, either directly or from a TAKE file, and Kermit subcommands, in turn, may be issued from EXEC's as long as Kermit is active. See the TAKE subcommand for more details.

Server mode:

Command execution in server mode is different in several respects from normal operation. First of all, some Kermit subcommands are not allowed (see the list of subcommands in the Kermit-370 chapter). Moreover, command errors always terminate any active TAKE file. Also, commands other than CP commands run in a special environment with RUN ON, TIMER OFF, and so forth. Another difference is that Kermit intercepts all SVC instructions in order to catch console I/O and transmit the data to the local Kermit as text packets. Since Kermit does not emulate the substitution functions of the LINEDIT macro, some messages will appear rather cryptic. A more serious problem with this redirection is that some VM/CMS system commands may issue console I/O directly to CP, so that some messages never appear to the local Kermit (except, perhaps, as bad packets). For non-TTY terminals, such messages are stacked up in the console output queue and appear all at once when Kermit returns from server mode.

9.3. Kermit-CMS Subcommands

Kermit-CMS supports all the subcommands described in the corresponding section of the Kermit-370 chapter. In addition, there are two more, both of which can be issued as remote Kermit commands when Kermit-CMS is in server mode. The first is `CMS`, which is just a synonym for the generic HOST subcommand. The second is `CP`, which specifically issues a command to CP. In most circumstances, the latter is not needed, since CMS will pass along CP commands to CP.

The remainder of this section concentrates on the subcommands that have special form or meaning for Kermit-CMS. See also the chapter on Kermit-370 for further details.

The SEND Subcommand

Syntax: `SEND` *filespec* [*foreign-filespec*]

The SEND subcommand causes a file or file group to be sent from CMS to the Kermit on the other system. *filespec* takes the form:

```
filename filetype [filemode]
```

but the filemode if optional only if the *foreign-filespec* is omitted.

filespec may contain the wildcard characters ‘‘*’’ or ‘‘%’’. If *filespec* contains wildcard characters then all matching files will be sent. If, however, a file exists by the same name on more than one disk, only the first one Kermit-CMS encounters, according to the disk search order, is sent. See also the CWD subcommand.

The *foreign-filespec*, if any, is used for the file header of the outgoing file, replacing the usual filename.filetype copied from the CMS *filespec*. It may take one of two forms:

```
filename filetype
```

or

```
arbitrary-string
```

Normally, this form of the SEND subcommand is used only for single files because the *foreign-filespec* is used only for the first file of a group (subsequent files having default headers). However, in the two-token form of the *foreign-filespec* either the name or type may be an Equals sign ‘‘=’’ to signify that the corresponding CMS name or type is to be retained in the file header. In that case, the partial renaming carries through an entire group of files. It is the user’s responsibility to prevent such partial renaming from sending duplicate file headers within a file group.

Although the file transfer cannot be cancelled from the CMS side, Kermit-CMS is capable of responding to "cancel file" or "cancel batch" signals from the local Kermit; these are typically entered by typing Control-X or Control-Z, respectively.

The RECEIVE Subcommand

Syntax: `RECEIVE` [*filespec*]

The RECEIVE subcommand tells Kermit to receive a file or file group from the other system. You should then issue a SEND subcommand to the other Kermit.

The format of *filespec* is:

```
filename filetype [filemode]
```

If the optional *filespec* is omitted, Kermit-CMS will use the name(s) provided by the other Kermit. If that name is not a legal CMS file name, Kermit-CMS will delete excess characters and will change illegal characters to underscores. A *filespec* in the subcommand indicates what name the incoming file should be given. The *filespec* may include a filemode to designate the destination disk. If none is provided, the file will be saved on the "home" disk with filemode number ‘‘`1`’’. If you want to use the same name but a different filemode, specify ‘‘`= = FM`’’. Wildcards may not be used.

If the optional *filespec* is provided, but more than one file arrives, the first file will be stored under the given *filespec*, and the remainder will be stored under their own names on the "home" disk. If, however, ‘‘`= = FM`’’ is used, all files will be placed onto the specified disk.

When the record format is "F", any received record longer than the logical record length (LRECL) will be truncated, and shorter records will be padded. The padding character is a blank for text files and a null for binary files. Received binary (but not V-binary or D-binary) files are treated as byte streams and broken up into records all of the logical record length. See the SET FILE TYPE, SET FILE LRECL, and SET FILE RECFM subcommands.

If an error occurs during the file transfer, as much of the file as was received is saved on disk. If the sending of a file is cancelled by the user of the foreign system, Kermit-CMS will discard whatever had arrived, unless APPEND is ON or INCOMPLETE is KEEP.

If the incoming file has the same name as an existing file, and WARNING is OFF, the original file will be overwritten. If WARNING is set ON, however, Kermit-CMS will change the incoming name so as not to obliterate the pre-existing file. It attempts to find a unique name by successively modifying the original and checking for the existence of such a file at each step. The procedure begins by truncating the filetype to six characters if necessary, and then appending ‘‘`$0`’’. If a file by that name exists, Kermit then replaces the ‘‘`0`’’ with a ‘‘`1`’’. It continues in this manner up to ‘‘`9`’’, and if an unused name cannot be found, the transfer fails.

The GET Subcommand

Syntax: `GET` *foreign-filespec* `[`*filespec*`]`

The GET subcommand tells Kermit to request a file or file group from the other system, which must have a Kermit running in server mode. The syntax is complicated by the allowance of two forms for the *foreign-filespec*, just as in the SEND subcommand. Here the parsing is based on the number of "words" (blank-delimited strings) in the subcommand argument, which can be anything from one to five. If the number is anything but four, the interpretation is unambiguous, but when there are four words, the first word plays the key role. If it has more than eight characters or contains a ‘‘`.`’’ or ‘‘`/`’’, it is assumed to be the whole *foreign-filespec*; otherwise, it is assumed to be the first of two words that, when joined by a ‘‘`.`’’, make up the *filespec* on the other system.

The TAKE Subcommand

Syntax: `TAKE` *filespec*

Execute Kermit subcommands from the specified file, where *filespec* has the format `fn [ft [fm]]`. The default filetype is ‘‘`TAKE`’’, and the default filemode is ‘‘`*`’’.

Kermit subcommands may also be executed from CMS EXEC’s, so that the TAKE subcommand is, in a sense, superfluous under VM/CMS. In CMS terminology, Kermit establishes a Kermit subcommand environment, and EXEC’s written in EXEC 2 or REXX may invoke subcommands within that environment. For example, to display the current packet checksum type, an EXEC 2 would issue

```
&SUBCOMMAND KERMIT SHOW BLOCK-CHECK
```

and a REXX macro would issue

```
Address KERMIT 'SHOW BLOCK-CHECK'
```

There is one important difference between executing a TAKE file and an EXEC: the former may issue a QUIT or EXIT subcommand, but the latter may not. Also, a Kermit subcommand issued from an EXEC returns a completion code according to the current error status (see the table under "After Kermit Completes" in the Kermit-370 chapter). An EXEC could therefore be set up to react appropriately to file transmission errors or other unpredictable events.

The SET Subcommand

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

The SET subcommand establishes or modifies various parameters controlling file transfers. The following SET parameters are available in Kermit-CMS, but not in Kermit-370 in general:

DESTINATION	"Home" disk.
FILE	
LRECL	Logical Record length for incoming file.
RECFM	Record format for incoming files.
SEARCH-ALL	Determine the default disk search scope.

SET DESTINATION

Syntax: SET DESTINATION *letter*

This subcommand is equivalent to the CWD subcommand (q.v.).

SET FILE LRECL

Syntax: `SET FILE LRECL` *number*

This sets the logical record length for incoming files to a *number* from 1 to 65535 (64K-1). This variable is used only for fixed format and binary files. The default is 80.

SET FILE RECFM

Syntax: `SET FILE RECFM` *option*

This sets the record format to use for incoming files. Valid *option*s are "Fixed" and "Variable" (the default). Fixed-format records are padded or truncated, as needed, to the current LRECL.

SET SEARCH-ALL

Syntax: `SET SEARCH-ALL ON` *or* `OFF`

ON If the user omits the filemode from a SEND subcommand (or a GET request to the other Kermit), Kermit-CMS will search all accessed disks for the named file or files.

OFF If the filemode is not specified, only the "home" disk and its read-only extensions will be searched for matching files. (Default.)

The CWD Subcommand

Syntax: `CWD` *letter*

The CWD (Change Working Directory) subcommand establishes a new default ("home") CMS disk. *letter* may be the mode letter of any accessed disk. Subsequent file transfers take place preferentially to and from the default disk. The initial home disk is ‘‘`A`’’. Note: setting the home disk in Kermit has no effect on the CMS search order.

The DIRECTORY Subcommand

Syntax: `DIRECTORY [`*filespec*`]`

Under Kermit-CMS, the DIRECTORY subcommand is identical to the CMS LISTFILE command.

The SPACE Subcommand

Syntax: `SPACE [`*letter*`]`

This subcommand displays the storage allocation on the specified CMS disk. If *letter* is omitted, the default disk specified by the CWD subcommand is displayed. Aside from this default, the subcommand is identical with CMS QUERY DISK.

The GIVE Subcommand

Syntax: `GIVE` *table-name filespec*

This subcommand compares the named table with its default values and saves the differences in a TAKE file named *filespec*. The format of *filespec* is `fn [ft [fm]]`. The default filetype is ‘‘`TAKE`’’, and the default filemode is that of the "home" disk. See the CWD subcommand.

The CP and CMS Subcommands

Syntax: `CP` *or* `CMS` *text of command*

Although Kermit-CMS does not have a full set of its own subcommands for managing local files, it provides those services through the operating system. You can issue any CP or CMS command, but if Kermit-CMS has been invoked as a normal user-area program, rather than as a high-memory "resident" program or nucleus extension, other user-area CMS commands (such as COPYFILE) are illegal. Even then, you can list, type, rename or delete files, send messages, and so on. The `CMS` subcommand under Kermit is synonymous with the `HOST` subcommand.

9.4. How to build an executable version of Kermit-CMS

Before attempting to build Kermit-CMS, look in the Kermit distribution under both IK0KER and IKCKER for an installation document, as well as "beware", help, and update files, and read them first. They will probably contain information that is more current than what you see here.

Kermit-CMS consists at present of a large assembly and a small optional one. The large assembly (KERMIT ASSEMBLE) contains the Kermit program, and the small one (KERMBOOT ASSEMBLE) is a bootstrap program for loading Kermit into high memory and running it. Although KERMBOOT is all in one file in the Kermit distribution, the source for Kermit itself is in many pieces, some generic for Kermit-370 and some specific to CMS.

All the necessary pieces are sequenced in columns 73-80 so that the numbers form a strictly increasing sequence when the pieces are correctly "pasted" together. It is important to preserve the original sequence numbers so that updates, if any, can be applied to the source.

To create a runnable version:

1. Combine the following ‘‘`ASM`’’ files from the Kermit distribution into a single file with `RECFM F` and `LRECL 80`: IK0DOC, IK0MAC, IKCMAC, IK0DEF, IK0MAI, IK0CMD, IK0COM, IKCUTL, and IK0PRO. The resulting file is the composite source for Kermit-CMS, called KERMIT ASSEMBLE. This source must retain the original sequence numbers in columns 73-80 (in other words, be sure not to resequence the source accidentally by using the editor!)
2. Copy or rename IKCBOO ASM from the Kermit distribution (if desired) to a file called KERMBOOT ASSEMBLE with `RECFM F` and `LRECL 80`.
3. GLOBAL the necessary MACLIBs. Under VM/SP, these are DMSSP, CMSLIB, OSMACRO, and TSOMAC.
4. Assemble the source file(s).
5. Load one file into memory via: `LOAD KERMIT` or `LOAD KERMBOOT`. In the former case, the entire Kermit program is now loaded; in the latter, only a bootstrap program which expects to find the object file `KERMIT TEXT` at run time. Under CMS/SP Release 4 and above, there is a third and better option, namely, `LOAD KERMIT (RLDSAVE)`.
6. Create the executable called `KERMIT MODULE` via: `GENMOD KERMIT`. Alternatively (under CMS/SP Release 3 and below), create both KERMIT and KERMBOOT modules to give the user a choice of user-area or high-memory execution. Since Kermit-CMS is serially reusable, it can be reinvoked in the user area with the START command, but the high-memory version must be reloaded each time. If Kermit is loaded using the RLDSAVE option (Release 4 and above), the module can, in fact, be run either way; the command `NUCXLOAD KERMIT` will load Kermit "permanently" into high memory as a nucleus extension for invocation at need. Note: the nucleus extension can be removed by the command `NUCXDROP KERMIT`.
7. If your site’s ASCII/EBCDIC translation table for TTY lines does not conform to the one listed in the appendix (which in turn conforms to the one given in the IBM System/370 Reference Summary), then enter the appropriate SET ATOE/ETOA/TATOE/TETOA subcommands in the SYSTEM KERMINI file, which should reside on the same disk as KERMIT MODULE (and KERMIT TEXT). *NOTE:* If the ASCII/EBCDIC translation is not invertible, Kermit will not and cannot work.

9.5. What’s New

Below is a list of the more important CMS-specific features in Version 4.0 of Kermit-CMS added since the previous release, Version 3.1, in September 1986. For the list of generic additions, see the chapter on Kermit-370.

1. System commands issued through Kermit via the `CMS` or `HOST` subcommands are automatically passed on to CP if (a) CMS rejects them and (b) IMPCP is set ON.
2. Kermit subcommands may be executed directly from CMS EXEC’s.
3. Reject files known (via A-packets) to be too big for available storage.
4. Bypass user translation tables and set TERMINAL SCROLL CONT for protocol mode on TTY lines.
5. KERMBOOT avoids the loading problem (`VIRTUAL STORAGE CAPACITY EXCEEDED`) due to large GLOBAL TXTLIB’s and preserves the untokenized command line so that Kermit may be given mixed-case or long words as part of the initial command.

9.6. What's Missing

Work on Kermit-CMS will continue. Features that need to be improved or added include:

- Allow timeouts so Kermit-CMS does not wait forever if a packet does not arrive in a timely fashion. This is not possible under CMS at present.
- Detect file properties from Attribute packets and allow overriding current parameter settings. Also implement file archiving.
- Add a `SET REPEAT` subcommand.
- Finish `SET LINE`, so that Kermit-CMS can be used as a local Kermit, connecting to a remote host over an alternate communication port. Add a `CONNECT` subcommand.
- Intercept CP messages during protocol mode, rather than just suppressing them. Display the messages later or log them or send in packets as appropriate.
- Define EXEC variables from Kermit by analogy with the XEDIT EXTRACT subcommand.

10. IBM MVS/TSO KERMIT

Program:	John Chandler (Harvard/Smithsonian Center for Astrophysics); contributions from Vace Kundakci and Daphne Tzoar (Columbia U), Bob Shields (U. Maryland), Victor Lee (Queens U.), Gary Bjerke (U. Texas at Austin), Greg Small (UC Berkeley), Clark Frazier (Harvard Bus. Sch.), Bob Bolch and Steve Blankinship (Triangle), Ron Rusnak (U. Chicago), Roger Fajman and Dale White (NIH), Andre Pirard (U. Liege)
Language:	IBM/370 Assembler
Documentation:	John Chandler (CfA)
Version:	4.0 (88/1/31)
Date:	1988 February

Kermit-TSO Capabilities At A Glance:

Local operation:	No
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
`^X/^Z` interruption:	Yes (through micro)
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	Yes
Alternate block checks:	Yes
Terminal emulation:	No
Communication settings:	No
Transmit BREAK:	No
Transaction logging:	Yes
Session logging:	No
Raw transmit:	Yes (no prompts)
Sliding window:	No
Long packets:	Yes
Act as server:	Yes
Talk to server:	Yes
Advanced server functions:	Yes
Advanced commands for servers:	Yes
Local file management:	Yes
Handle Attribute Packets:	Yes
Command/init files:	Yes
Command macros:	No

Kermit-TSO is a member of the generic Kermit-370 family and shares most of the features and capabilities of the group. As its name implies, Kermit-TSO is the version of Kermit-370 that runs under the MVS/TSO operating system. The primary documentation for Kermit-TSO is actually the chapter on Kermit-370, which describes general properties; the present chapter assumes the reader is familiar with that material. Only the details specific to TSO operation will be discussed here, e.g., command syntax relating to the TSO file system or commands not offered in general by Kermit-370.

TSO Specifics of Kermit-370:

Global INIT file:	`'SYS1.KERMINI'`
User INIT file:	`KERMINI`
Debug packet log:	`KER.LOG`
Server reply log:	`KER.REPLY`
Maximum packet size:	1913
Maximum disk LRECL:	32756

10.1. The MVS/TSO File System

The features of the TSO file system of greatest interest to Kermit users are the format of file specifications (*filespecs*) and the concept of records. The latter is described in the Kermit-370 chapter.

The MVS/TSO *filespec* (called the data set name or DSN) takes the form of tokens (known as qualifiers) of up to 8 alphanumeric characters each, separated by periods. Each qualifier must begin with an alphabetic or national character. The total length must be no more than 44 characters, including periods. To be precise, a DSN may contain uppercase letters, digits, and the special characters ‘‘`$`’’ (dollar sign), ‘‘`#`’’ (pound sign), ‘‘`@`’’ (at sign), and ‘‘`-`’’ (hyphen) in addition to the separating periods. Other characters may be not be included.

There is another, structural restriction on data set names from the fact that TSO data sets are all cataloged. In a standard MVS catalog, each qualifier of each DSN is associated with an index of the same name. The index is a hierarchical catalog component which points downward either to a list of next-level indices or to a data set, but never to both. Consequently, a cataloged DSN is a chain of indices corresponding one-for-one with DSN qualifers, and the last index is a pointer to the data set itself. While there may be many indices with the same name, no two such duplicates may both be chained to the same next-higher-level index, so that, if DSN’s are considered as strings of qualifiers (not of characters), no DSN can be a major substring of any other. For example, if the name ’`A.BB.C`’ exists in the catalog, then ’`A.BB`’ and ’`A.BB.C.X`’ are illegal, but ’`A.B`’, ’`A.BB.Y`’, ’`A.BBB`’, and ’`A.BB.BB`’ are all legal.

A DSN given in its entirety (as in the foregoing examples) is called "fully qualified" and must be enclosed in single quotes when entered in TSO. However, by convention (and by definition) the first qualifier of each data set belonging to a given user must be the user’s logon ID, and, by default, that ID is the assumed prefix in TSO when a DSN is *not* enclosed in quotes. In practice, then, names are abbreviated by omitting the quotes and the prefix. The most common type of name, in fact, consists of the prefix plus two more qualifiers giving the data set’s name and type, respectively, so that many files have DSN’s that correspond exactly to the canonical Kermit representation of a *filespec*. For instance, `TEST.FORT` is the source of a Fortran program named TEST, and its fully qualified DSN would be ’<*userid*>`.TEST.FORT`’, where <*userid*> is the owner’s logon ID.

While this description is complete as far as it goes, it omits an important feature that is widely used in organizing files under MVS and TSO, namely, the partitioned data set (PDS). A PDS is a data set like any other, except that it has *members*, each of which can usually be treated as a file in its own right. In fact, with QSAM (the file access method employed by Kermit and many other applications), only members, and not the whole PDS, may be read or written. Thus, the term "file", as used in this chapter and the Kermit-370 chapter, may refer to either a PDS member or an ordinary data set, but not to a PDS. The notation for a PDS member consists of the member name enclosed in parentheses and appended to the DSN (and the whole enclosed in quotes if the DSN is fully qualified). For example, if the Fortran program `TEST` were copied into a PDS called `DEBUG.FORT`, it would then be known as

```
DEBUG.FORT(TEST)
```

Although the member name is written together with the DSN, it and the surrounding parentheses are not really part of the DSN and are not counted toward the 44-character limit. Still, a member name must conform to the rules for a DSN qualifier and, in addition, must not contain any hyphens.

All these properties of DSN’s come into play when a file is being received by Kermit-TSO because a valid DSN must be generated for the new data set. For example, any invalid character in the supplied *filespec* is replaced by a pound sign (or converted to uppercase if it is a lowercase letter). Also, each qualifier (and the member name, if any) is prefixed with a pound sign if it does not already begin with an alphabetic or national character and then shortened, if necessary, to eight characters. If no *filespec* is supplied, Kermit-TSO creates a default DSN of ‘‘`$.$`’’. The DSN is expanded to its fully qualified form and then truncated at 44 characters, if need be.

The DSN prefix, which defaults to the user’s logon ID, is similar to a device specification on microcomputer systems: it selects an area of disk storage, and it usually need not be specified. In some ways, the prefix is also like a disk directory designator, since the file system structure is hierarchical. For this reason, the Kermit concept of the "working directory" is equated with a particular DSN prefix under Kermit-TSO. The current "working directory" is the collection of all data sets whose names begin with the current prefix, and file transfers take place preferentially to and from that area (i.e., unless a fully qualified DSN is given).

To provide compatibility with other operating systems, when Kermit-TSO sends a file, it ordinarily makes a file header with only the last two qualifiers of the full DSN (or only the member name in the case of a PDS member). On the other hand, extra information may be added by way of the SET FOREIGN subcommand.

Kermit-TSO allows a group of files to be specified in a single *filespec* by including the special "wildcard" character ‘‘*’’, which matches any string of characters (even a null string) starting at a new DSN qualifier. Only one * may be used in a *filespec*. Also, the * may not be used for the member name of a PDS (nor for any part of a PDS name). Here are some examples:

`*.COBOL` All files of type `COBOL` (all COBOL source files) in the current working directory.

`BATCH*H.FORT`
: All files in the current directory which begin with the qualifier `BATCH` and which end with `H.FORT`. This would not include `BATCH.FORT`, however.

TSO files, like those in other IBM 370 systems, are record-oriented (see the introduction to the Kermit-370 chapter). In particular, TSO files are characterized by record format (RECFM), which may be fixed-length, varying-length, or undefined-length; by maximum record length (LRECL); and by maximum block size (BLKSIZE). Fixed-length and varying-length may be (and, under Kermit, always are) combined into blocks, but undefined-length records may not. Indeed, by convention, they have no logical record length, only a maximum block size. Records in TSO files may be up to 32760 bytes long, but varying-length records use four bytes to specify the length and have an effective limit of 32756.

Another file system feature of occasional interest is the means of reporting errors. When Kermit-TSO encounters a disk error, it attempts to prepare an explanatory message for inclusion in the `STATUS` report. The primary method is the standard `SYNADAF` macro.

10.2. Program Operation

At startup time, Kermit-TSO looks for two initialization files, ’`SYS1.KERMINI`’ and ’<*userid*>`.KERMINI`’ (where, as before, <*userid*> is the user’s logon ID). The latter file would also be known as just `KERMINI`. The file ’`SYS1.KERMINI`’ would be maintained by a systems programmer, but `KERMINI` would be maintained by the user.

Two parameters in the user’s profile (the character delete and line delete) are disabled during protocol mode (and restored afterwards) to prevent any conflict in case either of these characters has been defined to be printable. The settings in effect when Kermit starts up are saved as a sort of "normal" status snapshot (as opposed to the "protocol" status just described). The protocol status is selected whenever Kermit enters protocol mode, and the normal status is selected when Kermit leaves protocol mode. Note: if Kermit is interrupted in the midst of a transfer or while in server mode, these parameters will be left with peculiar settings (namely, the protocol status), and they may need to be restored by hand.

Although TSO does not allow an application program to take control of terminal synchronization on ‘‘`TTY`’’ lines, the various full-screen emulation front ends are quite a different matter. The standard IBM handshake (XON) is unnecessary, for example, with a 7171 or 4994 because the front end itself turns the line around with essentially no delay in transparent mode. Thus, handshaking should be suppressed for ‘‘`SERIES1`’’ devices (the micro Kermit should have HANDSHAKE set OFF, and Kermit-TSO should have HANDSHAKE set to 0). Since the generic Kermit-370 default handshake (XON) is retained in Kermit-TSO, the subcommand ‘‘`SET HANDSHAKE 0`’’ is a good candidate for inclusion the KERMINI file of any user who habitually uses ‘‘`SERIES1`’’ lines.

Interactive Operation:

To run Kermit-TSO interactively, invoke the program from TSO by typing KERMIT. When you see the prompt,

```
Kermit-TSO>
```

you may type a Kermit subcommand. When the subcommand completes, Kermit issues another prompt. The cycle repeats until you exit from the program. For example:

```
KERMIT

Kermit-TSO Version 4.0 (88/1/31)
Enter ? for a list of valid commands

Kermit-TSO>send foo.*
```

Files beginning with FOO are sent

```
Kermit-TSO>receive test.spss
```

File is received and called TEST.SPSS

```
Kermit-TSO>exit
```

Command Line Invocation:

Kermit-TSO may also be invoked with command line arguments from TSO. The arguments are interpreted as a subcommand to be executed by Kermit after completion of the initialization. For instance:

```
KERMIT send test.fort
```

Kermit will exit and return to TSO after completing the specified subcommand.

CLIST Operation:

Like other TSO programs, Kermit-TSO may be invoked from a CLIST. Subcommands can be passed to Kermit using the program input stack and/or command line arguments. For example, to start up Kermit-TSO and have it act as a server, include the line:

```
KERMIT server
```

To pass more than one subcommand, they must be stacked in the order in which they are to be executed. To start up a Kermit-TSO server with a three character CRC, create and stack a file with the following:

```
set block 3
server
```

and then invoke Kermit. Like many utility programs, Kermit-TSO uses the GETLINE/PUTLINE service routines for terminal I/O, and the nominally interactive subcommands can thus be supplied under program control. Another way of setting up multiple subcommands would be to collect the subcommands into a TAKE file and then issue the TAKE subcommand via the command line. Of course, CLIST's may be executed from Kermit, either directly or from a TAKE file, and CLIST's in turn may freely issue Kermit subcommands. The subcommand KERMIT is especially useful in this context for distinguishing Kermit subcommands from TSO commands.

Server mode:

Command execution in server mode is different in several respects from normal operation. First of all, some Kermit subcommands are not allowed (see the list of subcommands in the Kermit-370 chapter). Moreover, command errors always terminate any active TAKE file. Also, commands run in a special environment with the User Profile temporarily modified. Another difference is that Kermit intercepts terminal I/O as much as possible and transmits the data to the local Kermit as text packets. The problem with this redirection is that some MVS/TSO commands issue terminal I/O directly, so that some messages never appear to the local Kermit (except, perhaps, as bad packets).

10.3. Kermit-TSO Subcommands

Kermit-TSO supports all the subcommands described in the corresponding section of the Kermit-370 chapter. In addition, there is the system-specific subcommand `TSO`, which is just a synonym for the generic `HOST` subcommand. `TSO` can be issued as a remote Kermit command when Kermit-TSO is in server mode. Also, the `END` subcommand is available as a synonym for `EXIT` and `QUIT`.

The remainder of this section concentrates on the subcommands that have special form or meaning for Kermit-TSO. See also the chapter on Kermit-370 for further details.

The SEND Subcommand

Syntax: `SEND` *filespec* [*foreign-filespec*]

The SEND subcommand causes a file or file group to be sent from TSO to the Kermit on the other system. DSN prefixing is done on the *filespec* in the usual way (see also the CWD subcommand).

filespec may contain a wildcard ‘‘`*`’’. If it does, then all matching files will be sent, up to 711 files in all.

The *foreign-filespec*, if any, is used for the file header of the outgoing file, replacing the usual name.type derived from the MVS/TSO *filespec*. Normally, this form of the SEND subcommand is used only for single files because the *foreign-filespec* is used only for the first file of a group (subsequent files having default headers).

Although the file transfer cannot be cancelled from the TSO side, Kermit-TSO is capable of responding to "cancel file" or "cancel batch" signals from the local Kermit; these are typically entered by typing Control-X or Control-Z, respectively.

The RECEIVE Subcommand

Syntax: `RECEIVE` [*filespec*]

The RECEIVE subcommand tells Kermit to receive a file or file group from the other system. You should then issue a SEND subcommand to the other Kermit.

A *filespec* in the subcommand indicates what name the incoming file should be given. Wildcards may not be used. If the *filespec* is invalid, Kermit-TSO will suppress the transfer. If the optional *filespec* is omitted (and, in any case, for all files after the first in a group) Kermit-TSO will use the name(s) provided by the other Kermit. If a name is not a legal DSN, Kermit-TSO will delete excess characters, change illegal characters to pound signs, and so on, to create a legal name.

When the record format is ‘‘`F`’’, any received record longer than the logical record length (LRECL) will be truncated, and shorter records will be padded. The padding character is a blank for text files and a null for binary files. Received binary (but not V-binary or D-binary) files are treated as byte streams and broken up into records all of the logical record length. See the SET FILE TYPE, SET FILE LRECL, SET FILE BLKSIZE, and SET FILE RECFM subcommands.

If an error occurs during the file transfer, as much of the file as was received is saved on disk. If the sending of a file is cancelled by the user of the foreign system, Kermit-TSO will discard whatever had arrived, unless APPEND is ON or INCOMPLETE is KEEP.

If the incoming file has the same name as an existing file (either a data set or a PDS member), and WARNING is OFF, the original file will be overwritten. If WARNING is set ON, however, Kermit-TSO will protect the existing file in one of two ways. If the *filespec* was entered with the subcommand, Kermit will prompt the user for permission to overwrite the file. If the *filespec* came from the foreign Kermit, Kermit-TSO will change the incoming name so as not to obliterate the pre-existing file. It attempts to find a unique name by successively modifying the original and checking for the existence of such a file at each step. The procedure operates on the second qualifier of the full DSN (or the member name in the case of a PDS member) and begins by truncating it to seven characters, if necessary, and then appends ‘‘`0`’’. If a file by that name exists, Kermit then replaces the ‘‘`0`’’ with a ‘‘`1`’’. It continues in this manner up to ‘‘`9`’’, and if an unused name cannot be found, the transfer fails.

The SET Subcommand

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

The SET subcommand establishes or modifies various parameters controlling file transfers. The following SET parameters are available in Kermit-TSO, but not in Kermit-370 in general:

FILE	
BLKSIZE	Block size for incoming file.
LRECL	Logical Record length for incoming file.
RECFM	Record format for incoming files.
SPACE	Allocation unit (in tracks) for incoming files.
UNIT	Device type for incoming files.
VOLUME	Disk pack for incoming files.
PREFIX	Default disk area.
TIMER	Determine whether Kermit-TSO should time out.

SET FILE BLKSIZE

Syntax: `SET FILE BLKSIZE` *number*

This sets the block size for incoming files to a *number* from 1 to 32760. In the case of fixed-format files, this number is just an upper bound; the actual block size is taken to be the largest multiple of the LRECL which does not exceed this limit. The default is 6233.

SET FILE LRECL

Syntax: `SET FILE LRECL` *number*

This sets the effective logical record length for incoming files to a *number* from 1 to 32756. This parameter is not used for files of undefined record format. Moreover, it is not exactly the same as the MVS/TSO LRECL, which is four more than the actual maximum data length for varying-length records. The default is 80.

SET FILE RECFM

Syntax: `SET FILE RECFM` *option*

This sets the record format to use for incoming files. Valid *option*s are "Fixed", "Varying" (the default), and "Undefined". This parameter is thus limited to a subset of the range of possibilities for the MVS/TSO RECFM. In Kermit-TSO, all incoming files of fixed or varying format are automatically blocked according to the current block size. Fixed-format records are padded or truncated, as needed, to the current LRECL.

SET FILE SPACE

Syntax: `SET FILE SPACE` *number*

This sets the track allocation unit for incoming files to a number from 1 to 32760. The default is 5. Since data sets are allowed as many as 15 extents, this default provides for files up to 75 tracks.

SET FILE UNIT

Syntax: `SET FILE UNIT` *type*

This sets the device type or group for incoming files. Valid *option*s are installation-dependent. The default is `SYSDA`, which is universally available, but not necessarily desirable, since many installations restrict TSO data sets to a particular set of disk volumes. In such cases, there is usually a unit name which refers only to those TSO volumes, and the global `INIT` file `'SYS1.KERMINI'` should set the file unit parameter to that name.

SET FILE VOLUME

Syntax: `SET FILE VOLUME` *name*

This sets the disk volume for incoming files. Valid *name*s are installation-dependent, but are, in any case, no more than six alphanumeric characters. The default is blank (none); in that case, the system chooses one of the available volumes of the specified UNIT type.

SET PREFIX

Syntax: `SET PREFIX` [*string*]

This subcommand is equivalent to the CWD subcommand (q.v.).

SET TIMER

Syntax: `SET TIMER ON` *or* `OFF`

This specifies whether Kermit-TSO is to maintain a timer for each packet it expects to read. The default is ON. If the timer is enabled, its duration is set by the SET SEND TIMEOUT subcommand initially and then set according to the request of the other Kermit.

The CWD Subcommand

Syntax: `CWD` [*string or PDSname*()]

The CWD (Change Working Directory) subcommand establishes a new default DSN prefix or turns prefixing off. This facility is similar to, but not quite the same as, the prefix defined in the User Profile. The *string*, if specified, must consist of one or more DSN qualifiers, and the first must already be an index in the disk catalog. Subsequent file transfers take place to and from the corresponding disk area whenever a fully qualified DSN (one enclosed in quotes) is not given. The initial prefix is the user's logon ID, i.e., the same as the default prefix in the User Profile. If no prefix is given in this subcommand, then prefixing is no longer performed. The user must be careful to remember the distinction between the prefix defined for Kermit and that for TSO. Pure Kermit subcommands (like `SEND` and `TAKE`) always use the former, but TSO commands (and the TSO-related subcommand `TYPE`) use the latter.

An alternative form of the CWD subcommand allows specifying the full (but unquoted) name of a PDS followed by paired parentheses. When such a "working directory" is in use, a *filespec* other than a fully qualified DSN is taken to be a member name within the PDS. For that reason, this form should be used cautiously, since the Kermit-TSO log *filespecs* (such as `KER.LOG` and `KER.REPLY`) would be treated the same way. In particular, it is advisable to turn on debug mode only when the Kermit prefix is a partially qualified DSN (once started, the log continues to the same data set regardless of what happens to the prefix).

The DIRECTORY Subcommand

Syntax: `DIRECTORY` [*filespec*]

The DIRECTORY subcommand uses the TSO LISTCAT command to display part of the data set catalog, i.e., all data sets whose names begin with the qualifiers in the Kermit prefix (if any) concatenated with the given *filespec* (if any).

The TSO Subcommand

Syntax: `TSO` *text of command*

Although Kermit-TSO does not have a full set of its own subcommands for managing TSO files, it provides those services through the operating system. You can issue any TSO command, e.g., to list, type, rename or delete files, send messages, and so on. The `TSO` subcommand under Kermit is synonymous with the `HOST` subcommand.

10.4. How to build an executable version of Kermit-TSO

Before attempting to build Kermit-TSO, look in the Kermit distribution under both IK0KER and IKTKER for an installation document, as well as "beware", help, and update files, and read them first. They will probably contain information that is more current than what you see here.

Kermit-TSO consists at present of a large assembly (KERMIT.ASM, containing the Kermit program) and a small one (DYNALC.ASM, containing a subroutine for allocating data sets). Although DYNALC is a single file in the Kermit distribution, the source for Kermit itself is in many pieces, some generic for Kermit-370 and some specific to TSO. All the necessary pieces are sequenced in columns 73-80 so that the numbers form a strictly increasing sequence when the pieces are correctly "pasted" together. It is important to preserve the original sequence numbers so that updates, if any, can be applied to the source.

To create a runnable version:

1. Combine the following ‘‘`ASM`’’ files from the Kermit distribution into a single file with `RECFM=F(B)` and `LRECL=80`: IK0DOC, IK0MAC, IKTMAC, IK0DEF, IK0MAI, IK0CMD, IK0COM, IKTUTL, and IK0PRO. The resulting file is the composite source for Kermit-TSO, called KERMIT.ASM. This source must retain the original sequence numbers in columns 73-80 (in other words, be sure not to resequence the source accidentally by using the editor!)
2. Copy or rename IKTDYN.ASM from the Kermit distribution to a file called DYNALC.ASM with `RECFM=F(B)` and `LRECL=80`.
3. Assemble the source file(s).
4. Create the executable load module `KERMIT` using the linkage editor. Kermit is designed to run as a command processor, and so it must be placed in SYS1.CMDLIB or in a PDS concatenated to SYS1.CMDLIB (for example, via the STEPLIB command).
5. If your site’s ASCII/EBCDIC translation table for TTY lines does not conform to the one listed in the appendix (which in turn conforms to the one given in the IBM System/370 Reference Summary), then enter the appropriate SET ATOE/ETOA/TATOE/TETOA subcommands into ’SYS1.KERMINI’. *NOTE:* If the ASCII/EBCDIC translation is not invertible, Kermit will not and cannot work.

In order to verify the operation of a new version of Kermit-TSO, you may run it under TEST using the CP parameter.

10.5. What's New

Below is a list of the more important features in Version 4.0 of Kermit-TSO added since the release of TSO/3708 in May 1987. Since Version 4.0 is the first release of Kermit-370 for TSO, some of the "new" features are actually new only to the Columbia TSO distribution.

1. Suppression of LINE and CHAR delete functions during protocol mode.
2. Advanced server functions and subcommands for talking to another Kermit running in server mode.
3. Long packet protocol.
4. `TYPE`, `ECHO`, `XTYPE`, and `XECHO` subcommands (the last two being Series/1 analogs of the first two.)
5. REMOTE KERMIT commands honored by TSO server, including `SET`, `SHOW`, `TAKE`, `TDUMP`, `STATUS`, `HOST`, `TSO`, `CWD`, `DIR`, and `TYPE`.
6. TEST mode for debugging.
7. Multi-column, two-level, selective SHOW display.
8. Send and acknowledge attribute packets.
9. Optionally append to, rather than replace, old data sets with duplicate names.
10. Automatic detection of terminal controller type (`TTY` or `SERIES1`).
11. SYNADAF message in cases of disk I/O error.

10.6. What's Missing

Work on Kermit-TSO will continue. Features that need to be improved or added include:

- Detect file properties from Attribute packets and allow overriding current parameter settings. Also implement file archiving.
- Add a `SET REPEAT` subcommand.
- Finish `SET LINE`, so that Kermit-TSO can be used as a local Kermit, connecting to a remote host over an alternate communication port. Add a `CONNECT` subcommand.
- Compute file size for outgoing A-packets and implement the SPACE subcommand.
- Reject files known (via A-packets) to be too big for available storage.
- Intercept *all* terminal output during protocol mode.
- Allow wildcard notation for PDS members.

11. VAX/VMS KERMIT

Authors: Robert C. McQueen, Nick Bush, Stevens Institute of Technology;
Jonathan Welch, Amherst College
Language: Bliss-32
Documentation:
R. McQueen, N. Bush, C. Gianone
Version: `3.3.117`
Date: June 1, 1988

VAX/VMS Kermit-32 Capabilities At a Glance:

Local operation:	Yes
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
`^X/^Y` interruption:	Yes
Filename collision avoidance:	Yes
Timeouts:	Yes
8th-bit prefixing:	Yes
Repeat character compression:	Yes
Alternate block check types:	Yes
Communication settings:	Yes
Transmit BREAK:	No
IBM mainframe communication:	Yes
Transaction logging:	Yes
Session logging (raw capture):	Yes
Debug logging:	Yes
Raw transmit:	Yes
Login scripts:	No
Act as server:	Yes
Talk to server:	Yes
Advanced commands for servers:	Yes
Local file management:	Yes
Initialization file:	Yes (`VMSKERMIT.INI`)
Long packets:	No
Sliding windows:	No
Attribute packets:	No

Kermit-32 is a program that implements the Kermit file transfer protocol for the Digital Equipment Corporation VAX series computers under the VAX/VMS operating system. It is written in BLISS-32 and MACRO-32, with sources for all BLISS modules also available as MACRO-32 sources. Kermit-32 should run on any VAX/VMS system from version 4.0 on (Version `3.1` of Kermit-32 is the last version that runs under pre-4.0 releases of VMS).

The first section of this chapter will describe the things you need to know about the VAX/VMS file system and how Kermit-32 uses it. The second section describes the special features of Kermit-32. The final section contains information of interest to those who need to install Kermit-32 on a system.

11.1. The VAX/VMS File System

The two main items of interest of the VAX/VMS file system (for the Kermit user) are the format of file specifications and the types of files and file data.

VAX/VMS File Specifications

VAX/VMS file specifications are of the form

```
NODE::DEVICE:[DIRECTORY]NAME.TYPE;VERSION
```

Under version 3.x of VMS, NAME may be up to 9 characters, TYPE may be up to 3 characters and each item in DIRECTORY may be up to 9 character long. Only alphanumeric characters may be used in DIRECTORY, NAME and TYPE.

Under version 4.0 (and later) of VMS, NAME, TYPE and each item in DIRECTORY may be up to 39 characters long, and may contain alphanumeric characters plus underscore.

VERSION is a decimal number indicating the version of the file (generation). DEVICE may be either a physical or logical device name. If it is a logical name, it may be up to 63 characters long and may contain alphanumeric characters plus dollar signs and underscores. NODE may be either a logical name which translates to a DECnet node name or a physical DECnet node name. In either case, access information can be included (see the DECnet-VMS User's guide for more information). The node name is not normally present, since most files are accessed on the same node where the user's job is running. The version number is not normally given (in fact, should not normally be given). When device and/or directory are not supplied, they default to the user's current default device and directory. Therefore, `NAME.TYPE` is normally all that is needed to specify a file on the user's default device and directory. This is also all the Kermit-32 will normally send as the name of a file being transferred.

The node field specifies the name (and access information) for the DECnet node where the file is located. Kermit-32 does not transmit the node field to the target system, but will attempt to honor a node field in an incoming file name.

The device field specifies a physical or "logical" device upon which the file is resident. The directory field indicates the area on the device, for instance the area belonging to the owner of the file. Kermit-32 does not normally transmit the device or directory fields to the target system, but will attempt to honor device or directory fields that may appear in incoming file names. It will not create new directories, however, so any directory must already exist.

The name is the primary identifier for the file. The type, also called the "extension", is an indicator which, by convention, tells what kind of file we have. For instance `FOO.FOR` is the source of a Fortran program named FOO; `FOO.OBJ` might be the relocatable object module produced by compiling `FOO.FOR`; `FOO.EXE` could an executable program produced by LINKing `FOO.REL`, and so forth.

VAX/VMS 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, including no characters at all; a "`%`" matches any single character. Here are some examples:

`*.FOR` All files of type `FOR` (all Fortran source files) in the default directory.

`FOO.*` Files of all types with name `FOO`.

`F*.*` All files whose names start with F.

`F*X*.*` All files whose names start with F and contain at least one X.

`%.*` All files whose names are exactly one character long.

`*.%%*` All files whose types are at least two characters long.

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.

Text Files and Binary Files

The file system used by VAX/VMS provides for a large number of attributes to be associated with each file. These attributes provide some indication of whether the file is a text file, or is some other type of non-text data. The two major attributes that affect Pro/Kermit are the record type and record attribute. The record type describes how logical records are stored in the file. Records may be of some fixed length (specified by another attribute), or variable length (specified within each record), or stream (implying no real record divisions). The record attributes describe how the breaks between records are to be treated. For example, a record attribute of implied carriage return means that any program reading the file with intentions of printing it out should add a carriage return/line feed sequence between each record. Other attributes include FORTRAN carriage control and print file format.

The "standard" method of storing text in a file under VAX/VMS is to store one line of text per record (variable length records), with a carriage return/line feed sequence implied by the end of the record (implied carriage return). This is the method Kermit-32 uses to store files it receives when using FILE TYPE TEXT. Note that there are other formats which are used to store text under VAX/VMS, however, the one used be Kermit-32 is the only one which is handled correctly by all known utility programs under VAX/VMS. Also, most programs which work with text files (the editor EDT, for example) place some limit on the length of the lines which can be handled. Typically this is 255. Kermit-32 can write files with up to 4095 characters on a line, which means a text file from another system may be transferred and stored correctly by Kermit-32, but may still be unusable by certain VAX/VMS programs.

Certain PC applications may create text files with lines even longer than Kermit-32's maximum. Typical examples are the ASCII export procedures of database, spreadsheet, and CAD packages. If you try to send such a file to Kermit-32, the transfer will fail with a message:

```
%KERMIT32-E-REC_TOO_BIG, Record too big for KERMIT's internal buffer
```

If this happens, you can SET FILE TYPE BINARY on the VAX before transferring the file to it. You should still be able to use the file as a text file, with the above proviso about record length.

There is no standard format for storing binary files. Basically, any record format with no record attributes are used for binary files. Since programs which work with binary files under VAX/VMS expect to see some particular format, more information is needed for transfer of binary files than for transfer of text files. The current version of Kermit-32 is not capable of transferring all types of binary files which were created on a VAX/VMS system to another system and retrieving them intact, nor is it capable of transferring all of types binary files created on a VAX/VMS system to another VAX/VMS, P/OS, or RSX-11M/M+ system intact. However, certain formats of binary files can be transferred, and binary files from some other systems may be transferred to a VAX and recovered intact.

Binary files which are created on a VAX (or other Files-11 systems) with fixed 512 byte records (a fairly common format) can be transferred using Kermit-32. The only required action is to set the file type to "fixed" in the receiving Kermit-32.

Using two programs supplied with Kermit-32, it is possible to transfer almost any type of sequential file between VAXes, or between a VAX and a P/OS or RSX-11M/M+ system. These two programs (VMSHEX and VMSDEH) will convert the binary files to text (using a variation on Intel hex format). The resulting text file can be transferred like any other, and finally "dehexified" reproducing the original file, with the major attributes intact. Unfortunately, the text files tend to be about twice the size of the original binary files, so the transfers take a bit longer than regular text files. On the plus side, the text versions of the files can be transferred to any system with a Kermit and still retrieved intact. They can also be transferred over 7-bit data paths without any problems. The bootstrap procedure (described below), makes use of hexified versions of the binary file which makes up Kermit-32.

Using the VAX to Archive Microcomputer Files

You can use Kermit to send textual files from a microcomputer or any 8-bit system to a VAX/VMS system with no special provisions, since Kermit-32 stores incoming files as text files (variable length records with implied carriage returns) unless it is explicitly told otherwise. But Kermit-32 has no automatic way of distinguishing an incoming binary file from an incoming text file. It turns out that because of the method used by Kermit-32 for storing text files, a binary file can be stored like a text file so long as it does not contain a string of more than 4095 characters between carriage return, line feed sequences, and ends with a carriage return line feed. Since most binary files do not have these characteristics, you must inform Kermit-32 that a file it is about to receive is to be stored as a binary file. This is done using the SET FILE TYPE BINARY command. This instructs Kermit-32 to store the data it receives in the file without checking for carriage return, line feed sequences. The file it creates will be variable record length, with no record attributes. Each record will contain 510 bytes of data, except the last, which will contain whatever amount is left before the end of the file. This allows Kermit-32 to correctly return exactly the data it was sent when the file is returned to the original system.

Note that because of the need to use a different file type for binary files, it is not possible to use a "wildcard send" command to send a mixture of text and binary files to a VAX system unless the text files are not intended for use on the VAX; rather, you must send all text files with Kermit-32's file type set to text, and all binary files with the file type set to binary.

Once you get the foreign file into the VAX system, stored with the correct file type, you need take no special measures to send it back to its system of origin. This is because Kermit-32 honors the record type and attributes of the file as it is stored on the VAX. In fact, SET FILE TYPE BINARY or TEXT only affects how Kermit-32 receives files - it does not affect how Kermit-32 transmits files.

Files Kermit-32 Cannot Handle

The Kermit protocol can only accommodate transfer of *sequential* files, files which are a linear sequence of bytes (or words).

Some files on a VAX/VMS system are not sequential, and cannot be successfully sent or received by Kermit-32. These are mainly indexed data files, but can also include other files which require more than just the data in the file to be completely reconstructed. External control information and file attributes are not transmitted. However, *any* VMS file can be transferred with Kermit if it has been "hexified" with VMSHEX.

11.2. Program Operation

Kermit-32's prompt is normally "`Kermit-32>`". If a foreign command is defined to run Kermit-32 (eg. `KERMIT := $KERMIT`), Kermit-32 will accept a single command on the command line, like this:

```
$
$ Kermit send foo.bar
```

the file is sent

```
$
$ mcr kermit send foo.bar
```

the file is sent

```
$
```

You can run the program interactively to issue several commands, like this:

```
$
$ run sys$system:kermit
VMS Kermit-32 version 3.3.117
```

```
Default terminal for transfers is: _TTA1:
Kermit-32>send foo.*
```
files are sent
```
Kermit-32>statistics
```
performance statistics are printed
```
Kermit-32>receive
```
files are received
```
Kermit-32>exit
$
```

Upon initial startup, Kermit-32 executes commands from its initialization file, `VMSKERMIT.INI`, if any, in your directory.

Command keywords may be abbreviated to their shortest prefix that sets them apart from any other keyword valid in that field.

Kermit-32 provides most of the commands possible for an "ideal" Kermit program, as described in the main part of the *Kermit User Guide*. The following sections will concentrate on system-dependent aspects of Kermit-32.

11.3. Conditioning Your Job for Kermit

Kermit-32 does as much as it can to condition your line for file transfer. It saves all your terminal settings, and restores them after use. However, there are some sources of interference over which Kermit-32 has no control. In particular, messages issued by other processes in your job could become mingled with Kermit packets and slow things down or stop them entirely. This is a fairly rare occurence and can be easily avoided by not running any other process which wishes to perform I/O to your terminal while you are running Kermit-32.

Normally, when Kermit-32 is run, it assumes you wish to use it in remote mode and perform file transfers over the terminal line which controls your job. This can be overridden, however, by defining a logical name which equates to some other terminal line in the system. The default terminal line to be used for file transfers is determined by the first of the following logical names which translates to a terminal line which is available for use by your process: KER$COMM, SYS$INPUT, SYS$OUTPUT, and SYS$COMMAND. If none of these logical names translate to an available terminal line, there is no default terminal line and a SET LINE command must be used before any transfer command is performed. Note that this is the typical case in a batch job.

Kermit-32 will also default the type of parity to be used on the communication line to that which is set on its default terminal line when it is started. This means that if all communication at a site is normally done using even parity (for example), Kermit-32 will also use even parity.

There are two things to keep in mind when using Kermit-32 in local mode (where the file transfers are done over a different terminal line from where commands are typed):

- Under VAX/VMS, every terminal line has an owner UIC and protection code associated with it. This UIC and protection is used to determine who can allocate (and therefore use) the terminal line when they are not logged in on that line. Therefore, in order for Kermit-32 to be able to perform file transfers over a terminal line other than the one on which you are logged in, the field of the protection code for the terminal which applies to your job (based on your UIC and the owner UIC of the terminal) must allow your job access to the terminal. You may need to request your system manager to change the protection for a terminal line to allow you to use it with Kermit-32 in local mode. See the section on Installation for details.
- Terminal lines which have been declared as modem control lines will have the phone "hung up"

whenever the terminal line becomes free (deallocated). This means that if you do not use the DCL ALLOCATE command to allocate the terminal line to your job before entering Kermit-32, exiting Kermit-32 will cause the terminal line to "hang up" the modem. If you do wish to get to DCL after having used Kermit-32 to connect a modem control line which you do not have allocated, you can use the PUSH command to spawn a subprocess running DCL.

11.4. Kermit-32 Commands

This section describes the Kermit-32 commands -- in detail where they differ from the "ideal" Kermit, briefly where they coincide. Kermit-32 has the following commands:

@	synonym for "take".
BYE	to remote server.
CONNECT	as terminal to remote system.
EXIT	from Kermit-32.
FINISH	Shut down remote server.
GET	remote files from server.
HELP	with Kermit-32.
LOCAL	prefix for local file management commands.
LOG	remote terminal session.
LOGOUT	remote server.
PUSH	to DCL command level.
QUIT	from Kermit-32.
RECEIVE	files from remote Kermit.
REMOTE	prefix for remote file management commands.
SEND	files to remote Kermit.
SERVER	mode of remote operation.
SET	various parameters.
SHOW	various parameters.
STATUS	about most recent file transfer.
TRANSMIT	Transmit (upload) a file with no error checking.
TAKE	Kermit-32 commands from a file.

11.4.1. Commands for File Transfer

Kermit-32 provides the standard SEND, RECEIVE, and GET commands for transferring files using the Kermit protocol.

The SEND Command

Syntax:

Sending a file or files:

`SEND` *filespec*

The SEND command causes a file or file group to be sent from the VAX to the other system. If *filespec* contains wildcard characters then all matching files will be sent, in alphabetical order (according to the ASCII collating sequence) by name. If *filespec* does not contain any wildcard characters, then the single file specified by *filespec* will be sent.

SEND Command General Operation:

Files will be sent with at least their VAX/VMS file name and type (for instance `FOO.BAR`). If a SET FILE NAMING FULL command has been given, Kermit-32 will also send the device name, directory name and version number (for instance `USER$DISK:[JOE]FOO.BAR;25`). If a SET FILE NAMING UNTRANSLATED command has been given, Kermit-32 will send the file name, type and version number (for instance `FOO.BAR;25`). If a SET FILE NAMING NORMAL_FORM command has been given (this is the initial default), Kermit-32 will only send the file name and type.

Each file will be sent according to the record type and attributes recorded in its file descriptor. Kermit-32 attempts to translate all formats of text file (including those with FORTRAN or print carriage control) to a format usable on any system. Note that there is no need to set the FILE TYPE parameter for sending files, since Kermit-32 always uses the information from the file descriptor to determine how to send the file.

If communication line parity is being used (see SET PARITY), Kermit-32 will request that the other Kermit accept a special kind of prefix notation for binary files. This is an advanced feature, and not all Kermits have it; if the other Kermit does not agree to use this feature, binary files cannot be sent correctly. This includes executable programs (like `.EXE` files, CP/M `.COM` files), relocatable object modules (`.OBJ` files), as well as any text file containing characters with the eighth bit on.

Kermit-32 will also ask the other Kermit whether it can handle a special prefix encoding for repeated characters. If it can, then files with long strings of repeated characters will be transmitted very efficiently. Columnar data, highly indented text, and binary files are the major beneficiaries of this technique.

If you're running Kermit-32 locally, for instance dialing out from a VAX to another system using an autodialer, you should have already run Kermit on the remote system and issued either a RECEIVE or a SERVER command. Once you give Kermit-32 the SEND command, the name of each file will be displayed on your screen as the transfer begins. If the file is successfully transferred, you will see `"[OK]"`, otherwise there will be an error message.

During local operation, you can type Control-A at any point during the transfer to get a brief status report. You may also type Control-X or Control-Z to interrupt the current file or file group.

The RECEIVE Command

Syntax: `RECEIVE [`*filespec*`]`

The RECEIVE command tells Kermit-32 to receive a file or file group from the other system. If only one file is being received, you may include the optional *filespec* as the name to store the incoming file under; otherwise, the name is taken from the incoming file header. If the name in the header is not a legal VAX/VMS file name, Kermit-32 will normally replace the illegal characters with "X" (see SET FILE NAMING NORMAL_FORM).

If an incoming file has the same name as an existing file, Kermit-32 just creates a new version of the same name and type, for instance `FOO.BAR;3, FOO.BAR;4`.

Incoming files will all be stored with the prevailing file type, ASCII by default, which is appropriate for text files. If you are asking Kermit-32 to receive binary files from a microcomputer or other 8-bit system, you must first type SET FILE TYPE BINARY. Otherwise, an error may occur when receiving the file, or a carriage return, line feed will be added to the end of the file and the file will be useless when sent back to the system of origin.

If parity is being used on the communications line, then 8th-bit prefixing will be requested. If the other side cannot do this, binary files cannot be transferred correctly.

If an incoming file does not arrive in its entirety, Kermit-32 will normally discard it; it will not appear in your directory. You may change this behavior by using the command SET INCOMPLETE KEEP, which will cause as much of the file as arrived to be saved in your directory.

If you are running Kermit-32 locally, you should already have issued a SEND command[1] to the remote Kermit, and then escaped back to Kermit-32. As files arrive, their names will be displayed on your screen. You can type Control-A during the transfer for a brief status report.

If a file arrives that you don't really want, you can attempt to cancel it by typing Control-X; this sends a cancellation request to the remote Kermit. If the remote Kermit understands this request (not all implementations of Kermit support this feature), it will comply; otherwise it will continue to send. If a file group is being sent, you can request the entire group be cancelled by typing Control-Z.

The GET Command

Syntax: `GET [`*remote-filespec*`]`

The GET command requests a remote Kermit server to send the file or file group specified by *remote-filespec*. This command can be used only when Kermit-32 is local, with a Kermit server on the other end of the line specified by SET LINE. This means that you must have CONNECTed to the other system, logged in, run Kermit there, issued the SERVER command, and escaped back to the VAX.

The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. Any leading spaces before the remote filespec are stripped, and lower case characters are raised to upper case.

As files arrive, their names will be displayed on your screen. As in the RECEIVE command, you may type Control-A to get a brief status report, ^X to request that the current incoming file be cancelled, ^Z to request that the entire incoming batch be cancelled.

If the remote Kermit is not capable of server functions, then you will probably get an error message back from it like "Illegal packet type". In this case, you must connect to the other Kermit, give a SEND command, escape back, and give a RECEIVE command.

The STATUS Command

Give statistics about the most recent file transfer.

The PUSH Command

Syntax: `TAKE`

Invoke an inferior DCL command processor, to which you may issue any DCL commands. Type LOGOUT to return to Kermit-32.

[1] *not* SERVER -- use the GET command to receive files from a Kermit server.

The TAKE Command

Syntax: `TAKE` *file-spec* `[ /DISPLAY ]`

Where 'file-spec' is any normal VAX/VMS file specification. If file-spec does not specify a file-type Kermit-32 will supply a default of `.COM`. The /DISPLAY option causes the commands read from the file to be displayed on the user's terminal.

The TAKE command tells Kermit-32 to execute commands from the specified file. You may also use the VAX/VMS notation "@" instead of Take to specify a command file.

The file `VMSKERMIT.INI` is automatically taken upon program startup.

11.4.2. Server Operation

The SERVER Command

The SERVER command puts a remote Kermit-32 in "server mode", so that it receives all further commands in packets from the local Kermit. The Kermit-32 server is capable (as of this writing) of executing the following remote server commands: SEND, GET, FINISH, BYE, REMOTE DIRECTORY, REMOTE CWD, REMOTE SPACE, REMOTE DELETE, REMOTE TYPE, REMOTE HELP, REMOTE COPY, REMOTE RENAME, REMOTE SEND_MESSAGE, REMOTE WHO, and REMOTE HOST.

Any nonstandard parameters should be selected with SET commands before putting Kermit-32 into server mode, in particular the file type. The Kermit-32 server can send all files in the correct manner automatically. However, if you need to ask Kermit-32 to receive binary files you must issue the SET FILE TYPE BINARY command before putting it into server mode, and then you must only send binary files. You cannot send a mixture of text files and 8-bit binary files to a Kermit-32 server unless the files are not for use on the VAX.

Commands for Servers

When running in local mode, Kermit-32 allows you to give a wide range of commands to a remote Kermit server, with no guarantee the that the remote server can process them, since they are all optional features of the protocol. Commands for servers include the standard `SEND`, `GET`, `BYE`, `LOGOUT` and `FINISH` commands, as well as the `REMOTE` command.

Syntax: `REMOTE` *command*

Send the specified command to the remote server. If the server does not understand the command (all of these commands are optional features of the Kermit protocol), it will reply with a message like "Unknown Kermit server command". If does understand, it will send the results back, and they will be displayed on the screen. The REMOTE commands are:

COPY *filespec*
: Copy file. The server is asked to make a copy of the specified file. Kermit-32 will prompt for the new file name on a separate line. Both filespecs must be in the correct format for the remote system. Kermit-32 does not parse or validate the file specifications. Any leading spaces will be stripped and lower case characters converted to upper case. Note that this command simply provides for copying a file within the server's system - it does not cause a file to be transferred.

CWD [*directory*]
: Change Working Directory. If no directory name is provided, the server will change to the default or home directory. Otherwise, you will be prompted for a password, and the server will attempt to change to the specified directory. The password is entered on a separate line, and does not echo as you type it. If access is not granted, the server will provide a message to that effect. Note that while not all server Kermits require (or accept) a password to change the

working directory, Kermit-32 will always ask for one when a directory name is provided.

DELETE *filespec* Delete the specified file or files. The names of the files that are deleted will appear on your screen.

DIRECTORY [*filespec*]
The names of the files that match the given file specification will be displayed on your screen, perhaps along with size and date information for each file. If no file specification is given, all files from the current directory will be listed.

DISK_USAGE [*directory*]
Display information about disk usage in the given directory (or by the given user). If no directory is provided, disk usage information is provided for the current working directory (or user). This is the same as the REMOTE SPACE command.

EXIT Requests the server to leave Kermit, allowing the terminal to be used for normal commands.

FINISH Requests the server to return to the Kermit prompt, allowing statistics to be obtained about the transfers.

HELP [*topic*] Provide information about the given topic. If no topic is given, provide a list of the functions that are available from the server. Some servers may ignore the topic and always display the same information.

HOST [*command*] Pass the given command to the server's host command processor, and display the resulting output on your screen.

LOGIN *user-id* Supply information to the server Kermit to indicate what user-id, account and password are to be used. The server Kermit may use this to validate the user's access to the system as well as for billing purposes. It may also use this information to provide the user with access to files on its system.

LOGOUT Request the server to exit Kermit and logout its job (or process). This command is identical to the LOGOUT command.

RENAME *filespec* Change the name on the specified file (or files). Kermit-32 will prompt for the new file specification on the next line. Both file specifications must be valid for the server's system.

SEND_MESSAGE *destination-address*
Request the server to send a single line message to the specified destination address (which might be a user-id, terminal designator, or some other item, depending upon the server Kermit). Kermit-32 will prompt for the single line message on the next line.

SPACE [*directory*] Display information about disk usage in the given directory (or by the given user). If no directory is provided, disk usage information is provided for the current working directory (or user). This is the same as the REMOTE DISK_USAGE command.

STATUS Display information about the status of the server.

TYPE *filespec* Display the contents of the specified file on your screen.

WHO [*user-id*] Display information about the given user. If no user-id is given, display information about the currently active users. Kermit-32 will prompt for options for selecting what information to display and/or formatting parameters. The format of both the user-id and the options are dependent upon the server Kermit.

11.4.3. Commands for Local File Management

Syntax: `LOCAL [`*command*`]`

Execute the specified command on the local system -- on the VAX/VMS system where Kermit-32 is running. These commands provide some local file management capability without having to leave the Kermit-32 program. These commands are very similar to the REMOTE commands in function and syntax. They are all executed locally, and are available when Kermit-32 is either local or remote. The arguments to these commands are the same as the arguments expected from the user Kermit when Kermit-32 is processing a command in server mode.

COPY *filespec* Make a copy of the given file (or files). Kermit-32 will prompt for the new file specification. The command is actually performed by using the DCL COPY command (COPY/LOG *old-file new-file*), and any options which are valid on the DCL COPY command may be included.

CWD [*directory*] Change working directory, or, in VAX/VMS terminology, change the default device/directory. This command takes the same arguments as the DCL SET DEFAULT command (i.e., a device and directory, only a directory, or only a device). If no argument is given, the default device and directory are reset to that in effect when Kermit-32 was run. The new default device and directory will be typed out.

DELETE *filespec* Delete the specified file or files. This command is performed by using the DCL DELETE command (DELETE/LOG *filespec*). Therefore, any options which are valid on the DCL DELETE command may be included.

DIRECTORY [*filespec*]
Provide a directory listing of the specified files. This command is performed by using the DCL DIRECTORY command (DIRECTORY *filespec*), so any options valid for the DCL DIRECTORY command may be included.

DISK_USAGE [*uic*]
Display disk usage information for the given UIC. If no UIC is given, display disk usage information for the process UIC. This command is performed by using the DCL SHOW QUOTA command (SHOW QUOTA or SHOW QUOTA/USER=*uic*).

HELP Display the help message describing the server commands which are available.

HOST *DCL command*
Perform the given DCL command. The command should not perform any action which will require more input. Any output resulting from the command will be typed on the terminal.

RENAME *filespec* Change the name of the specified file. Kermit-32 will prompt for the new name on the next line. This command is performed by using the DCL RENAME command (RENAME/LOG *old-file new-file*), so any options which are valid on the DCL RENAME command may be included.

SEND_MESSAGE *terminal-name*
Send a single line message to the given terminal. Kermit-32 will prompt for the message on the next line. Since this command is performed using the DCL REPLY command

`REPLY/TERMINAL=`*terminal-name* `"`*message*`"`

OPER privileges are needed to perform it.

TYPE *filespec* Display the contents of the specified file or files at your terminal. Each file will be preceded by its name in angle brackets.

11.4.4. The CONNECT Command

Syntax: `CONNECT [`*terminal-name*`]`

Establish a terminal connection to the system connected to the terminal line specified here or in the most recent SET LINE command, using full duplex echoing and no parity unless otherwise specified in previous SET commands. Get back to Kermit-32 by typing the escape character followed by the letter C. The escape character is Control-Close-Square-Bracket (`^]`) by default. When you type the escape character, several single-character commands are possible:

`C` Close the connection and return to Kermit-32.
`Q` If a session log is active, temporarily Quit logging.
`R` Resume logging to the session log.
`S` Show status of the connection.
`0` Send a null character.
`?` List all the possible single-character arguments.
`^]` (or whatever you have set the escape character to be):
Typing the escape character twice sends one copy of it to the connected host.

You can use the SET ESCAPE command to define a different escape character, and SET PARITY, and SET LOCAL_ECHO to change those communication-line-oriented parameters. Type the SHOW LINE command for information about your current communication settings.

Kermit-32 does not have any special autodialer interface. It assumes that the connection has already been made and the line assigned.

11.4.5. The SET and SHOW Commands

The SET Command

Syntax: `SET` *parameter* `[`*option* `[`*value*`]]`

Establish or modify various parameters for file transfer or terminal connection. You can examine their values with the SHOW command. The following parameters may be SET:

BLOCK_CHECK	Packet transmission error detection method
DEBUGGING	Record or display state transitions or packets
DELAY	How long to wait before starting to send
ESCAPE	Character for terminal connection
FILE	For setting file parameters like file type
HANDSHAKE	For establishing half duplex line turnaround handshake
IBM_MODE	For communicating with an IBM mainframe
INCOMPLETE_FILE	What to do with an incomplete file
LINE	Terminal line to use for file transfer or CONNECT
LOCAL_ECHO	For terminal connection, ON or OFF
MESSAGE	The type of typeout to be done during transfers
PARITY	Character parity to use
PROMPT	Change the program's command prompt
RECEIVE	Various parameters for receiving files
REPEAT_QUOTE	Character to use for repeat compression
RETRY	How many times to retry a packet before giving up
SEND	Various parameters for sending files
TRANSMIT	Control TRANSMIT command echo and delay

Those SET commands which differ from the "ideal" Kermit are now described in detail.

SET DEBUGGING

Syntax: `SET DEBUGGING` *options*

Record the packet traffic, either on your terminal or in a file. Some reasons for doing this would be to debug a version of Kermit that you are working on, to record a transaction in which an error occurred for evidence when reporting bugs, or simply to vary the display you get when running Kermit-32 in local mode. Options are:

ON Display each incoming and outgoing packet (lengthy).

OFF Don't display or record debugging information (this is the normal mode). If debugging was in effect, turn it off.

The debugging information is recorded in the file specified by the most recent LOG DEBUGGING command.

SET ESCAPE

`SET ESCAPE` *octal-number*

Specify the control character you want to use to "escape" from remote connections back to Kermit-32. The default is 35 (Control-]). The number is the octal value of the ASCII control character, 1 to 37 (or 177), for instance 2 is Control-B. After you type the escape character, you must follow it by a one of the single-character "arguments" described under the CONNECT command, above.

SET FILE

Syntax: `SET FILE` *parameter keyword*

Establish file-related parameters:

TYPE *keyword*

Type of file for VAX/VMS file output. The choices are ASCII, BINARY, or FIXED.

ASCII Store the file as a standard VAX/VMS text file. Any file received is stored as variable length records with carriage return, line feed sequences implied between records. This is the format preferred by most utility programs under VAX/VMS. An error will occur if any line is more than 4096 characters long. Note that lines are only terminated by carriage return, line feed sequences. A carriage return that is not followed by a line feed or a line feed that is not preceded by a carriage return is not considered the end of a line, and is included within the body of a record.

BINARY Store the file as a binary file. Any file received is stored as variable length records with no record attributes. Kermit-32 actually will write 510 bytes in each record except the last. This makes each record take up one disk block (510 data bytes plus two bytes of record length). The last record is written containing only as much data is left to the end of the file. Any file which is just a stream of bytes can be stored as a BINARY file, and recovered intact later. This is the preferred file type for use in archiving files.

FIXED Store the file as a fixed length binary file. Any file received is stored as fixed length 512 byte records with no record attributes. This is the format used for binary files such as VAX/VMS "EXE" files and RSX-11M/M+ "TSK" files. Since even the last record of the file is written with 512 bytes (even if it is not filled), this format does not necessarily maintain the correct length of a file. It should normally only be used for files which are coming from a VAX/VMS system which are currently stored in fixed 512 byte records.

NAMING *keyword*

Determine the form of names to be sent with outgoing files and determine the translation performed on incoming file names. The choices are FULL, NORMAL_FORM and UNTRANSLATED.

FULL Kermit-32 will send full file names (including device, directory, file name, file type and version number). When receiving a file, Kermit-32 will perform no translation of the file name (which must therefore be a legal VAX/VMS file specification).

NORMAL_FORM

Kermit-32 will send only the file name and file type. When receiving a file, Kermit-32 will convert the file specification received to contain only uppercase letters, digits, and at most one period. Any other characters will be translated to "X". There will be at most 9 characters before the period (if any), and at most 3 characters afterwards. This forces the file name to be a valid VAX/VMS 3.x file specification. This is the default.

UNTRANSLATED

Kermit-32 will send only the file name and file type. When receiving a file, Kermit-32 will not perform any conversions on the file specification, which therefore must be a legal VAX/VMS file specification. If you want to receive files with long names, use this option. To transfer files with VAX/VMS long names between two VMS 4.0-or-later systems, use this option on both sides.

SET HANDSHAKE

Syntax: `SET HANDSHAKE` *o*

Sets the half duplex line turnaround handshake character to the ASCII character whose octal value is *o*. Normally required for communication with half duplex systems like IBM mainframes.

SET IBM_MODE

Syntax: `SET IBM_MODE ON` *or* `OFF`

When IBM_MODE is set to ON, Kermit-32 will override the parity and local echo settings and use odd parity, local echo on, and also enable a handshake character of XON (control-Q, ASCII 021 octal). This feature allows Kermit-32 to talk with certain systems (notably some IBM mainframes), which require waiting for a XON before sending data.

The various features selected by this command can be overridden subsequently by SET PARITY, SET LOCAL_ECHO, and SET HANDSHAKE commands.

SET LINE

Syntax: `SET LINE [`*terminal-name*`]`

Specify the terminal name to use for file transfer or CONNECT; the *terminal-name* can be up to 16 characters long. If you issue this command using other than your job's controlling terminal, you will be running Kermit-32 *locally*, and you must log in to the remote system and run Kermit on that side in order to transfer a file. If you don't issue this command, Kermit-32 determines whether it is to run locally or *remotely* based on the default terminal line found when Kermit-32 is started. Kermit-32 uses a list of logical names to determine which terminal should be the default terminal line. The first of these names which translates to a terminal which is available (i.e., not allocated by some other process) is used. The logical names Kermit-32 tries are KER$COMM, SYS$INPUT, SYS$$OUTPUT, and SYS$COMMAND. If none of these translate to an available terminal, Kermit-32 is running *detached*, and a terminal must be specified by the SET LINE command before any actions can be performed. If a terminal is found, Kermit-32 is running locally if this is a terminal other than the one controlling the job (i.e., different from SYS$COMMAND), otherwise Kermit-32 is running remotely. You can also select the line directly in the CONNECT command; the command

```
CONNECT TTA0
```

is equivalent to

```
SET LINE TTA0
CONNECT
```

If you type SET LINE with no argument, you will deassign any previous assigned line and revert to remote mode.

SET SERVER_TIMEOUT

Syntax: SET SERVER_TIMEOUT *number*

This specifies the number of seconds between timeouts during server command wait, 0 specifies that no timeouts should occur during server command wait. When a Kermit server times out, it sends a NAK packet. Some systems cannot clear piled-up NAKs from their input buffers; if you're using such a system to communicate with a Kermit-32 server, and you expect to be leaving the server idle for long periods of time, you should use this command to turn off server command-wait timeouts.

SET TRANSMIT

Syntax: SET TRANSMIT DELAY *integer*, SET TRANSMIT ECHO ON/OFF

It is possible to set a few parameters associated with the raw TRANSMIT command that vary both what the user sees on the screen as well as the speed of the transmit.

SET TRANSMIT DELAY

This parameter is the amount of time to delay after each carriage return is transmitted. Valid delay values range between 0 (the default) and 9 tenths of a second. The format of the command is: SET TRANSMIT DELAY *d* Where *d* is a single decimal digit representing tenths of a second.

Some remote hosts may not be able to receive the characters as fast as Kermit-32 can send them. The TRANSMIT DELAY can be used to slow up the transfer by adding a slight delay after each line is sent.

The transfer also runs slower if the transmit echo is on, and the remote system is echoing the characters as it receives them. If the transmit delay is set to 9 tenths of a second, the remote system is echoing characters, the transmit echo is on, and the remote system still cannot keep up, then the connection should be made at a slower baud rate.

Conversely, the file transfer speed can be increased by: setting the delay to 0 and the echo off, stopping the remote system from echoing the characters it receives, and connecting at higher baud rates.

SET TRANSMIT ECHO

This command controls what the user sees on the screen during the file transfer. The format of the command is SET TRANSMIT ECHO ON or OFF.

By default, the transmit echo is left off and the user sees the number of each line after it has been transmitted. With transmit echo on, the user sees whatever the remote system would normally echo back to him while he is typing in a file. Note that turning the echo on typically slows the file transfer down.

The SHOW Command

Syntax: `SHOW` [*option*]

The SHOW command displays various information:

ALL
: All parameters.

BLOCK_CHECK_TYPE
: The block check type being requested.

COMMUNICATIONS
: Parameters affecting the terminal line being used for communication.

DEBUGGING
: Debugging mode in effect, if any.

DELAY
: The number of seconds Kermit-32 will delay before starting a SEND or RECEIVE command when in remote mode.

ESCAPE
: The current escape character for the CONNECT processing.

FILE_PARAMETERS
: File type, file naming, and incomplete file disposition.

INCOMPLETE_FILE_DISPOSITION
: The action to take when a transfer is aborted.

LINE
: Terminal line in use.

LOCAL_ECHO	Whether characters should be echoed locally when CONNECTed.
PACKET	For incoming and outbound packets.
PARITY	The parity type in use.
RECEIVE	For inbound packets.
RETRY	The number of retries to be done on bad packets.
SEND	For outbound packets.
TRANSMIT	Parameters for TRANSMIT command.
VERSION	The program version number of Kermit-32.

11.4.6. Program Management Commands

The HELP Command

Syntax: `HELP` [*topic* {*subtopic*}]

Typing HELP alone prints a brief summary of Kermit-20 and its commands. You can also type

`HELP` *command*

for any Kermit-20 command, e.g. "help send" or "help set parity" to get more detailed information about a specific command.

The EXIT and QUIT Commands

Syntax: `EXIT`

Exit from Kermit-32. You can also exit from the Kermit-32 when it is waiting for a command by typing a control-Z. When Kermit-32 is running remotely, two control-Y's will abort the transfer, bringing Kermit-32 back to command mode. The two control-Y's must be typed together, as if a timeout occurs between them the first is ignored. When Kermit-32 is running locally, two control-Y's will stop Kermit-32 and return you to DCL. You will be able to CONTINUE if you do not perform any command which runs a program. However, after continuing, control-A, control-X and control-Z will no longer be accepted as commands.

`QUIT` is a synonym for EXIT.

The LOG Command

Syntax: `LOG` [*option* [*filespec*]]

Log the specified option to the specified file:

SESSION	During CONNECT log all characters that appear on the screen to the specified file. During CONNECT, the session log can be temporarily turned off during the remote session by typing the escape character followed by Q (for Quit logging), and turned on again by typing the escape character followed by R (for Resume logging).
TRANSACTIONS	During file transfer, log the progress of each file. Transaction logging is recommended for long or unattended file transfers, so that you don't have to watch the screen. The log may be inspected after the transfer is complete to see what files were transferred and what errors may have occurred.
DEBUGGING	Log debugging info to the specified file. If no SET DEBUGGING command was previously

issued, the file will be opened and no information written. If DEBUGGING is turned on (either via the SET DEBUGGING command or by typed control-D during a local transfer), the packet debugging information will be written to the file. Packet format is described in the *Kermit Protocol Manual*.

Any log files are closed when you EXIT or QUIT from Kermit. You may explicitly close a log file and terminate logging by using the LOG command without a file specification.

The STATUS Command

Syntax: `STATUS`

The current status of Kermit-32 will be displayed. This includes the number of characters that have been sent and received from the remote Kermit. Also included is an estimate of the effective baud rate of the transfer. This number is not intended to be exact, but only an indication of what range of throughput has been provided.

11.5. Raw Upload and Download

The TRANSMIT Command

Syntax: `TRANSMIT` *file-spec*

The TRANSMIT command allows you to upload files "raw" to systems that don't have a Kermit program available. Note that there is no error checking or packets involved in this method of file transfer.

This command does a raw transmit of an ASCII file, one character at a time, with carriage returns (no line-feeds) at the end of each line. It is used with Kermit-32 in local mode. The user must first prepare the remote host to receive the file by starting an edit session in input mode. Then the user can escape back to Kermit-32 and issue the TRANSMIT command. After the transmit is finished, the user then CONNECTs back to the remote host again and ends the edit session.

During a file transmit, the following control characters can be used to affect the transfer in progress:

CTRL-C	Abort the transmit
CTRL-X	Abort the file currently being transmitted
CTRL-Z	Abort the file group currently being transmitted

See SET TRANSMIT for information about controlling echo and delays.

The LOG SESSION Command

Syntax: `LOG SESSION` *file-spec*

"Raw Download" is the term commonly used to describe the capture of a remote file on the local system, without any kind of error detection or correction. This allows you to obtain files from remote systems that do not have Kermit, but with the risk of loss or corruption of data.

Kermit-32 provides raw downloading via the LOG SESSION command during CONNECT to a remote system. The session log is described above. To use session logging to capture a file:

1. Run Kermit on the VAX/VMS system.

2. SET LINE to the terminal line through which you will be connected to the remote system.

3. Perform any required SET commands to condition Kermit for communication with the remote system.

4. CONNECT to the remote system and log in.

5. Condition your job on the remote system not to pause at the end of a screenful of text, and give whatever commands may be necessary to achieve a "clean" terminal listing -- for instance, disable messages from the system or other users.

6. Type the appropriate command to have the desired file displayed at the terminal, *but do not type the terminating carriage return*. On most systems, the command would be "type", on Unix it's "cat".

7. Escape back to Kermit-32 and give the LOG SESSION command with the file specification where you wish to store the data.

8. CONNECT back to the remote system and type a carriage return. The file will be displayed on your screen and recorded in the session log file.

9. Escape back to Kermit-32 and give the LOG SESSION command without a file specification.

The file you specified will contain everything that was typed on your screen. You will probably find that some editing necessary to remove extraneous prompts, messages, padding characters, or terminal escape sequences, or to fill in lost or garbled characters.

Use the TRANSMIT command for raw uploading.

11.6. Installation of Kermit-32

VMS Kermit-32 comes in 3 forms: hex, Macro source, and Bliss source. Each can be used as the basis for installation.

Before beginning, make a special directory for VMS Kermit and read the files `VMS*.*` from the Kermit distribution tape into this directory. Columbia's Ker- mit tapes are written with blocksize 8192, which is 4 times larger than the default tape blocksize for VMS. You should mount these tapes on the VMS system with the following command:

```
MOUNT/BLOCK=8192/DENSITY=1600 MTA0: KERMIT
```

(or substitute some other tape drive name for `MTA0:`) Do not use the `/FOREIGN` switch. Once the tape is mounted, you can use normal VMS COPY commands to copy the files from the tape. For instance, if you have defined your Kermit directory to have logical name KER:, you can use the following command to copy the VMS Kermit files into this directory:

```
$ copy mta0:vms*.* ker:
```

METHOD 1: DECODE THE HEX FILE

WARNING -- If you are running a pre-4.0 release of VMS, see Method 1.5 below!

The easiest way to install VMS Kermit is to "dehexify" the hexadecimal encoded task image. Follow these steps:

1. There should be a file VMSDEH.MAR on your disk. This is a Macro-32 program that decodes the hex file of the VMS Kermit task image. Compile and load this program:

   ```
   $ macro vmsdeh
   $ link vmsdeh
   ```

2. Run the VMSDEH program:

```
$ run vmsdeh
Please type the file name: vmsmit.hex
```

The VMSDEH program automatically creates KERMIT.EXE.

3. Now you can type "run kermit". Make sure it works. If not, try method 2.

4. Install KERMIT.EXE in the appropriate system area.

5. Install the help files, for instance:

```
$ LIBRARY/HELP/DELETE=KERMIT SYS$HELP:HELPLIB.HLB
$ LIBRARY/INSERT/HELP SYS$HELP:HELPLIB.HLB VMSSYS.HLP
$ LIBRARY/CREATE/HELP SYS$HELP:KERMIT.HLB VMSUSR.HLP
```

See the file `VMSINS.HLP` for further hints about the help files.

Note, the companion program to `VMSDEH.MAR` is `VMSHEX.MAR`. It creates hex files from binary files. You can use this program to encode any VMS binary file into printable form, and VMSDEH to decode it back into its original form, with all of its directory information intact.

METHOD 1.5: DEHEXIFY THE OLD KERMIT-32 HEX FILE

If you are running a version of VMS before 4.0, then the current release of Kermit-32 will not work on your system. If you tried Method 1, and the resulting Kermit program gave an error message like "invalid image header" when you tried to run it, this is probably the cause. In that case, you must use version 3.1 of VMS Kermit until you upgrade your VMS version. To use version 3.1 of VMS Kermit, follow the directions for Method 1, but use VMSDEH to dehexify the file `VMSV31.HEX` rather than `VMSMIT.HEX`. The result will be called `VMSMIT.EXE` rather than `KERMIT.EXE`. You can rename this file to `KERMIT.EXE` and install it and the help files normally, but note that the help files apply to the newer releases.

METHOD 2: ASSEMBLE THE MACRO FILES

Only use this method if method 1 fails for some reason. This method assembles all the Macro-32 source programs into object files, and then links them together into `KERMIT.EXE`, and then installs `KERMIT.EXE` and the help files in the system. NOTE: Kermit-32 is NOT written in Macro-32; it is written in Bliss-32, and these Macro files are output by the Bliss compiler. It is not recommended that changes be made to the Macro source, except as noted below. To build from Macro, follow these steps:

1. Make sure you have all the VMS*.MAR files on your current disk and directory.

2. There is a DCL procedure to assemble and link all of these files, but for some reason it assumes the files are called `KER*.*` instead of `VMS*.*`. There is another DCL procedure, `VMSREN.COM`, to rename them:

```
$ @vmsren
```

3. Now that the VMS Kermit files are renamed to `KER*.*`, you can run the DCL procedure VMSINS.COM to assemble and link them into KERMIT.EXE.

```
$ @vmsins

Kermit-32 Installation Procedure

Rebuild from sources? (YES or NO) yes

Which version? (BLISS or MACRO) macro

Install Kermit-32 on the system? yes
```

(note, you have to have write access to the system directories to do this)

```
This may take some time
```

(time passes)

```
Kermit-32 installation is complete.
```

METHOD 3: COMPILE THE BLISS SOURCES

You can use this method only if you have a Bliss-32 compiler. Even then, there's no reason to do this unless you want to change the sources in some way. Follow the steps in Method 2, except reply "bliss" instead of "macro" to the "Which version?" question.

FILES

Kermit-32 is built from a number of BLISS-32 sources and one MACRO-32 source. In order to make it possible for sites without BLISS-32 to build, MACRO-32 sources generated by BLISS-32 are also included for all of the BLISS modules. In the normal distribution of Kermit-32, all of the files start with the prefix "VMS". This will need to be changed to "KER" in order to build the program properly. The following files are distributed as part of Kermit-32:

`VMSTT.BLI` Common BLISS source for the terminal text output support.

`VMSGLB.BLI` Common BLISS source for the global storage for VMSMSG.BLI.

`VMSMSG.BLI` Common BLISS source for the protocol handling module.

`VMSCOM.REQ` Common BLISS require file which defines various common parameters. This is required by VMSMSG.BLI. This file must be renamed to KERCOM.REQ.

`VMSMIT.BWR` "Beware File" for Kermit-32 (read it!).

`VMSMIT.BLI` BLISS-32 source for the command parser, and some basic support routines.

`VMSFIL.BLI` BLISS-32 source for the file I/O.

`VMSTRM.BLI` BLISS-32 source for the terminal processing. This handles the driving of the terminal line for the transfers and the connect command processing.

`VMSSYS.BLI` System interface routines for the Kermit generic command processing.

`VMSGEN.MAR` Macro-32 source file that contains the REMOTE command text that is given to VMS. Sites desiring to change what DCL commands are used to process the various generic server commands can make those changes in this source. This also contains the text of the help message returned in response to the server generic help command.

`VMSERR.MSG` MESSAGE source for error messages used by VAX/VMS Kermit.

`VMSERR.REQ` BLISS-32 require file which defines the error codes. This is REQUIREd by the BLISS-32 sources.

`VMSMIT.MSS` SCRIBE source file for VMSMIT.DOC (this document).

`VMSMIT.RNH` RUNOFF source for the help files for VAX/VMS Kermit. When this is run through RUNOFF with /VARIANT=SYSTEM, it produces a .HLP (VMSSYS.HLP) file suitable for inserting into the system help library (SYS$HELP:HELPLIB.HLB) to provide a KERMIT topic for the system HELP command. When run through RUNOFF without the /VARIANT=SYSTEM, it produces a .HLP file (VMSUSR.HLP) to be stored on SYS$HELP: for use by the Kermit HELP command.

`VMSSYS.HLP` RUNOFF output file for system wide Kermit HELP.

`VMSUSR.HLP` RUNOFF output file for Kermit's HELP command.

`VMSREN.COM` Command file to rename VMS*.* to KER*.*.

`VMSINS.COM` Command file to build and install VAX/VMS Kermit.

`VMSMIT.HEX` A hexified version of .EXE file for VMS Kermit. This file can be dehexified using the supplied program. In the hexified form, the file should be transferable over any medium which handles normal text. This is the most reliable copy of the executable version of VMS Kermit.

`VMSHEX.MAR` Source for the hexification program. This is the program which was used to produce VMSMIT.HEX. It can also be used to produce hexified version of any (or at least almost any) Files-11 file. The dehexification program should then be able to reproduce a copy of the original file with the file parameters correctly set. Note that the format used for the hexified files is basically Intel hex format. There are some additional records used to store the record format, etc. Also, the file name as typed to the prompt from VMSHEX is stored in the hexified version of the file for use by the dehexification program. By doing this, it is possible to store more than one binary file with a single hexified file.

`VMSDEH.MAR` Source for the dehexification program.

`VMSV31.*` Version VMS Kermit, the last version that will run under release 3.x of VMS. Versions 3.2 and later require VMS release 4.0 or later.

`VMSV3x.MEM` ocumentation on the changes between releases 3.1 and 3.1, and 3.2 and 3.3 of Kermit-32, and additional installation information.

OTHER INSTALLATION CONSIDERATIONS

As distributed, Kermit-32 should work on any VAX/VMS system (version 4.0 and later). Customization is possible with or without a BLISS-32 compiler. Default parameter values may be changed by changing the appropriate LITERALs in the BLISS-32 source for VMSMSG, or the actual values which are stored in the routine MSG_INIT in the MACRO-32 source for VMSMSG.

Sites can also easily change the commands which are used for processing the generic server functions (REMOTE commands when running as a server). The text which makes up these commands is in the file VMSGEN.MAR, along with the text of the REMOTE HELP message. This allows a site to make use of local programs for performing some of the commands (perhaps using FINGER to perform the WHO command, etc.).

If you want to allow your users to assign external terminal lines for connect- ing to remote systems from the VAX, e.g. by dialing out, you will have to con- figure those lines to allow the desired access. Otherwise, users will get a message like "No privilege for attempted operation" when they do a SET LINE command. Sample commands for terminal TXA0: might include:

```
$ SET PROTECTION=(W:R) TXA0:/DEVICE
```

or

```
$ SET PROTECTION=(W:RWLP)/DEVICE/OWNER=[1,4] TXA0:
```

or

```
$ SET ACL/OBJECT=DEVICE/ACL=(IDENTIFIER=INTERACTIVE,OPTIONS=NONE,-
  ACCESS=READ+WRITE) TXA0:
```

Consult your VAX/VMS system manager’s manual for the ramifications (especially on security) of each of these commands.

12. DECSYSTEM-20 KERMIT

Authors: Frank da Cruz, Bill Catchings, Columbia University
Language: MACRO-20
Version: 4.2(262)
Date: January 1988

Kermit-20 Capabilities At a Glance:

Local operation:	Yes
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
^X/^Y interruption:	Yes
Filename collision avoidance:	Yes
Timeouts:	Yes
8th-bit prefixing:	Yes
Repeat character compression:	Yes
Alternate block check types:	Yes
Communication settings:	Yes
Transmit BREAK:	Yes
IBM mainframe communication:	Yes
Transaction logging:	Yes
Session logging:	Yes
Debug logging:	Yes
Raw transmit:	Yes
Login scripts:	Yes
Act as server:	Yes
Talk to server:	Yes
Advanced commands for servers:	Yes
Local file management:	Yes
Command/init files:	Yes
Long packets:	No
Sliding windows:	No
Handle file attributes:	No

Kermit-20 is a program that implements the Kermit file transfer protocol for the Digital Equipment Corporation DECSYSTEM-20 mainframe computer. It is written in MACRO-20 assembly language and should run on any DEC-20 system with version 4 of TOPS-20 or later.

The Kermit-20 section will describe the things you should know about the DEC-20 file system in order to make effective use of Kermit, and then it will describe the special features of the Kermit-20 program.

12.1. The DEC-20 File System

The features of the DEC-20 file system of greatest interest to Kermit users are the form of the file specifications, and the distinctions between text and binary files.

DEC-20 File Specifications

DEC-20 file specifications are of the form

```
DEVICE:<DIRECTORY>NAME.TYPE.GEN;ATTRIBUTES
```

where the DIRECTORY, NAME, and TYPE may each be up to 39 characters in length, GEN is a generation (version number), and various attributes are possible (protection code, account, temporary, etc). Generation and attributes are normally omitted. Device and directory, when omitted, default to the user's own (or "connected") disk and directory. Thus NAME.TYPE is normally sufficient to specify a file, and only this information is sent along by Kermit-20 with an outgoing file.

The device, directory, name, and type fields may contain uppercase letters, digits, and the special characters "-" (dash), "_" (underscore), and "$" (dollar sign). There are no imbedded or trailing spaces. Other characters may be included by prefixing them (each) with a Control-V. The fields of the file specification are set off from one another by the punctuation indicated above.

The device field specifies a physical or "logical" device upon which the file is resident. The directory field indicates the area on the device, for instance the area belonging to the owner of the file. Kermit-20 does not transmit the device or directory fields to the target system, and does not attempt to honor device or directory fields that may appear in incoming file names; for instance, it will not create new directories.

The name is the primary identifier for the file. The type, also called the "extension", is an indicator which, by convention, tells what kind of file we have. For instance FOO.FOR is the source of a Fortran program named FOO; FOO.REL might be the relocatable object module produced by compiling FOO.FOR; FOO.EXE could an executable program produced by LOADing and SAVing FOO.REL, and so forth.

The DEC-20 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, including no characters at all; a "%" matches any single character. Here are some examples:

*.FOR All files of type FOR (all Fortran source files) in the connected directory.

FOO.* Files of all types with name FOO.

F*.* All files whose names start with F.

F*X*.* All files whose names start with F and contain at least one X.

%.* All files whose names are exactly one character long.

.%%% All files whose types are at least three characters long.

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.

Text Files and Binary Files

The DEC-20, like most computers, has a file system with its own peculiarities. Like many other systems, the DEC-20 makes a distinction between *text files* and *binary files*. Text files are generally those composed only of printing characters (letters, digits, and punctuation) and "carriage control" characters (carriage return, line feed, form feed, tab). Text files are designed to be read by people. Binary files are designed to be read by a computer program, and may have any contents at all. If you use the DEC-20 TYPE command to display a text file on your terminal, the result will be intelligible. If you type a binary file on your terminal, you will probably see mainly gibberish. You can not always tell a text file from a binary file by its name or directory information, though in general files with types like .TXT, .DOC, .HLP are textual (as are "source files" for computer programs like text formatters and programming language compilers), and files with types like .EXE, .REL, .BIN are binary.

The DEC-20 has an unusual word size, 36 bits. It differs from most other systems by storing text in 7-bit, rather than 8-bit, bytes. Since text is encoded in the 7-bit ASCII character set, this allows more efficient use of storage. However, the word size is not a multiple of the normal byte size. The DEC-20 therefore stores five 7-bit characters per word, with one bit left over.

It is also possible to store files with other byte sizes. The common layouts of bytes within a word are shown in Figure 12-1.

7: Text Files: Five 7-bit bytes per word.

```
+------+------+------+------+------++
|      |      |      |      |      ||
+------+------+------+------+------++
0      7      14     21     28     35
```

Normally, bit 35 is unused and set to zero. However, in EDIT (or SOS, or OTTO) line-numbered files, bit 35 is set to 1 when the word contains a line number.

8: "Foreign" binary files: Four 8-bit bytes per word.

```
+-------+-------+-------+-------+---+
|       |       |       |       |   |
+-------+-------+-------+-------+---+
0       8       16      24      32  35
```

Bits 32-35 are unused.

36: "Native" binary files: One 36-bit byte per word.

```
+-----------------------------------+
|                                   |
+-----------------------------------+
0                                   35
```

All bits are used.

Figure 12-1: DECSYSTEM-20 Word/Byte Organization

The minimum unit of disk allocation on the DEC-20 is a *page*, 512 36-bit words, or 2560 7-bit characters, or 2048 8-bit bytes. Any file that contains at least one bit of information occupies at least a full page on the disk. The directory information for a file includes the number of pages occupied on the disk, the bytesize of the file, and the number of bytes of that size which are in the file. This information can be seen by using the DEC-20 VDIRECTORY command, for instance

```
@vdir foo.*

   PS:<MY-DIRECTORY>
 Name     Protection    Pages Bytes(Size) Creation
 FOO.COM.1;P774242          1 384(8)     27-Dec-83
    MAC.1;P774242           1 152(7)     27-Dec-83
   .REL.1;P774242           1 39(36)     27-Dec-83
   .EXE.1;P774242           2 1024(36)   27-Dec-83

 Total of 5 pages in 4 files
```

In this example, `FOO.MAC` occupies 1 page, and is composed of 152 7-bit bytes. This file is textual (program source for the MACRO assembler), 152 characters long. Programs which read text files (such as text editors, program compilers, the TYPE command, etc) determine the end of a file from the byte count specified in the directory. Kermit-20 determines the end of file in the same way, so although `FOO.MAC` occupies an entire 2560-

byte page of storage, only the first 152 characters are transmitted. Binary files, such as `FOO.EXE` (an executable DEC-20 program), tend to occupy full pages. In this case too, Kermit-20 uses the byte count to determine the end of file.

Why do you need to know all this? In most cases, you don't. It depends on whether you are using the DEC-20 as your "home base".

Using a Microcomputer to Archive DEC-20 Files

Most computers (other than the DEC-10 and DEC-20) store characters in 8-bit bytes. Let's call any such system an 8-bit-byte system. Microcomputers that run CP/M or MS-DOS or PC-DOS, and any computers than run Unix, store these 8-bit bytes in a linear sequence. Certain other 8-bit-byte systems (PDP-11 or VAX systems with FILES-11, IBM mainframes) have more complex file formats. This discussion applies to all linear 8-bit-byte systems, including most popular microcomputers.

Kermit can send any "native" DEC-20 sequential file, text or binary, to an 8-bit-byte system and bring it back to the DEC-20 restored to its original form. If you are using a microcomputer to archive your DEC-20 files, you need never concern yourself with details of byte size or file format. The same holds true between two DEC-20s, or a DEC-10 and a DEC-20.

There is, however, one special complication of which you should be aware. Certain microcomputer operating systems, notably CP/M, do not have an entirely satisfactory way of indicating the end of file. The file length is recorded in blocks rather than bytes. For text files, the end of file is marked within a block by inserting a Control-Z after the last data character. Binary files, however, might easily contain Control-Z characters as data. Therefore, in order not to lose data, these systems must transmit binary files in complete blocks. If the binary file is of foreign origin (for instance, from a DEC-20), and it did not happen to fill up the last block when it was transferred to the micro, then when that file is sent back to the system of origin in "binary mode," junk will appear at the end (if it is sent back in "text mode," it could be truncated at the first data byte that happened to correspond to Control-Z). For DEC-20 programs in `.EXE` format, this generally has no effect on the runnability or behavior of the program. But for other binary files, particularly internal format numerical data or relocatable program object (`.REL`) files, the junk could have bad effects. For instance, extraneous data at the end of a `.REL` file will generally cause LINK to fail to load the file.

Most microcomputer Kermit programs have commands to control end-of-file detection -- commands like SET FILE TEXT, SET FILE BINARY, SET EOF CTRLZ.

Using the DEC-20 to Archive Microcomputer Files

You can use Kermit to send textual files from a microcomputer or any 8-bit system to the DEC-20 with no special provisions, since Kermit-20 stores incoming characters in 7-bit bytes as text unless you explicitly instruct it otherwise. But Kermit-20 has no automatic way of distinguishing an incoming binary file from an incoming text file.[2] Binary files from 8-bit-byte systems generally contain significant data in the 8th bit, which would be lost if the incoming characters were stored in 7-bit bytes, rendering the file useless when sent back to the original system. Thus if you want to use Kermit to store foreign 8-bit binary data on the DEC-20, you must tell it to store such files with a bytesize of 8 rather than 7. This can be the source of much confusion and inconvenience. In particular, you cannot use a "wildcard send" command to send a mixture of text and binary files from an 8-bit-byte system to the DEC-20; rather, you must send all text files with Kermit-20's file bytesize set to 7, and all 8-bit binary files with the bytesize set to 8.

---

[2]Unless the incoming file has an "ITS Binary Header"; see below.

Once you get the foreign binary file into the DEC-20, stored with the correct bytesize (as `FOO.COM` is stored in the example above), you need take no special measures to send it back to its system of origin. This is because Kermit-20 honors the bytesize and byte count from the directory. For instance, if you told Kermit-20 to `SEND FOO.*`, every file in the example above would be transmitted in the correct manner, automatically.

The previous discussion assumes you want to store text files in usable form on the DEC-20. However, if you are using the DEC-20 purely as a repository for your microcomputer files, and you have no desire to display or share the contents of those files on the DEC-20, you can SET FILE BYTESIZE 8 for all incoming files, both text and binary. When the files are sent back to a microcomputer, they will be stored correctly.

Files Kermit-20 Cannot Handle

The Kermit protocol can only accommodate transfer of *sequential* files, files which are a linear sequence of bytes (or words).

Some files on the DEC-20 are not sequential, and cannot be successfully sent or received by Kermit-20. These include directory files, files with holes (missing pages), ISAM files, and RMS files. These files require external information (kept in the DEC-20's file descriptor block and/or index table) in order to be reconstructed; when sending files, Kermit-20 presently transmits only the file name and the contents of the file. External control information and file attributes are not transmitted.

12.2. Program Operation

Kermit-20's prompt is "`Kermit-20>`". Kermit-20 will accept a single command on the Exec command line, like this:

```
@
@Kermit send foo.bar
```

the file is sent

```
@
```

or you can run the program interactively to issue several commands, like this:

```
@
@Kermit

TOPS-20 Kermit version 4.2(262)

Kermit-20>send foo.*
```

files are sent

```
Kermit-20>statistics
```

performance statistics are printed

```
Kermit-20>receive
```

files are received

```
Kermit-20>exit
@
```

During interactive operation, you may use the TOPS-20 help ("?") and recognition (ESC) features freely while typing commands. A question mark typed at any point in a command displays the options available at that point; typing an ESC character causes the current keyword or filename to be completed (or default value to be supplied), and a "guide word" in parentheses to be typed, prompting you for the next field. If you have not typed sufficient characters to uniquely specify the keyword or filename (or if there is no default value) then a beep will be sounded and you may continue typing.

Command keywords may be abbreviated to their shortest prefix that sets them apart from any other keyword valid in that field.

If you have a file called `KERMIT.INI` in your login directory, Kermit-20 will execute an automatic TAKE command on it upon initial startup. `KERMIT.INI` may contain any Kermit-20 commands, for instance SET commands, or DEFINEs for SET macros to configure Kermit-20 to various systems or communications media.

Kermit-20 provides most of the commands possible for an "ideal" Kermit program, as described in the main part of the *Kermit User Guide*. The following sections will concentrate on system-dependent aspects of Kermit-20.

12.3. Remote and Local Operation

Kermit-20 normally runs in remote mode, with the user sitting at a PC. But Kermit-20 can also run in local mode. Local operation of Kermit-20 is useful if the DEC-20 has an autodialer, or a hardwired connection to another computer. When in local mode, file transfer takes place over an assigned TTY line, and Kermit-20 is free to update your screen with status information, and to listen to your keyboard for interrupt characters.

Local Operation of Kermit-20:

```
 DECSYSTEM-20
+---------------------------------------+
|                                       |
|   +--------------------+              |
|   | Your Job           |              |
|   |                    |              |
|   |   +------------+   | <--Commands  | Your Job's
|   |   | Kermit-20  +---+--------------+----------------- (-: You
|   |   |            |   | Display--->  | Controlling TTY
|   |   |            |   |              |
|   |   |            |   |              |
|   |   |            |   | <--Packets   | Kermit's
|   |   |            +---+--------------+------------------> Remote
|   |   +------------+   | Packets-->   | Assigned TTY      System
|   |                    |              |
|   +--------------------+              |
|                                       |
+---------------------------------------+
```

Figure 12-2: DEC-20 Kermit Local Operation

Kermit-20 enters local mode when you issue a SET LINE *n* command, where *n* is the octal TTY number of any line other than your own controlling terminal.

12.4. Conditioning Your Job for Kermit

Kermit-20 does as much as it can to condition your line for file transfer. It saves all your terminal and link settings, and restores them after use. However, there are some sources of interference over which Kermit-20 can have no control. In particular, messages issued by superior or parellel forks could become mingled with Kermit packets and slow things down or stop them entirely. For this reason, before using Kermit-20 for any extended period, you should:

- Type the Exec commands SET NO MAIL-WATCH and SET NO ALERTS

- Make sure you don't have any print or batch jobs pending that were submitted with the /NOTIFY option.
- Make sure you don't have any superior or parallel forks that have enabled terminal interrupts on Control-A; these could prevent Kermit packets (which start with Control-A) from getting through.

After running Kermit, you can restore your mail-watch and alerts by hand. Alternatively, you could have an Exec command file for invoking Kermit like this:

```
set no alerts
set no mail-watch
kermit
set mail-watch
set alert  1:00PM Go to lunch
set alert  6:00PM Go to dinner
set alert 11:30PM Go to sleep
```

12.5. Kermit-20 Commands

This section describes the Kermit-20 commands -- in detail where they differ from the "ideal" Kermit, briefly where they coincide. Kermit-20 has the following commands:

BYE	to remote server.
CLEAR	a stuck connection
CLOSE	log file and stop logging remote session.
CONNECT	as terminal to remote system.
CWD	change local working directory.
DEFINE	macros of Kermit-20 commands.
DELETE	local files.
DIRECTORY	listing of local files.
ECHO	a line of text.
EXIT	from Kermit-20.
FINISH	Shut down remote server.
GET	remote files from server.
HELP	about Kermit-20.
INPUT	characters from communication line.
LOCAL	prefix for local file management commands.
LOG	remote terminal session.
OUTPUT	characters to communication line.
PAUSE	between commands.
PUSH	to TOPS-20 command level.
QUIT	from Kermit-20
RECEIVE	files from remote Kermit.
REMOTE	prefix for remote file management commands.
RUN	a DEC-20 program.
SEND	files to remote Kermit.
SERVER	mode of remote operation.
SET	various parameters.
SHOW	various parameters.
SPACE	inquiry.
STATISTICS	about most recent file transfer.
TAKE	commands from a file.
TRANSMIT	a file "raw".
TYPE	a local file.

12.5.1. Commands for File Transfer

Kermit-20 provides the standard SEND, RECEIVE, and GET commands for transferring files using the Kermit protocol.

The SEND Command

Syntax:

Sending a single file:

`SEND` *nonwild-filespec1* `(AS) [`*filespec2*`]`

Sending multiple files:

`SEND` *wild-filespec1* `(INITIAL) [`*filespec2*`]`

The SEND command causes a file or file group to be sent from the DEC-20 to the other system. There are two forms of the command, depending on whether *filespec1* contains wildcard characters ("`*`" or "`%`"). Kermit-20 automatically recognizes the two cases and issues the appropriate guide word, (AS) or (INITIAL), depending on the form of *filespec1*.

Sending a File Group

If *filespec1* contains wildcard characters then all matching files will be sent, in alphabetical order (according to the ASCII collating sequence) by name. If a file can't be opened for read access, it will be skipped. The initial file in a wildcard group can be specified with the optional *filespec2*. This allows a previously interrupted wildcard transfer from where it left off, or it can be used to skip some files that would be transmitted first.

Sending a Single File

If *filespec1* does not contain any wildcard characters, then the single file specified by *filespec1* will be sent. Optionally, *filespec2* may be used to specify the name under which the file will arrive at the target system; *filespec2* is not parsed or validated in any way by Kermit-20, but lower case letters are raised to upper case, and leading "whitespace" (blanks and tabs) are discarded. If *filespec2* is not specified, Kermit-20 will send the file with its own name.[3]

SEND Command General Operation:

Files will be sent with their DEC-20 filename and filetype (for instance `FOO.BAR`, no device or directory field, no generation number or attributes). If you expect to be sending files whose names contain characters that would be illegal in filenames on the target system, and you know that the Kermit on the target system does not have the ability to convert incoming filenames, you can issue the SET FILE NAMING NORMAL-FORM command to have Kermit-20 replace suspect characters by `X`'s.

Each file will be sent according to its bytesize and byte count from the directory unless you specify otherwise using SET FILE BYTESIZE, or unless the file has an "ITS Binary" header. If the bytesize is 8, then four 8-bit bytes will be sent from each DEC-20 36-bit word, and the low order four bits will be skipped. If other than 8, then five 7-bit bytes will be sent from each word, with the 8th bit of the 5th character set to the value of the remaining bit ("bit 35") from the word.[4]

---

[3] Control-V's, which are used to quote otherwise illegal characters in DEC-20 file specifications, are stripped.

[4] This is the same method used by the DEC-20 to encode 36-bit data on "ANSI-ASCII" tapes. It allows not only DEC-20 binary files, but also the line-sequence-numbered files produced by EDIT, SOS, or OTTO, which use bit 35 to distinguish line numbers from text, to be sent and retrieved correctly.

If communication line parity is being used (see SET PARITY), Kermit-20 will request that the other Kermit accept a special kind of prefix notation for binary files. This is an advanced feature, and not all Kermits have it; if the other Kermit does not agree to use this feature, binary files cannot be sent correctly. This includes executable programs (like DEC-20 `.EXE` files, CP/M `.COM` files), relocatable object modules (`.REL` files), as well as text files with line sequence numbers.

Kermit-20 will also ask the other Kermit whether it can handle a special prefix encoding for repeated characters. If it can, then files with long strings of repeated characters will be transmitted very efficiently. Columnar data, highly indented text, and binary files are the major beneficiaries of this technique.

If you're running Kermit-20 locally, for instance dialing out from the DEC-20 to another system using an autodialer, you should have already run Kermit on the remote system and issued either a RECEIVE or a SERVER command. Once you give Kermit-20 the SEND command, the name of each file will be displayed on your screen as the transfer begins; a "`.`" will be displayed for every 5 data packets sucessfully sent, and a "`%`" for every retransmission or timeout that occurs (you may also elect other typeout options with the SET DEBUG command). If the file is successfully transferred, you will see "`[OK]`", otherwise there will be an error message. When the specified operation is complete, the program will sound a beep. If you see many "`%`" characters, you are probably suffering from a noisy connection. You may be able to cut down on the retransmissions by using SET SEND PACKET-LENGTH to decrease the packet length; this will reduce the probability that a given packet will be corrupted by noise, and reduce the time required to retransmit a corrupted packet.

During local operation, you can type Control-A at any point during the transfer to get a brief status report. You may also type Control-X or Control-Z to interrupt the current file or file group.

The RECEIVE Command

Syntax: `RECEIVE [`*filespec*`]`

The RECEIVE command tells Kermit-20 to receive a file or file group from the other system. If only one file is being received, you may include the optional *filespec* as the name to store the incoming file under; otherwise, the name is taken from the incoming file header. Even if the name in the header is not a legal TOPS-20 file name, Kermit-20 will store it under that name, in which case you can refer to it later only by quoting each illegal character (spaces, control characters, etc) with Control-V. If for some reason an incoming filename simply cannot be converted to legal form, the file will be saved as `-UNTRANSLATABLE-FILENAME-.KERMIT` (new generation). You may also use SET FILE NAMING NORMAL-FORM to have Kermit-20 choose more conventional names for incoming files.

If an incoming file has the same name as an existing file, Kermit-20 just creates a new generation of the same name and type, for instance `FOO.BAR.3, FOO.BAR.4`. The oldest generation will be automatically deleted, but you can still UNDELETE it.

Incoming files will all be stored with the prevailing bytesize, 7 by default, which is appropriate for text files. If you are asking Kermit-20 to receive binary files from a microcomputer or other 8-bit system, you must first type SET FILE BYTESIZE 8. Otherwise, the 8th bit of each byte will be lost and the file will be useless when sent back to the system of origin.

If you have SET PARITY, then 8th-bit prefixing will be requested. If the other side cannot do this, binary files cannot be transferred correctly. In all cases, Kermit-20 will request the other Kermit to compress repeated characters; if the other side can do this (not all Kermits know how) there may be a significant improvement in transmission speed.

If an incoming file does not arrive in its entirety, Kermit-20 will normally discard it; it will not appear in your directory. You may change this behavior by using the command SET INCOMPLETE KEEP, which will cause as much of the file as arrived to be saved in your directory.

If you are running Kermit-20 locally, you should already have issued a SEND command[5] to the remote Kermit, and then escaped back to DEC-20 Kermit. As files arrive, their names will be displayed on your screen, along with "`.`" and "`%`" characters to indicate the packet traffic; you can type Control-A during the transfer for a brief status report.

If a file arrives that you don't really want, you can attempt to cancel it by typing Control-X; this sends a cancellation request to the remote Kermit. If the remote Kermit understands this request (not all implementations of Kermit support this feature), it will comply; otherwise it will continue to send. If a file group is being sent, you can request the entire group be cancelled by typing Control-Z.

The GET Command

Syntax: `GET [`*remote-filespec*`]`

The GET command requests a remote Kermit server to send the file or file group specified by *remote-filespec*. This command can be used only when there is a Kermit server on the other end of the line. This means that you must have CONNECTed to the other system, logged in, run Kermit there, issued the SERVER command, and escaped back to the DEC-20, or else you Kermit-20 is in remote mode, TAKEing commands from a file, and interacting with a local Kermit server.

The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. You should not put a trailing comment on the GET command, since this will be sent as part of the remote filespec.

If you need to include otherwise illegal characters such as "`!`" or "`;`" (the normal command comment delimeters), "`?`" (the command help character), "`@`" (the indirect command file indicator), or certain control characters, then you should precede each such character by a Control-V. Kermit-20 will discard these Control-V quoting prefixes before sending the file specification to the remote host.

If you want to store the incoming file name with a different name than the remote host sends it with, just type GET alone on a line; Kermit-20 will prompt you separately for the source (remote) and destination (local) file specification. If more than one file arrives, only the first one will be stored under the name given; the rest will be stored under the names they are sent with. Example:

```
Kermit-20>get
 Remote Source File: profile exec a1
 Local Destination File: profile.exec
```

As files arrive, their names will be displayed on your screen, along with "`.`" and "`%`" characters to indicate the packet traffic. As in the RECEIVE command, you may type Control-A to get a brief status report, ^X to request that the current incoming file be cancelled, ^Z to request that the entire incoming batch be cancelled.

If the remote Kermit is not capable of server functions, then you will probably get an error message back from it like "Illegal packet type". In this case, you must connect to the other Kermit, give a SEND command, escape back, and give a RECEIVE command.

[5] *not* SERVER -- use the GET command to receive files from a Kermit server.

The STATISTICS Command

Give statistics about the most recent file transfer. For instance, here's what Kermit-20 displayed after transmitting a short binary file, using repeated-character compression:

```
Maximum number of characters in packet:  80 received; 80 sent
Number of characters transmitted in 2 seconds
       Sent:      34          Overhead:       34
       Received:  107         Overhead:       -408
 Total received:  141         Overhead:       -374
Total characters transmitted per second:      70
Effective data rate:   2570 baud
Efficiency:            214.1667 per cent
Interpacket pause in effect: 0 sec

Timeouts: 0
NAKs:     0
```

Note that the data compression allowed the effective baud rate to exceed the actual speed of the communication line, which in this case happened to be 1200 baud. The efficiency is displayed only if the actual baud rate is known.

12.5.2. Server Operation

The SERVER Command

The SERVER command puts a remote Kermit-20 in "server mode", so that it receives all further commands in packets from the local Kermit. The Kermit-20 server is capable (as of this writing) of executing the following remote server commands: SEND, GET, FINISH, BYE, REMOTE DIRECTORY, REMOTE CWD, REMOTE SPACE, REMOTE DELETE, REMOTE TYPE, REMOTE HELP.

Any nonstandard parameters should be selected with SET commands before putting Kermit-20 into server mode, in particular the file bytesize. The DEC-20 Kermit server can send most files in the correct manner automatically, by recognizing the DEC-20 file bytesize. However, if you need to ask the DEC-20 Kermit server to receive binary files from an 8-bit-byte system (that is, from almost any system that's not a DEC-10 or DEC-20) you must issue the SET FILE BYTESIZE 8 command before putting it into server mode, and then you must only send 8-bit binary files. You cannot send a mixture of text files and 8-bit binary files to a Kermit-20 server.

Commands for Servers

When running in local mode, Kermit-20 allows you to give a wide range of commands to a remote Kermit server, with no guarantee the that the remote server can process them, since they are all optional features of the protocol. Commands for servers include the standard `SEND`, `GET`, `BYE`, and `FINISH` commands, as well as the `REMOTE` command.

These commands are generally issued when Kermit-20 is in local mode, i.e. you have already connected to another system, run Kermit there and put it into server mode, and escaped back to Kermit-20. However, Kermit-20 also allows you to operate in the opposite direction, i.e. Kermit-20 is the remote Kermit, and the local Kermit is in server mode. This is handy when, for instance, you want to transfer a disparate collection of files that can't be readily specified by a wildcard group, all in a single, unattended operation. In this case, you can create a TAKE command file for Kermit-20 that SENDs and/or GETs the desired files, and then shuts down local server when done, e.g.:

```
set delay 0             ; No need to pause before sending
; Connect to own directory, leave a blank line for password.
cwd me:

log transactions        ; Keep a log
```

```
; Change directories on the PC.
remote cwd \kermit

send ker:mskerm.doc     ; Send the MS-DOS Kermit manual
send ker:mskerm.bwr     ; Send the MS-DOS Kermit "beware file"
; Now change to the MS-DOS binaries area
remote cwd \bin

send kb:msvibm.exe      ; Send the executable DOS Kermit program
; Put DOS back in default directory
remote cwd \chris

; Connect back to default directory on the DEC-20
cwd me:

close transactions      ; Close transaction log
send transaction.log    ; Send it
finish                  ; Shut down DOS Kermit server
```

Commands to servers (GET, BYE, FINISH, REMOTE) can be issued from a remote Kermit-20 only by means of a TAKE file. When Kermit-20 is local (i.e. after SET LINE), you can issue these commands interactively as well.

The REMOTE Command

Send the specified command to the remote server. If the server does not understand the command (all of these commands are optional features of the Kermit protocol), it will reply with a message like "Unknown Kermit server command". If does understand, it will send the results back, and they will be displayed on the screen. The REMOTE commands are:

CWD [*directory*] Change Working Directory. If no directory name is provided, the server will change to the default or home directory. Otherwise, you will be prompted for a password, and the server will attempt to change to the specified directory. The password is entered on a separate line, and does not echo as you type it. If access is not granted, the server will provide a message to that effect. Do not put trailing comments after a REMOTE CWD command, or after the password.

DELETE *filespec* Delete the specified file or files. The names of the files that are deleted will appear on your screen.

DIRECTORY [*filespec*]
The names of the files that match the given file specification will be displayed on your screen, perhaps along with size and date information for each file. If no file specification is given, all files from the current directory will be listed.

HELP Provide a list of the functions that are available from the server.

HOST [*command*] Pass the given command to the server's host command processor, and display the resulting output on your screen.

SPACE Provide information about disk usage in the current directory, such as the quota, the current storage, the amount of remaining free space.

TYPE *filespec* Display the contents of the specified file on your screen.

12.5.3. Commands for Local File Management

Syntax: `LOCAL [`*command*`]`

Execute the specified command on the local system -- on the DEC-20 where Kermit-20 is running. These commands provide some local file management capability without having to leave the Kermit-20 program.

CWD [*directory*] Change working directory, or, in DEC-20 terminology, CONNECT to the specified directory. If a password is required, you will be prompted for one. Do not include a trailing comment after the password.

DELETE *filespec* Delete the specified file or files, but do not expunge them (unless you have SET EXPUNGE ON).

DIRECTORY [*filespec*]
Provide a directory listing of the specified files.

RUN [*filespec*] Attempts to run the specified file, which must be in "`.EXE`" format (`.EXE` is the default filetype), in an inferior fork. Control returns to Kermit-20 when the program terminates. Once you have used this command, you can restart the same program by issuing a RUN command with no arguments. If you RUN `SYSTEM:EXEC`, then you will be able to issue TOPS-20 commands without leaving Kermit; you can get back to Kermit from the EXEC by typing the EXEC POP command.

SPACE Show how much space is used and remaining in the current directory.

TYPE Display the contents of the specified file or files at your terminal. This works like the DEC-20 TYPE command, except that if a file has a bytesize of 8, Kermit-20 will do 8-bit input from it rather than 7-bit. Also, the DEC-20 Control-O command discards output only from the file currently being displayed; if multiple files are being typed, then output will resume with the next file.

The LOCAL commands may also be used without the "LOCAL" prefix.

12.5.4. The CONNECT Command

Syntax: `CONNECT [`*number*`]`

Establish a terminal connection to the system connected to the octal TTY number specified here or in the most recent SET LINE command, using full duplex echoing and no parity unless otherwise specified in previous SET commands. Get back to Kermit-20 by typing the escape character followed by the letter C. The escape character is Control-Backslash (`^\`) by default. When you type the escape character, several single-character commands are possible:

`C` Close the connection and return to Kermit-20.
`S` Show status of the connection; equivalent to SHOW LINE.
`P` Push to a new Exec. POP from the Exec to get back to the connection.
`Q` If a session log is active, temporarily Quit logging.
`R` Resume logging to the session log.
`B` Send a simulated BREAK signal.
`?` List all the possible single-character arguments.
`^\` (or whatever you have set the escape character to be):
Typing the escape character twice sends one copy of it to the connected host.

You can use the SET ESCAPE command to define a different escape character, and SET PARITY, SET DUPLEX, SET HANDSHAKE, SET FLOW, and SET SPEED to change those communication-line-oriented parameters. In order for the simulated BREAK signal to work, TOPS-20 must know the speed of the terminal. If it does not, you may use the SET SPEED command. Type the SHOW LINE command for information about your current communication settings.

Kermit-20 does not have any special autodialer interface. It assumes that the connection has already been made and the line assigned.

12.5.5. The SET, SHOW, and DEFINE Commands

SET is used for establishing or changing parameters, DEFINE lets you group several SET commands together into a single "macro" command, and SHOW lets you examine current settings or macro definitions.

The SET Command

Syntax: `SET` *parameter* [*option* [*value*]]

Establish or modify various parameters for file transfer or terminal connection. You can examine their values with the SHOW command. The following parameters may be SET:

BREAK	Adjust the BREAK simulation parameter
BLOCK-CHECK	Packet transmission error detection method
DEBUGGING	Record or display state transitions or packets
DELAY	How long to wait before starting to send
DUPLEX	For terminal connection, FULL or HALF
ESCAPE	Character for terminal connection
FILE	For setting file parameters like byte size
FLOW-CONTROL	For enabling or disabling XON/XOFF flow control
HANDSHAKE	For turning around half duplex communication line
IBM	For communicating with an IBM mainframe
INCOMPLETE	What to do with an incomplete file
INPUT	For specifying behavior of the INPUT command
ITS-BINARY	For recognizing a special 8-bit binary file format
LINE	TTY line to use for file transfer or CONNECT
PARITY	Character parity to use
PROMPT	Change the program's command prompt
RECEIVE	Various parameters for receiving files
RETRY	How many times to retry a packet before giving up
SEND	Various parameters for sending files
SPEED	Baud rate of communication line
TVT-BINARY	For negotiating binary mode on ARPANET

The DEFINE command may be used to compose "macros" by combining SET commands. Those SET commands which differ from the "ideal" Kermit are now described in detail.

SET BREAK

Syntax: `SET BREAK` *n* Specify the number of nulls to be sent at 50 baud to simulate a BREAK signal when connected to a remote host via SET LINE and CONNECT.

SET DEBUG

Syntax: `SET DEBUG` *options*

Record the packet traffic, either on your terminal or in a file. Some reasons for doing this would be to debug a version of Kermit that you are working on, to record a transaction in which an error occurred for evidence when reporting bugs, or simply to vary the display you get when running Kermit-20 in local mode. Options are:

STATES
: Show Kermit state transitions and packet numbers (brief).

PACKETS
: Display each incoming and outgoing packet (lengthy).

OFF
: Don't display or record debugging information (this is the normal mode). If debugging was in effect, turn it off and close any log file.

The debugging information is recorded in the file specified by the most recent LOG DEBUGGING command, `DEBUGGING.LOG` by default.

SET ESCAPE

`SET ESCAPE` *octal-number*

Specify the control character you want to use to "escape" from remote connections back to Kermit-20. The default is 34 (Control-\). The number is the octal value of the ASCII control character, 1 to 37 (or 177), for instance 2 is Control-B. After you type the escape character, you must follow it by a one of the single-character "arguments" described under the CONNECT command, above.

SET EXPUNGE

`SET EXPUNGE ON` *or* `OFF`

Tell whether you want a DELETE command (either the LOCAL DELETE command or a REMOTE DELETE command sent to a Kermit-20 server) to expunge files as it deletes them. On the DEC-20, a deleted file continues to take up space, and may be "undeleted" at a later time in the same session. To expunge a deleted file means to remove it completely and irrevocably, freeing its space for further use. EXPUNGE is OFF by default; deleted files are not automatically expunged. SET EXPUNGE applies only to files that are deleted explicitly by Kermit-20, and not to files that are implicitly deleted when new generations of existing files are created.

SET FILE

Syntax: `SET FILE` *parameter keyword*

Establish file-related parameters:

BYTESIZE *keyword or number*

Byte size for DEC-20 file input/output. The choices are SEVEN (7), EIGHT (8), and AUTO.

SEVEN (or 7) Always store or retrieve five 7-bit bytes per word. When sending a file, ignore the file bytesize and do 7-bit input from the file. There would be no reason to use this option except to explicitly force an 8-bit file to be treated as a 7-bit file.

EIGHT (or 8) Always store or retrieve four 8-bit bytes per word. When sending a file, ignore the file bytesize and do 8-bit input from the file. This command is necessary when receiving binary files from 8-bit-byte systems, such as most microcomputers.

AUTO Equivalent to SEVEN for incoming files, and for outgoing files means to use EIGHT if the DEC-20 file bytesize (as shown by the Exec VDIR command) is 8, otherwise use SEVEN. The default is AUTO.

The DEC-20 can send any mixture of file types in the correct way automatically, but you *must* set the file bytesize to 8 for any incoming 8-bit binary files, and to AUTO (i.e. 7) for any incoming text files or DEC-20 binary files.

NAMING UNTRANSLATED *or* NORMAL-FORM

If NORMAL-FORM the names of incoming or outgoing files will be converted to contain only uppercase letters, digits, and at most one period; any other characters will be translated to "`X`". If UNTRANSLATED, filenames will be sent and used literally. UNTRANSLATED is the default.

SET IBM

Syntax: `SET IBM ON` *or* `OFF`

SET IBM is really a predefined SET macro rather than a "hardwired" SET command; it can be redefined or undefined (see DEFINE); as distributed from Columbia, Kermit-20 defines IBM to be "parity mark, handshake XON, duplex half".

SET IBM should be used when running Kermit-20 in local mode, connected to an IBM or similar mainframe. If you have redefined the SET IBM macro, then your parameters will be used instead.

SET ITS-BINARY

Syntax: `SET ITS-BINARY ON` *or* `OFF`

Specify whether ITS-Binary file headers are to be recognized or ignored. By default, they are recognized. ITS binary format is a way (devised at MIT) of storing foreign 8-bit binary data on a 36-bit machine to allow automatic recognition of these files when sending them out again, so that you don't have to depend on the file byte size, or to issue explicit SET FILE BYTESIZE commands to Kermit.

An ITS format binary file contains the sixbit characters "DSK8" left-adjusted in the first 36-bit word. If ITS-BINARY is ON, then Kermit-20 will send any file starting with this "header word" using 8-bit input from the file even if the file bytesize is not 8, and will not send the header word itself. Kermit-20 will also store any incoming file that begins with that header word using 8-bit bytesize, again discarding the header word itself. If ITS-BINARY is OFF, then the header word, if any, will be sent or kept, and i/o will be according to the setting of FILE BYTESIZE.

This facility is provided for compatibility with the file formats used on certain public-access CP/M libraries.

SET INPUT

Syntax: `SET INPUT` *parameter value*

The INPUT command is used in TAKE command files or DEC-20 Batch control files as part of the login script facility, which is explained in greater detail later. SET INPUT controls the behavior of the INPUT command. The parameters are as follows:

SET INPUT DEFAULT-TIMEOUT *n*

> *n* is the number of seconds for an INPUT command to time out after not receiving the requested input, when no interval is explicitly given in the INPUT command. For instance, if the default timeout interval is 10 seconds, then the command
>
> ```
> INPUT login:
> ```
>
> will look for the "`login:`" prompt for 10 seconds. The default may be overriden by including an explicit interval in the INPUT command:
>
> ```
> INPUT 15 login:
> ```
>
> The default timeout interval is 5 seconds.

SET INPUT TIMEOUT-ACTION PROCEED *or* QUIT

> If the INPUT command comes from a Kermit-20 command file (see TAKE command) or a TOPS-20 Batch control file, then use this command to specify whether processing of the command file should proceed or quit after a timeout occurs. For TAKE files, the current command file is terminated and control returns to the invoking level (Kermit-20 prompt level, or a superior TAKE file). The default action is PROCEED.

SET INPUT CASE IGNORE *or* OBSERVE

> Specify whether alphabetic case should be ignored ("a" matches "A") or observed ("a" does not match "A") when scanning the input for the specified search string. By default, aphabetic case is ignored.

SET INPUT commands are "global"; the settings are not "pushed" and "popped" when entering or leaving TAKE command files.

SET LINE

Syntax: `SET LINE [`*octal-number*`]`

Specify the octal TTY number to use for file transfer or CONNECT. If you issue this command, you will be running Kermit-20 *locally*, and you must log in to the remote system and run Kermit on that side in order to transfer a file. If you don't issue this command, Kermit-20 assumes it is running *remotely*, and does file transfer over its job's controlling terminal line. You can also select the line directly in the CONNECT command; the command

```
CONNECT 12
```

is equivalent to

```
SET LINE 12
CONNECT
```

If you type SET LINE with no number argument, you will deassign any previous assigned line and revert to remote mode.

The SHOW LINE command will display the currently selected communication line and its charactistics, including parity, duplex, handshake, flow control, the speed if known, whether carrier is present (if it is a modem-controlled line), and whether Kermit-20 is in local or remote mode.

SET RECEIVE

In addition to the full complement of SET RECEIVE commands described in the main part of the Kermit User Guide, you may also SET RECEIVE SERVER-TIMEOUT to a value between 0 and 94. This specifies the number of seconds between timeouts during server command wait, 0 specifies that no timeouts should occur during server command wait. When a Kermit server times out, it sends a NAK packet. Some systems cannot clear piled-up NAKs from their input buffers; if you're using such a system to communicate with a Kermit-20 server, and you expect to be leaving the server idle for long periods of time, you should use this command to turn off server command-wait timeouts.

SET SPEED

Syntax: `SET SPEED` *n*

Set the baud rate of the currently selected communication to *n*, the decimal baud rate, for instance 300, 1200, 4800. When operating in local mode, it may be necessary to issue this command in order to enable BREAK simulation.

SET TVT-BINARY

Syntax: `SET TVT-BINARY ON` *or* `OFF`

Only for users running Kermit-20 on an ARPANET DEC-20, signed on to an ARPANET virtual terminal (TVT) from another host or through an ARPANET TAC. SET TVT ON causes Kermit-20 to negotiate binary mode (8-bit) communication with the ARPANET during file transfer. Without this command, file transfer through a TVT would not work in most cases.

TVT-BINARY is OFF by default. If you normally use Kermit-20 through the ARPAnet, you can put the command SET TVT-BINARY ON into your `KERMIT.INI` file.

CAUTION: This facility requires certain features in the Release 5 TOPS-20 ARPANET monitor, which may not be present in releases distributed by DEC. See the Kermit-20 source code for details.

The DEFINE Command

Syntax: `DEFINE` *macroname* `[`*set-option* `[,` *set-option* `[...]]]`

The DEFINE command is available in Kermit-20 for building "macros" of SET commands. The macro name can be any keyword-style character string, and the set options are anything you would type after SET in a SET command; several set options may be strung together, separated by commas. Example:

```
define notimeout send timeout 0, receive timeout 0, receive server 0
```

Macro definitions may not include macro names. You can list all your macros and their definitions with the SHOW MACROS command. You can list a particular macro definition with HELP SET *macroname*.

The SHOW Command

Syntax: `SHOW` [*option*]

The SHOW command displays various information:

DAYTIME	Current date, time, phase of moon.
DEBUGGING	Debugging mode in effect, if any.
FILE-INFO	Byte size for DEC-20 file i/o, incomplete file disposition.
INPUT	INPUT command parameters.
LINE	TTY line, parity, duplex, flow control, handshake, escape character, speed (if known), and session loggin information. Note that before release 6.0 of TOPS-20, the DEC-20 does not keep a record of the actual baud rate of a modem-controlled or "remote" TTY line.
MACROS	Definitions for SET macros.
PACKET-INFO	For incoming and outbound packets. Items under RECEIVE column show parameters for packets Kermit-20 expects to receive, under SEND shows parameters for outgoing packets.
TIMING-INFO	Delays, retries, server NAK intervals.
VERSION	Program version of Kermit-20. This is also displayed when Kermit-20 is initially started.
ALL	(default) All of the above.

12.5.6. Program Management Commands

The TAKE Command

Syntax: `TAKE` *filespec*

Execute Kermit-20 commands from the specified file. The file may contain contain any valid Kermit-20 commands, including other TAKE commands; command files may be nested up to a depth of 20. Default file type for the command file is `.CMD`. Most commands may have trailing comments, beginning by semicolon, but these should be avoided in REMOTE commands, GET commands, and the passwords that are prompted for after CWD and REMOTE CWD commands.

The ECHO Command

Syntax: `ECHO` *line of text*

The line of text is echoed at the terminal. This is useful when issued from within TAKE command files, to report progress or issue instructions.

The HELP Command

Syntax: `HELP [`*topic* `[`*subtopic*`]]`

Typing HELP alone prints a brief summary of Kermit-20 and its commands. You can also type

`HELP` *command*

for any Kermit-20 command, e.g. "help send" or "help set parity" to get more detailed information about a specific command. Type

`HELP ?`

to see a list of the available help commands.

The EXIT and QUIT Commands

Syntax: `EXIT`

Exit from Kermit-20. You can CONTINUE the program from the TOPS-20 Exec, provided you haven't run another program on top of it. You can also exit from Kermit-20 by typing one or more control-C's, even if it's in the middle of transferring a file. Kermit-20 will always restore your terminal to its original condition, and you will be able to CONTINUE the program to get back to "`Kermit-20>`" command level with current settings intact.

`QUIT` is a synonym for EXIT.

The LOG Command

Syntax: `LOG` [*option* [*filespec*]]

Log the specified option to the specified file:

SESSION During CONNECT or execution of a login script, log all characters that appear on the screen to the specified file. During CONNECT, the session log can be temporarily turned off during the remote session by typing the escape character followed by Q (for Quit logging), and turned on again by typing the escape character followed by R (for Resume logging). Default log is `SESSION.LOG` in the current directory.

TRANSACTIONS During file transfer, log the progress of each file. The DEC-20 transaction log file looks like this:

```
Kermit-20 Transaction Log File, Monday 27-Feb-1984

18:40:13: Opened Log: PS:<TIMREK>SAMPLE.LOG.1
18:40:31: -- Send Begins --
   8th bit prefixing: Off
   Block check type: 1
18:40:31: Opened File: PS:<SY.FDC>LOGIN.CMD.6
   Sending As "LOGIN.CMD"
   Sent: 547 7-bit bytes
18:40:34: Closed PS:<SY.FDC>LOGIN.CMD.6
```

```
18:40:34: Send Complete
18:40:50: -- Receive Begins --
   8th bit prefixing: Off
   Block check type: 1
18:40:50: Opened: PS:<TIMREK>AUTOEXEC.BAT.1
   Written: 186 7-bit bytes
18:40:51: Closed: PS:<TIMREK>AUTOEXEC.BAT.1
18:40:56: Closed Transaction Log
```

Transaction logging is recommended for long or unattended file transfers, so that you don't have to watch the screen. The log may be inspected after the transfer is complete to see what files were transferred and what errors may have occurred. Default log is `TRANSACTION.LOG` in the current directory.

DEBUGGING Log STATES or PACKETS, as specified in the most recent SET DEBUGGING command, to the specified file. If log file not specified, then use TTY if local, or `DEBUGGING.LOG` in the current directory if remote. If no SET DEBUGGING command was previously issued, log STATES to the specified file. Also allow specification of bytesize for the log file, 7 (normal, default), or 8 (for debugging binary transfers when the parity bit is being used for data), for instance

```
LOG DEBUGGING BINARY.LOG 8
```

A 7-bit log file can be typed, printed, or examined with a text editor or searching program. An 8-bit log file can only be examined with a system utility like FILDDT. When logging packets, each packet is preceded by a timestamp, the current timeout interval (preceded by a slash), and "`R:`" or "`S:`" to indicate data being received and sent, respectively. Packet format is described in the *Kermit Protocol Manual*.

SESSION is the default option. Thus the command "LOG" alone will cause CONNECT sessions to be logged in `SESSION.LOG` in the current directory. Any log files are closed when you EXIT or QUIT from Kermit, and are reactivated if you CONTINUE the program. You may explicitly close a log file and terminate logging with the CLOSE command.

The CLOSE Command

Syntax: `CLOSE` *option*

Close the specified log file, SESSION, TRANSACTION, or DEBUGGING, and terminate logging to that file.

12.6. Login Scripts: The INPUT, OUTPUT, CLEAR, and PAUSE Commands

When running Kermit-20 in local mode, connecting from the DEC-20 to another system via an external TTY line (for instance, through an autodialer), you may use the Kermit-20 INPUT, OUTPUT, CLEAR, and PAUSE commands to carry on a dialog with the remote system. When combined into a "script" in a Kermit-20 TAKE command file, or included in a Batch control file, these commands provide the ability to initially connect and log in to a remote system, and to set it up for file transfer. During script execution, session logging may be used to record the dialog.

The CLEAR Command

Syntax: `CLEAR`

Clear the input and output buffers of the currently selected line, and attempt to clear any XOFF deadlock.

The PAUSE Command

Syntax: `PAUSE` [*interval*]

Pause the specified number of seconds before executing the next command. The default interval is one second.

The INPUT Command

Syntax: `INPUT` [*interval*] [*string*]

On the currently selected communication line, look for the given string for the specified interval of time, which is specified in seconds. If no interval is specified, then wait for the default interval, which may be specified by SET INPUT DEFAULT-TIMEOUT, and is normally 5 seconds. Specifying an interval of 0 (or less) means no timeout -- wait forever for the specified string. An INPUT command can by interrupted by typing one or more Control-C's, which will return you to `Kermit-20>` prompt level.

Characters coming in from the line will be scanned for the search string, and when a match is found, the command will terminate successfully; if the string is not found within the given interval, the command will terminate unsuccessfully. While the INPUT command is active, all incoming characters will appear on your screen.

The search string may contain any printable characters. Control or other special characters that you could not normally type as part of a command may be included by preceding their octal ASCII values with a backslash, for instance `foo\15` is "foo" followed by a carriage return (ASCII 15, octal). A backslash alone will be taken as is, unless it is followed by an octal digit (0-7); if you want to actually specify a backslash in this context, double the backslash (`\\5` will be taken as `\5`).

The behavior of the INPUT command is governed by the SET INPUT CASE, SET INPUT DEFAULT-TIMEOUT, and SET INPUT TIMEOUT-ACTION commands, as described in the Kermit Commands section of the User Guide, or in the Kermit book.

In addition to normal use, Kermit-20 scripts can also be used in DEC-20 batch control files. Failure to match an input string in the timeout interval will result in a message starting with ''`?`'', which signals the Batch controller to detect an error. If INPUT TIMEOUT-ACTION is SET to PROCEED, any timeout error messages will be issued starting with a "`%`", which does not signal an error to Batch.

In addition to otherwise untypable control characters (like Control-C), certain printable characters in the search string may need to be "quoted" using the backslash mechanism:

`@` (ASCII 100) If it is the first character in the string, atsign tells TOPS-20 that the following characters will be the name of an indirect command file, for instance

```
input 10 @foo.txt
```

tells Kermit to spend 10 seconds scanning the communication line input for the string which is contained in the file `FOO.TXT`. If you need to specify a string that starts with "`@`", use `\100` instead.

`?` (ASCII 77) A question mark tells TOPS-20 to provide a brief help message about this part of the command; use `\77` instead.

`!` (ASCII 41) If it is the first character in the string, an exclamation point will cause TOPS-20 to ignore the rest of the string, i.e. treat it as a comment, use `\41`.

`;` (ASCII 73) Same as exclamation mark, use `\73`.

`(` (ASCII 50) In first position, TOPS-20 will think this marks the beginning of a "guide word"; use `\50`.

`-` (ASCII 55) In *final* position, a dash tells TOPS-20 that the command line is to be continued, concatenated with the following line. Use `\55` instead of a final dash. For instance, to specify the string "`More?--`", use "`More\77-\55`".

The OUTPUT Command

Syntax: `OUTPUT` *string*

The given string is sent out the currently selected communication line. The string is in the same form as the INPUT string; control or special characters may be included by prefacing their octal ASCII value with a backslash. Note that any terminating carriage return must be included explicitly as `\15`. The string will also be echoed at your terminal.

Login Script Hints

It is not a good idea to store passwords in plain text in a file. The facilities of the TOPS-20 command parser make this unnecessary, so long as you are sitting at your terminal. Suppose you have a script that looks for the string "`Password: `" and then outputs your password using a command like

```
out mypassword\15
```

If you change this line to

```
out @tty:
```

you may enter the password from your terminal as follows:

```
login: myuserid
Password: mypassword\15^Z
```

That is, you type the password, a backslash-encoded carriage return, and then Control-Z. This may be done even when executing commands from a TAKE file; after the `^Z`, control returns to the TAKE file. In the OUTPUT command, "`@TTY:` designates `TTY:` (your job's controlling terminal) to be an indirect command file; the `^Z` is the "end of file" for a terminal. This same technique could have been used in the first script example to allow you to supply from the terminal the name of the file to be sent. It might be a good idea to for you to include an ECHO command in your script file to remind you to do this, for instance:

```
input password:
echo ^GType your password, followed by "\15" and then a CTRL-Z
output @tty:
```

The `^G` is a Control-G, which should get your attention by sounding a beep at your terminal.

One might expect to be able to use the same indirect file mechanism with the OUTPUT command to provide a crude "raw upload" facility, as in

```
output @foo.bar
```

to send the contents of the file `FOO.BAR` to the remote system, with *no* synchronization or error checking. Unfortunately, there are two problems with this approach: first, TOPS-20 converts all carriage return / linefeeds in an indirect command file to spaces, and second, only very short files may be treated this way, because they must fit within TOPS-20's command "atom" buffer. The Kermit-20 TRANSMIT command provides a synchronized raw uploading of files.

12.7. Raw Download and Upload

"Raw Download" is the term commonly used to describe the capture of a remote file on the local system, without any kind of error detection or correction. This allows you to obtain files from remote systems that do not have Kermit, but with the risk of loss or corruption of data.

Kermit-20 provides raw downloading via the LOG SESSION command during CONNECT to a remote system. The session log is described above. To use session logging to capture a file:

1. Run Kermit on the DEC-20.

2. SET LINE to the TTY number through which you will be connected to the remote system.

3. Perform any required SET commands to condition Kermit for communication with the remote system. You may need SET PARITY, SET DUPLEX, SET FLOW, SET HANDSHAKE, etc., depending on the characteristics of the remote system and the communication medium.

4. CONNECT to the remote system and log in.

5. Condition your job on the remote system not to pause at the end of a screenful of text, and give whatever commands may be necessary to achieve a "clean" terminal listing -- for instance, disable messages from the system or other users.

6. Type the appropriate command to have the desired file displayed at the terminal, *but do not type the terminating carriage return*. On most systems, the command would be "type", on Unix it's "cat".

7. Escape back to Kermit to the DEC-20 and give the LOG SESSION command.

8. CONNECT back to the remote system and type a carriage return. The file will be displayed on your screen and recorded in the session log file.

9. Escape back to Kermit on the DEC-20 and give the CLOSE SESSION command.

The file will be in `SESSION.LOG` in your connected directory, unless you gave another name for it in your LOG SESSION command. You will probably find that some editing necessary to remove extraneous prompts, messages, padding characters, or terminal escape sequences, or to fill in lost or garbled characters. Here's an example showing how to capture a file `foo.bar` from a remote Unix system:

```
@kermit
Kermit-20>set line 23
Kermit-20>connect
[KERMIT-20: Connecting to remote host over TTY23:,
 type <CTRL-\>C to return.]
4.2 BSD UNIX

login: myuserid
Password: mypassword
% cat foo.bar^\C
[KERMIT-20: Connection Closed]
Kermit-20>log session foo.bar
Kermit-20>connect
[KERMIT-20: Connecting to remote host over TTY23:,
 type <CTRL-\>C to return.]
[KERMIT-20: Logging to File FOO.BAR.1]
(Type carriage return now.)
This is the file foo.bar.
It has three lines.
This is the last line.
% ^\
[KERMIT-20: Closing Log File FOO.BAR.1>
[KERMIT-20: Connection Closed]
```

```
Kermit-20>close session
```

Note that in this case, the Unix "`% `" prompt at the end of the text will have to be edited out.

Raw Upload

"Raw Upload" means sending a file from the local system to a remote one, again without error detection or correction. This allows you to send files from the DEC-20 to remote systems that don't have Kermit. Kermit-20 provides the TRANSMIT command for this purpose.

Syntax: `TRANSMIT` *filespec* [*prompt*]

For use in local mode only. Sends the specified text file a line at a time, "raw" (as is, *without* using Kermit protocol), to the remote system, waiting for the specified prompt for each line. Only a single file may be sent with the TRANSMIT command; wildcards are not allowed in the filespec. The file should be a text file, not a binary file. Since protocol is not being used, no assurance can be given that the file will arrive at the destination correctly or completely.

The *prompt* is any string, for instance the prompt of a line editor in text insertion mode. The prompt string may include special characters by preceding their octal ASCII values with a backslash, e.g. `\12` for linefeed, `\21` for XON (`^Q`). The syntax of the prompt string is explained in greater detail above, with the INPUT command.

If a prompt string is supplied, alphabetic case will be ignored in searching for it unless you SET INPUT CASE OBSERVE. If a prompt string is not supplied, then linefeed will be used by default, unless you have performed a SET HANDSHAKE command, in which case the current handshake character will be used. If you really want to send the entire file without waiting for any prompts, specify a prompt of "\0" (ASCII zero, null) (this is not advised).

The file will be sent using the current settings for duplex, parity, and flow control. There are no timeouts on input, as there are with the INPUT command. The TRANSMIT command waits forever for the prompt to appear. However, if you observe that the transfer is stuck, there are three things you can do:

- Type a Carriage Return to transmit the next line.
- Type a Control-P to retransmit the line that was just transmitted.
- Type two Control-C's to cancel the TRANSMIT command and get back to `Kermit-20>` command level.

TRANSMIT should be used as follows: CONNECT to the remote system, login, and start up some kind of process on the remote system to store input from the terminal into a file. On a DEC-20 (that doesn't have Kermit), you could do

```
copy tty: foo.bar
```

or you could start a line editor like EDIT or OTTO and put it into text insertion mode. On a Unix system, you could

```
cat /dev/tty > foo.bar
```

or you could run "ed" and give it the "a" command.

The Kermit-20 TRANSMIT command will send the first line of the file immediately. Then it will wait for a "prompt" from the remote system before sending the next line. When performing a copy operation from the terminal to a file, the "prompt" will probably be a linefeed, "`\12`" which is the default prompt -- most full duplex systems expect you to type a line of text terminated by a carriage return; they echo the characters you type and then output a linefeed. Half duplex systems, on the other hand, use some kind of line turnaround handshake character, like XON (Control-Q), to let you know when they are ready for the next line of input. Line editors like EDIT and OTTO prompt you with a line number followed by a tab; in that case your prompt character would be "`\11`" (be careful -- if the remote DEC-20 doesn't think your terminal has hardware tabs, it will simulate them by outputting spaces). In any case, to assure synchronization, it is your responsibility to set up the target system to accept line-at-a-time textual input and to determine what the system's prompt will be when it is ready for the next line.

Each line is sent with a terminating carriage return; linefeeds are not sent, since these are supplied by the receiving system if it needs them. The TRANSMIT command continues to send all the lines of the file in this manner until it reaches the end, or until you interrupt the operation by typing Control-C's.

If you cannot make the TRANSMIT command work automatically, for instance because the remote system's prompt changes for each line, you may TRANSMIT manually by specifying a prompt string that will not appear, and then typing a carriage return at your keyboard for each line you want to send.

If the TRANSMIT command completes successfully (you'll get a message to the effect that the transmission is complete), then you must connect back to the remote system and type whatever command it needs in orderto save and/or close the file there.

12.8. Kermit-20 Examples

Here are a few examples of the use of Kermit-20. Text entered by the user is underlined.

Remote Operation

The following example shows use of Kermit-20 as a server from an IBM PC. In this example, the user runs Kermit on the PC, connects to the DEC-20, and starts Kermit-20 in server mode. From that point on, the user need never connect to the DEC-20 again. In this example, the user gets a file from the DEC-20, works on it locally at the PC, and then sends the results back to the DEC-20. Note that the user can leave and restart Kermit on the PC as often as desired.

```
A>Kermit
Kermit-MS>connect
@
@Kermit
TOPS-20 Kermit version 4.2(262)

Kermit-20>server

 Kermit Server running on DEC-20 host.  Please type your escape
 sequence to return to your local machine.  Shut down the server by
 typing the Kermit BYE command on your local machine.
^[C
Kermit-MS>get foo.txt
```

The transfer takes place.

```
Kermit-MS>exit
A>
A>edit foo.txt ; (or whatever...)
A>
A>Kermit
Kermit-MS>send foo.txt
```

The transfer takes place.

```
Kermit-MS>bye
A>
```

The next example shows the special procedure you would have to use in order to send a mixture of text and binary files from a PC (or an 8-bit-byte system) to the DEC-20. Note that in this case, it's more convenient to avoid server mode.

```
Kermit-MS>connect
@
@Kermit
TOPS-20 Kermit version 4.2(262)

Kermit-20>receive
^]C
Kermit-MS>send *.txt
```

Textual files are sent.

```
Kermit-MS>connect
Kermit-20>set file bytesize 8
Kermit-20>receive
^]C
Kermit-MS>send *.exe
```

Binary files are sent.

```
Kermit-MS>connect
Kermit-20>exit
@logout
^]C
Kermit-86>exit
A>
```

Local Operation

In this example, a program DIAL is used to direct an autodialer to call another computer (a DECsystem-10); once the connection is made, DIAL starts Kermit with an implicit CONNECT command for the appropriate communication line. DIAL is not part of Kermit; if your system has an autodialer, there will be some site-specific procedure for using it.

```
@dial
Dial>dial stevens
STEVENS, 1-(201) 555-1234, baud:1200
[confirm]
Dialing your number, please hold...
Your party is waiting on TTY11:.
@
@Kermit
TOPS-20 Kermit version 4.2(262)

Kermit-20>connect 11
[Kermit-20: Connecting over TTY11:, type <CTRL-\>C to return]

CONNECTING TO HOST SYSTEM.
Stevens T/S 7.01A(10) 20:20:04 TTY41 system 1282
Connected to Node DN87S1(101) Line # 57

Please LOGIN or ATTACH

.log 10,35
JOB 51 Stevens T/S 7.01A(10) TTY41
Password:
20:20   15-Dec-83       Thur

.r new:Kermit

TOPS-10 Kermit version 2(106)

Kermit-10>server

[Kermit Server running on the DEC host.  Please type your escape
 sequence to return to your local machine.  Shut down the server by
 typing the Kermit BYE command on your local machine.]
```

```
^YC

[Kermit-20: Connection Closed.  Back at DEC-20.]

Kermit-20>set file bytesize 8
Kermit-20>get setdtr.cmd
^A for status report, ^X to cancel file, ^Z to cancel batch.
SETDTR.CMD.7 ^A
 Receiving SETDTR.CMD.7, file bytesize 8
 (repeated character compression)
 At page 1
 Files: 0, packets: 1, chars: 66
 NAKs: 0, timeouts: 0
.[OK]
Kermit-20>bye
Job 51  User F DA CRUZ [10,35]
Logged-off TTY41  at 20:22:58  on 15-Dec-83
Runtime: 0:00:01, KCS:33, Connect time: 0:02:39
Disk Reads:72, Writes:4, Blocks saved:160
....
Hangup? y
Click. Call duration was 193 seconds to area 201.
Dial>exit
```

Note the use of Control-A to get a status report during the transfer.

12.9. Installation of Kermit-20

Kermit-20 is built from a single MACRO-20 source file, `K20MIT.MAC`. It requires the standard DEC-distributed tools MONSYM, MACSYM, and CMD; the following files should be in `SYS:` -- `MONSYM.UNV`, `MACSYM.UNV`, `MACREL.REL`, `CMD.UNV`, and `CMD.REL`. The CMD package is also included with the Kermit distribution as `K20CMD.*`, in case you can't find it on your system.

The program should work on all TOPS-20 systems as distributed, but many customizations are possible. The site manager may wish to change various default parameters on a site-wide basis; this may be done simply by changing the definitions of the desired symbols, under "subttl Definitions", and reassembling.

The most notable site dependency is the definition of "SET IBM". As distributed from Columbia, Kermit-20 defines "SET IBM" in a built-in SET macro definition as "parity mark, duplex half, handshake xon". This definition may be found at MACTAB+1, near the end of the impure data section. It may be changed or deleted, and the program reassembled.

Sites that do not have ARPANET may wish to delete the TVT-BINARY entries from SET command tables, SETABL and SETHLP.

13. PDP-11 Kermit

Author:	Brian Nelson, University of Toledo, Ohio
Documentation:	Brian Nelson
Language:	Macro-11
Version:	3.58
Date:	September, 1987
Systems Supported:	RSTS/E, RSX-11M/M+, P/OS, Micro-RSX, RT-11 and TSX+

Kermit-11 Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
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
LONG Packet protocol support:	Yes
Sliding Windows protocol support:	No
Terminal emulation:	Yes
Communication settings:	Yes
Transmit BREAK:	Yes (depends on system)
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
File attributes packets:	Yes
Command macros:	No
Raw file transmit:	Yes

13.1. File Systems on the PDP-11

13.1.1. File Specifications

The general format of a file name is:

```
NODE::DEVICE:[DIRECTORY]NAME.TYPE;VERSION
```

'Node' refers to the DECNET node name, for example, FUBAR::, if applicable. 'Device', if present, refers to the physical device or logical name where the file resides.

For RSTS/E, 'device' can be a physical device, such as DB0: or DU1:, or it can be a user or system logical name which may include both a physical device name and a directory name. If the device name is a logical name, is it composed of 1 to 9 alphanumeric characters, including '$', as in DISK$ONE:, LB: and so on. For instance, the DCL system command

```
    $ ASS/SYS DB1:[200,210] SRC$DIR
```

would associate both the device DB1: and directory [200,210] with SRC$DIR:. Explicitly given directories override directory names imbedded in a logical name. Names longer than nine characters are truncated by the executive.

In the case of RSX-11M/M+ and RT-11, the device name can be either a physical name, such as DU0:, or a logical name which will translate to a physical device name, such as LB:.

On RSTS/E and RSX-11M/M+, the [directory] is a UIC (user identification code) or PPN (project,programmer) number of the format [NNN,MMM]. All users are assigned a UIC (or PPN) when accounts are created, this is the number you give to LOGIN to log into the system. It is also your default UIC (or PPN). Micro-Rsx and P/OS may have directories in either UIC format or named directory format, such as [1,2] or [KERMIT]. For P/OS, the default directory is [USERFILES]. Directories are not used in RT-11.

The NAME field is the primary identifier for the file. The name can be one to nine characters for RSX-11M/M+ and P/OS, and one to six characters for RSTS/E, RT-11 and TSX+. The TYPE field is usually used to group files according to some convention. For example, XXX.FTN refers to a Fortran-77 source file, FOO.C to a 'C' source file, and K11POS.TSK refers to a task image.

The version field is applicable ONLY to RSX type systems. The default version is always the highest version number.

All systems mentioned support some sort of filename wildcarding, the flexibility of which varies by executive. All support the use of '*' to represent either a fully wildcarded NAME or TYPE. RSTS/E supports the use of '?' to match any single character, whereas the others use a '%' to match any single character. The RSTS/E Kermit server will translate '%' to '?' internally for the GET and REMOTE DIR commands (see the section on Kermit-11 server operation).

Examples of wildcarded filenames:

*.B2S	Match any file with a TYPE of B2S.
K11%%%.MAC	match any file starting with K11, followed by one to three characters, with a TYPE of MAC.
K11???.MAC	Same as above, but for RSTS/E only.
XYZ.*;*	All versions of files with a NAME of XYZ with any TYPE (RSX-11M/M+ and P/OS only).

13.1.2. File Formats (Binary and Text)

13.1.2.1. RT-11 and TSX+

RT-11 treats all files as a contiguous stream of characters. There is no information stored in the directory to tell the system (or program) that a file is readable text (source program, runoff document,...) or consists of binary data (executable program, object file, `.SYS` file,...). An application program like Kermit-11 needs to know what type of file to expect, thus the presence of the SET FILE TYPE command (discussed later). The only real convention is that text files are streams of seven bit data with each record terminated by a carriage return/line feed character sequence and that binary files normally follow a filename TYPE convention. The TYPE (`.SAV`, `.SYS`, ...) is what Kermit-11 will look at to decide if a file should be sent as a text or binary file.

13.1.2.2. RSTS/E, P/OS and RSX-11M/M+

These systems can provide for a large number of file attributes for each file by using either FCS11 (RSX-11M/M+) or RMS11 (all). Text files are normally considered to be either STREAM format (FB$STM) or VARIABLE with implied carriage control (FB$VAR and FB$CR). RSTS/E has historically defaulted to STREAM, whereas the RSX based systems use VARIABLE. Kermit-11 follows those defaults when creating files unless told to do so otherwise by the presence of attribute data. The conversion of the internal data representation to one that can be transmitted to another Kermit is transparent for these types of files. Both the file attributes and the filename TYPE are examined by Kermit-11 to determine if a file needs to be sent as a text file (default) or a binary file. Additionally, on RSTS/E Kermit checks the file protection code, as one of the bits in it is used to flag an executable file (bit 6).

In all cases, unless (at this time) Kermit-11 is talking to another Kermit-11, or if Kermit-11 can't tell if a file is consists of binary data, the command SET FILE TYPE FIXED must be used to force Kermit to either send or get a non-text file correctly. When Kermit-11 is running in binary mode, all data is read from (or written to) the file without any translation or internal record control information. Any attribute information in the file's directory entry is ignored and the data read (or written) in 512 byte unformatted blocks. Thus it is indeed possible to transfer files like task images and object libraries. Since Kermit-11 supports a subset of a protocol feature called 'attributes', two Kermit-11's connected together can also correctly transfer files other than simple text and unformatted binary files, such as RMS indexed or relative files.

13.1.3. Saving Files on the PDP-11 From Your Microcomputer

You can send textual files to Kermit-11 without any special considerations as Kermit-11 defaults to creating normal text files. However, if you are sending a binary file (perhaps an `.EXE`) from say, your Rainbow under MS-DOS, you would need to tell Kermit-11 to expect binary data. This is done with the Kermit-11 command SET FILE TYPE FIXED. This will force Kermit-11 to write the data out exactly as it comes, in 512 byte unformatted records. Sending the same file back to the Rainbow would not require any special action since the file, as it sits on the PDP-11, has the proper information in the directory entry to tell Kermit-11 that the file is binary. As a note, for RT-11 you would need to use a filetype that is normally considered 'binary' like `.SAV` or `.OBJ` (see above notes for RT-11).

Never try to do a wildcarded send with mixed binary and text files with the file type set to FIXED. The result could be unusable as not all systems store text data in the same internal format. For example, if Kermit-11 is forced into binary mode (via SET FIL TYP FIX) and is requested to send a file with implied carriage control (normal for RSX text files), it will actually send, for each line, two bytes representing the record length, followed by the data and then followed by a ASCII NUL to pad the record to an even length. That is not incorrect, rather, it is EXACTLY how the data was stored on disk.

In general, avoid sending anything other than unformatted binary files and text file to unlike systems. For example, requesting a RMS indexed file from the PDP-11 to be sent to a PC would case Kermit-11 to send it as a binary file, but the file attributes would be lost. Sending such a file back to the PDP-11 would result in an unusable file unless you could reconstruct the attribute information.

13.1.4. Program Operation

Kermit-11's prompt is normally "Kermit-11>". This can be changed if need be via the SET PROMPT command. Invoking Kermit-11 is very site dependent.

13.1.4.1. RSTS/E

If Kermit-11 has a ccl definition, it would likely be invoked as "KER" or "KERMIT". If not, try "RUN $KERMIT", as this is a likely place where Kermit-11 may have been put. Otherwise consult your local support staff.

13.1.4.2. RSX-11M/M+

If Kermit-11 has been installed, it most likely will have a task name of `...KER` which means that typing "KER" should get things running. If not, consult your local support staff.

13.1.4.3. RT-11/TSX+

On version 5 of RT-11, programs can be run simply by typing the filename. Thus, if there is a file `SY:KERMIT.SAV`, simply type "KERMIT". If this fails, contact your local support staff for assistance.

13.1.4.4. P/OS

Kermit-11 is generally run from DCL on P/OS. The program is invoked via the DCL RUN command, as in RUN K11POS or RUN KERMIT, depending on what the task image name is.

Note that for the case where Kermit is installed (for RSTS/E and RSX-11M/M+) that Kermit-11 can get command line arguments, as in:

```
$ KER SERV                    Kermit starts as a server.
> KER send fubar.txt          Kermit sends the file.
```

Otherwise, the program is run interactively from the Kermit-11> prompt:

```
$ KERMIT
Kermit-11 V3.54
Kermit-11>SET BLO 3           Changes checksum type.
Kermit-11>SER                 Enter Kermit server.
```

Note that whenever Kermit-11 starts up, it will always try to find a file called `KERMIT.INI` in your current directory. This file can contain any valid Kermit command, though the usual use of this is to place various Kermit-11 SET commands in it. If this file does NOT exist, it will try to find it in `LB:[1,2]KERMIT.INI` (excluding RT-11). In addition to the `.INI` file, commands may be placed in a file and then executed via the Kermit-11 TAKE (or @) command.

13.2. Local and Remote Operation

Kermit-11 by default assumes that all file transfers will occur over the terminal line that you are currently logged in on (`TI:`, `TT:`, `KB:`). This is known as REMOTE mode (the PDP-11 is the remote system). This would be the desired case if you are running Kermit on a microcomputer such as a Rainbow and are currently logged into the PDP-11 through the micro. However, if you wanted to dial out, say by an autodial modem, from the PDP-11 to another system, you need to tell Kermit-11 to use some other terminal line. This would be called LOCAL mode (the PDP-11 is the local system). The line can be altered with the SET LINE command (see section on SET and

CONNECT). A SET LINE command is done implicitly if Kermit-11 finds itself running on a PRO/350, under either P/OS, RT-11 or TSX+.

Since support of parity varies by both interface type (DL11 vs DZ11) and by operating system, Kermit-11 makes NO attempt to find out what the current parity of it's line is. Kermit-11 generates it's own parity which is set with the SET PARITY command.

There are a couple of things to point out regarding Kermit-11 and LOCAL mode (you did a SET LINE command):

- The system manager may have lines other than your own protected (or owned by the system). On RSTS/E lines are often made unaccessible unless your account possesses the needed privilege(s). On RSX-11M/M+, privilege is required to alter settings on any other terminal line. You may have to talk to your system manager to get access to an outgoing terminal line.
- Once connected to a modem through another line, a means must exist for the connection to be broken (if the host you are calling won't do it). Given that your line has full or partial modem control (DZV11, DZ11, DH11, DHU/V11) the RSX, RT-11/TSX+ and RSTS/E Kermits have a HANGUP (or DISCONNECT) command, which instructs the system to disconnect the modem. Unless this is done, you never get disconnected and could run up a tidy phone bill.

Kermit-11 has, as of v3.53, a rudimentary command line editor. You can recall previous commands with the UP-Arrow key, and exit the command with the LEFT and RIGHT arrow keys. The RUBOUT key, of course, deletes characters, while the Control-R key retypes the line. Control-E moves to the end of the line and Control-H moves to the start of the line.

13.3. Kermit-11 Commands

Kermit-11 has the following commands available:

@	Synonym for TAKE
BYE	Logout a remote server
CONNECT	Connect to a remote system
COPY	Local copy of a file(s)
CWD	Set new working directory
DELETE	Local delete of a file(s)
DIAL	Have a connected modem dial a number
DIRECT	Local directory display
DISCONNECT	Hangup a remote line
DISPLAY	Internal debugging
ERASE	Local delete of a file(s)
EXIT	Exit to system
FINISH	Stop a remote server without logging out
GET	Get a file(s) from a remote server
HANGUP	Hangup a remote line
HOST	Execute system command locally (where applicable)
LOCAL	Force interpretation of command to the local system
LOGFILE	Create a log file
QUIT	Same as EXIT
PRINT	Print a file locally (where applicable)
RECEIVE	Receive a file(s) from a remote kermit
REMOTE	Prefix for file management commands to a server
RENAME	Local rename of filename(s)
SEND	Send a file(s) to a remote Kermit
SERVER	start a Kermit server
SET	Change Kermit parameters
SHOW	Display Kermit parameters
TAKE	Execute indirect command file

TYPE	Local display of file on terminal
WHO	Local display of logged in users (RSTS/E only)

13.4. Commands for File Transfer

Kermit-11 includes the standard repertoire of Kermit file transfer commands, including SEND, RECEIVE, and GET.

The SEND Command

Syntax: `SEND` *filespec*

The SEND command causes a file or file group to be sent from the PDP-11 to the other system. If filespec contains wildcard characters then all matching files will be sent, in alphabetical order (according to the ASCII collating sequence) by name. If filespec does not contain any wildcard characters, then the single file specified by filespec will be sent.

SEND Command General Operation

:

Files will be sent with their PDP-11 file name and type (for instance FOO.BAR). Each file will be sent according to the record type and attributes recorded in its file descriptor. Kermit-11 attempts to translate all formats of text file to a format usable on any system. Note that there is no need to set the FILE TYPE parameter for sending files, since Kermit-11 always uses the information from the file directory entry and the filetype (extension) to determine how to send the file.

If communication line parity is being used (see SET PARITY), Kermit-11 will request that the other Kermit use a special kind of prefix notation for binary files. This is an advanced feature, and not all Kermits have it; if the other Kermit does not agree to use this feature, binary files cannot be sent correctly. This includes executable programs (like `.EXE` files, CP/M `.COM` files), relocatable object modules (`.OBJ` files), as well as any text file containing characters with the eighth bit on.

Kermit-11 will also ask the other Kermit whether it can handle a special prefix encoding for repeated characters. If it can, then files with long strings of repeated characters will be transmitted very efficiently. Columnar data, highly indented text, and binary files are the major beneficiaries of this technique.

If you’re running Kermit-11 locally, for instance dialing out from a PDP-11 to another system using an autodialer, you should have already run Kermit on the remote system and issued either a RECEIVE or a SERVER command. Once you give Kermit-11 the SEND command, the name of each file will be displayed on your screen as the transfer begins. As the transfer continues, you will get a small display of the packet count along with the number of packets rejected. See the SET TERMINAL and SET UPDATE commands for more information. You may also type Control-X or Control-Z to interrupt the current file or file group. Control-E will also abort the transfer by sending an ’error’ packet to the other Kermit.

The RECEIVE command

Syntax: `RECEIVE [`*filespec*`]`

The RECEIVE command tells Kermit-11 to receive a file or file group from the other system. The name is taken from the incoming file header. If an incoming file has the same name as an existing file, Kermit-11 will by default create a new file. On RT-11 and RSTS/E, the old file will be deleted by the executive. On RSX-11M/M+ and P/OS, a new file with a higher version number will be created. To avoid files being superceded, see the SET FILE [NO]SUPERCEDE command.

Incoming files will all be stored with the prevailing file type, ASCII by default, which is appropriate for text files. If you are asking Kermit-11 to receive binary files from a microcomputer or other 8-bit system, you must first type SET FILE TYPE FIXED. Otherwise, an error may occur when receiving the file. Please note that this does NOT apply to two Kermit-11 programs connected to each other. In that case the sending Kermit-11 will tell the receiving Kermit-11 to switch to binary mode if need be.

If parity is being used on the communications line, then 8th-bit prefixing will be requested. If the other side cannot do this, binary files cannot be transferred correctly.

If you are running Kermit-11 locally, you should already have issued a SEND command to the remote Kermit, and then escaped back to Kermit-11. As files arrive, their names will be displayed on your screen.

If a file arrives that you don't really want, you can attempt to cancel it by typing Control-X; this sends a cancellation request to the remote Kermit. If the remote Kermit understands this request (not all implementations of Kermit support this feature), it will comply; otherwise it will continue to send. If a file group is being sent, you can request the entire group be cancelled by typing Control-Z.

Normally, one runs the remote Kermit as a SERVER, thus the RECEIVE command is never used, rather, the GET command, described next, is used.

The GET Command

Syntax: `GET [`*remote-filespec*`]`

The GET command requests a remote Kermit server to send the file or file group specified by *remote-filespec*. This command can be used only when Kermit-11 is local, with a Kermit server on the other end of the line specified by SET LINE. This means that you must have CONNECTed to the other system, logged in, run Kermit there, issued the SERVER command, and escaped back to the PDP-11.

The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. Any leading spaces before the remote filespec are stripped, and lower case characters are raised to upper case.

As files arrive, their names will be displayed on your screen. As in the RECEIVE command, Control-X (`^X`) to request that the current incoming file be ancelled, `^Z` to request that the entire incoming batch be cancelled.

If the remote Kermit is not capable of server functions, then you will probably get an error message back from it like "Illegal packet type". In this case, you must connect to the other Kermit, give a SEND command, escape back, and give a RECEIVE command.

13.4.1. Server Operation

The SERVER command puts a remote Kermit-11 in "server mode", so that it receives all further commands in packets from the local Kermit. The Kermit-11 server is capable (as of this writing) of executing the following remote server commands: SEND, GET, FINISH, BYE, REMOTE DIRECTORY, REMOTE CWD, REMOTE SPACE, REMOTE DELETE, REMOTE TYPE, REMOTE HELP, REMOTE COPY, REMOTE RENAME, REMOTE WHO, REMOTE LOGIN and REMOTE HOST.

Any nonstandard parameters should be selected with SET commands before putting Kermit-11 into server mode, in particular the file type. The Kermit-11 server can send all files in the correct manner automatically. As noted before, if a Kermit-11 is talking to another Kermit-11, they will negotiate any 'binary' parameters automatically. However, if this is NOT the case and you need to ask Kermit-11 to receive binary files you must issue the SET FILE TYPE FIX command before putting it into server mode, and then you must only send binary files. You cannot send a mixture of text files and 8-bit binary files to a Kermit-11 server unless the files are not for use on the PDP-11.

13.4.2. Commands for Servers

When running in local mode, Kermit-11 allows you to give a wide range of commands to a remote Kermit server, with no guarantee the that the remote server can process them, since they are all optional features of the protocol. Commands for servers include the standard SEND, GET, BYE, FINISH commands, as well as the REMOTE command.

The BYE Command

The BYE command tells a remote server to log out of the remote system. In addition, some remote systems will also disconnect the line for you. If this is not the case, the DISCONNECT command will (depending on your interface) cause the line to be dropped. See DISCONNECT.

The FINISH Command

The FINISH command tells the remote Kermit server to exit without logging out of the remote system. You can then CONNECT back to the Server operation system.

The REMOTE Command

Send the specified command to the remote server. If the server does not understand the command (all of these commands are optional features of the Kermit protocol), it will reply with a message like "Unknown Kermit server command". If does understand, it will send the results back, and they will be displayed on the screen. The REMOTE commands are:

REMOTE COPY *filespec newfilespec*
: Copy file. The server is asked to make a copy of the specified file. Both filespecs must be in the correct format for the remote system. Kermit-11 does not parse or validate the file specifications. Any leading spaces will be stripped and lower case characters converted to upper case. Note that this command simply provides for copying a file within the server's system - it does not cause a file to be transferred.

REMOTE CWD *directory*
: Change Working Directory. If no directory name is provided, the server will change to the default or home directory. Kermit-11 currently does not ask for a password.

REMOTE DELETE *filespec*
: Delete the specified file or files. The names of the files that are deleted will appear on your screen.

REMOTE DIRECTORY [*filespec*]
: The names of the files that match the given file specification will be displayed on your screen, perhaps along with size and date information for each file. If no file specification is given, all files from the current directory will be listed.

REMOTE HELP
: The remote server will send back a list of server commands that it can execute.

REMOTE HOST *command*
: Pass the given command to the server's host command processor, and display the resulting output on your screen. Not all Kermit servers can do this function. In the case of Kermit-11, only the RSTS/E Kermit-11 server can execute the REMOTE HOST command.

REMOTE LOGIN *user password*
: Ask a remote server to log into a different account or username. The support for this command is rarely implemented as many systems layer login/logout support over the executive. A Kermit-11 server can only support this on RSTS/E, and at that only for version 9.0 or later. Of the various DEC PDP-11 operating systems, only RSTS/E has the support for logging in and out built into the executive and accessible with directives.

REMOTE RENAME *oldfile newfile*
: Change the name on the specified file (or files). Both file specifications must be valid for the server's system.

REMOTE SPACE
: Display information about disk usage in the current directory.

REMOTE TYPE *filespec*
: Display the contents of the specified file on your screen.

REMOTE WHO
: Display current status of user's logged in.

13.5. Commands for Local File Management

These commands provide some local file management capability without having to leave the Kermit-11 program. These commands are very similar to the REMOTE commands in function and syntax. They are all executed locally, and are available when Kermit-11 is either local or remote. The arguments to these commands are the same as the arguments expected from the user Kermit when Kermit-11 is processing a command in server mode. Additionally, these commands can be prefixed by the LOCAL keyword.

COPY *filespec newfilespec*
CWD *directory*
DELETE *filespec*
DIRECTORY [*filespec*]
HELP
HOST *command*
RENAME *oldfile newfile*
SPACE
TYPE *filespec*
WHO

13.5.1. The CONNECT Command

The CONNECT command will allow you to connect in as a terminal over the line that was specified by the SET LINE command. (Using the CONNECT command before using the SET LINE command will result in an error message.) The terminal line must be one which is accessible to the user.

The distributed RSX-11M/M+ task has been built with the `/PR:0` switch to enable the task to change other terminal settings. Additionally, for RSX-11M/M+, the MCR command `SET /SLAVE=TT`*nn*`:` should be done before entering Kermit-11.

If you are running `K11POS.TSK` on a PRO/350, Kermit will set the line to `XK0:` and the speed to 9600 by default.

Please note that Kermit-11 CAN NOT change the speed of a DL11 type interface, nor can it change the speed of a PDT-150 modem port (use SPEED.SAV).

The following is an example of using a Racal-Vadic VA212 autodialing modem to log into a remote TOPS-20 system. There is one point at which there is no echoing of the user input, this is following the typing of the local 'escape sequence', which by default is Control-\ followed by a 'c'. The control-backslash informs the terminal emulator that the next character is a command. In this case, the command was 'C', which means to return to the local PDP-11 system. Control-\ followed by ? would print a help message. All the commands prior to the DIAL command were contained in the INI file, `KERMIT.INI`.

```
$ kermit
Kermit-11 V3.46 Last edit: 21-Feb-1986
Kermit-11>set modem vadic
Kermit-11>set pho num cu 9K12121234567
Kermit-11>set logfile 20.log
Kermit-11>set deb console
Kermit-11>set lin tt58:
Link: TT58: Speed: 9600, DTR not present
Kermit-11>set dtr
Kermit-11>set spe 1200
Kermit-11>dial cu
Using: 9K12121234567
Connection established, type CONNECT to access remote
Kermit-11>connect

enter class 4
class 004 start

CU20B
@log xx.abcdef
 CU20B, TOPS-20 Monitor 5.1(5101)-2
 Job 28, TTY32, 2-Apr-84 4:15:24PM
 Previous login was 2-Apr-84 4:10:16PM
        .
        .
@logout
[Confirm]
Logged out Job 28, User XX.ABCDEF , TTY 32,
  at  2-Apr-84 16:19:34,  Used 0:00:11 in 0:04:10

Kermit-11>disc
KERMIT link TT58: disconnected
Kermit-11>exit

$ logout
```

13.6. The SET Command

Syntax: `SET` *parameter keyword*

The SET command is used to set various parameters in Kermit. The format of the SET command is:

SET ATTRIBUTES

Syntax: `SET ATTRIBUTES {ON, OFF}`

Part of the Kermit protocol is the support of file attributes. Connected Kermits that support this can send information to each other about file size, time/date of creation, RMS file headers and other useful things. Due to potential problems with incompatible implementations this feature can be disabled. In this case, the sending Kermit-11 will never try to send file attributes, even though the receiver may have indicated that it supports this.

SET BAUD

This is the same as SET SPEED. See HELP SET SPEED

SET BINARY-TYPE

Kermit-11 has a default list of filetypes that are scanned to decide if a file should be sent in binary mode in addition to checking file attributes for RSX, P/OS and RSTS/E. The user can, however, override this list with the this command. The default list is fairly inclusive, with types such as `.SAV` and `.TSK` forcing Kermit-11 into binary transmission. See HELP SET FIL for the default list. Examples:

```
Kermit-11> set binary-type .sav
Kermit-11> set bin .exe
```

SET BLOCK-CHECK

Syntax: `SET BLOCK_CHECK {1, 2, 3}`

The SET BLOCKCHECK command is used to determine the block check sequence which will be used during transmission. The block check sequence is used to detect transmission errors. There are three types of block check available. These are the single character checksum (default), the two character checksum, and the three character CRC (cyclic redundancy check). This command does not ensure that the desired type of block check will be used, since both Kermit's involved in the transfer must agree on the block check type. Kermit-11 will request that the type of block check set by this command be used for a transfer. If the other Kermit has also had the same block check type requested, then the desired block check type will be used. Otherwise, the single character checksum will be used. The command should be given to BOTH Kermits since Kermit-11, when in server mode, has no say about what kind of checksum it wants to use. (See Kermit protocol manual for more information.)

SET CONSOLE

Syntax: `SET CONSOLE {7, 8}`

The SET CONSOLE command is used under P/OS to control the passing of 8 bit data to the terminal during the connect command. If you are getting multinational characters being printed, this is a very useful thing to set. The default is SET CON 7.

SET DEBUG

Syntax: `SET DEBUG {ALL, CONSOLE, CONNECT, FILE, PACKET, STATE}`

The SET DEBUG command is used to specify the type and level of debugging to a disk file . This disk file must have been created by the SET LOGFILE command.

SET DEBUG ALL

SET DEBUG ALL will turn on logging for CONSOLE,CONNECT,FILE,PACKET and STATE to the disk file specified by SET LOGFILE. This command is the same as SET DEBUG ON. The command format is:

SET DEBUG CONSOLE

SET DEBUG CONSOLE will turn on logging for all i/o during a remote connect to the disk file specified by SET LOGFILE. This command is the same as SET DEBUG CONNECT.

SET DEBUG CONNECT

SET DEBUG CONNECT will turn on logging for all i/o during a remote connect to the disk file specified by SET LOGFILE. This command is the same as SET DEBUG CONSOLE.

SET DEBUG FILE

SET DEBUG FILE will log all file 'opens' and 'creates' to the file specified by SET LOGFILE.

SET DEBUG HELP

SET DEBUG HELP gives the user a list of all qualifiers which can be used with SET DEBUG.

SET DEBUG NONE

SET DEBUG NONE 'turns off' all debugging. This is the same as the SET DEBUG OFF command.

SET DEBUG OFF

SET DEBUG OFF 'turns off' all debugging. This is the same as the SET DEBUG NONE command.

SET DEBUG ON

SET DEBUG ON will 'turn on' logging for CONSOLE,CONNECT,FILE,PACKET and STATE to the disk file specified by SET LOGFILE. This command is the same as SET DEBUG ALL.

SET DEBUG PACKET

SET DEBUG PACKET will 'turn on' logging of all receive and transmit packets to the disk file specified by SET LOGFILE.

SET DEBUG STATE

SET DEBUG STATE will turn on logging of all internal Kermit-11 state transitions.

SET DELAY

Syntax: `SET DELAY` *seconds*

The DELAY parameter is the number of seconds to wait before sending data after a SEND command is given. This is used when Kermit-11 is running in remote mode to allow the user time to escape back to the other Kermit and give a RECEIVE command.

SET DEFAULT

Syntax: `SET DEFAULT` *device*

The DEFAULT parameter allows you to specify a device and UIC (or PPN) for all subsequent file opens (for SENDING) and file creates (for RECEIVING). It is disabled by typing SET HOME. Example:

```
Kermit-11>set default db2:[200,201]
```

This is quite useful for Kermit-11 running on a DECNET link, as you can set the default for file operations to include node names and passwords as in:

```
Kermit-11>set def orion::sys$system:[fubar]
```

SET DIAL

Kermit-11 has knowledge built in to it of a number of the more common ’smart’ autodial modems. To find out if your modem is directly supported try the command SET MODEM ?. If your modem is not in this list then you need the SET DIAL command to generate the data base used by Kermit to control the modem. Kermit uses this information to implement the DIAL command. A command such as DIAL can only be done when Kermit knows both how to format commands to the modem, and what kind of text the modem will send back to it in response. As an example, the VADIC VA212PA modem is awakened from an idle state by the character sequence 05 015 (in octal), which is a Control-E followed by a carriage return. In response to this two-character string, the modem responds with:

```
HELLO: I’M READY
*
```

Thus Kermit has to know that when it sends the wakeup sequence it needs to wait for the asterisk to be sent back by the modem. At this point Kermit will know that the modem is in a state awaiting further commands, such as that to dial a phone number.

It is not possible for Kermit to have knowledge of all makes of modems. Instead Kermit supports a command, SET MODEM USER_DEFINED, which then allows you to use the SET DIAL command to inform Kermit how the modem works. Once Kermit knows how to control the modem, you can use the DIAL command to initiate a call from Kermit.

The SET DIAL commands are:

SET DIAL WAKEUP	Define the wakeup string
SET DIAL PROMPT	Define the prompt the modem uses
SET DIAL INITIATE	Define a string to start dialing
SET DIAL CONFIRM	Define the string to confirm number
SET DIAL FORMAT	Define the number formatting string
SET DIAL SUCCESS	Define string(s) for call complete
SET DIAL INFO	Define string(s) for informative text
SET DIAL FAILURE	Define string(s) for call failure
SET DIAL CONFIRM	Define string for number confirmation
SET DIAL WAKE_RATE	Set pause time between wakeup characters
SET DIAL DIAL_RATE	Set pause time between number digits
SET DIAL DIAL_PAUSE	Define string for dial tone pause

Suppose we had to tell Kermit about the Racal Vadic VA212PA modem (though in reality Kermit already knows about that kind). In checking the owners manual for it, we find that:

- To wake the modem up, we type a control E followed by a carriage return.
- To dial a number, we type the letter D followed by a carriage return. At this point, the modem prints a NUMBER? prompt, we then type the desired number in. It reprints the number and then waits for a carriage return from us to confirm that its really the correct phone number.
- When it completes dialing, it will print ’ON LINE’ or ’ONLINE’ for a successful call, otherwise it may display on the terminal ’BUSY’, ’FAILED CALL’, ’NO DIAL’, ’VOICE’ or ’TIME OUT’. While it is waiting for its call to be answered, it may print the line ’RINGING’ several times in order to tell you that it is working on it.

The Kermit commands required would be:

```
Kermit-11>SET MODEM USER_DEFINED
Kermit-11>SET DIAL WAKEUP \05\015
Kermit-11>SET DIAL PROMPT *
Kermit-11>SET DIAL INITIATE D\015
Kermit-11>SET DIAL FORMAT %P%S\015
```

```
Kermit-11>SET DIAL CONFIRM \015
Kermit-11>SET DIAL SUCCESS ONLINE
Kermit-11>SET DIAL SUCCESS ON LINE
Kermit-11>SET DIAL INFO RINGING
Kermit-11>SET DIAL FAILURE BUSY
Kermit-11>SET DIAL FAILURE FAILED CALL
Kermit-11>SET DIAL FAILURE NO DIAL
Kermit-11>SET DIAL FAILURE VOICE
Kermit-11>SET DIAL FAILURE TIME OUT
Kermit-11>SET DIAL DIAL_PAUSE 9K
Kermit-11>DIAL 14195551212
```

The notation "`\05\015`" indicates the Control E followed by a carriage return; 05 is octal for control E, 015 is octal for carriage return. An alternate notation for octal numbers can be used by placing the value inside of inequality characters, as in SET DIAL WAKE <05><015> though the former is preferred.

The notation "`%P%S\015`" indicates to Kermit that the phone number from the dial command is to be followed by a carriage return; the `%S` is simply a placeholder for the phone number. The presence of the `%P` is to indicate where to insert the dial pause string, in this case we need to dial 9 and wait for a second dial tone. The "K" is the Racal Vadic code to get the modem to pause. If you are dialing on a direct line, the DIAL_PAUSE command is unneeded. If for any reason you need to pass a "\" or "<" to your modem, simply prefix the character with another "\", as in "\\".

Many modems require only the WAKEUP, PROMPT, FORMAT and result strings. The Digital DF112 is an example of this; its definition would look like:

```
Kermit-11>SET MODEM USER_DEFINED
Kermit-11>SET DIAL WAKEUP \02
Kermit-11>SET DIAL PROMPT READY
Kermit-11>SET DIAL FORMAT %S#
Kermit-11>SET DIAL SUCCESS ATTACHED
Kermit-11>SET DIAL FAILURE BUSY
Kermit-11>SET DIAL FAILURE DISCONNECTED
Kermit-11>SET DIAL FAILURE ERROR
Kermit-11>SET DIAL FAILURE NO ANSWER
```

Some modems may be unable to accept data at the line speed; in this case we would need to use the SET DIAL WAKE_RATE and SET DIAL DIAL_RATE. These two commands accept a delay time in milliseconds; the actual delay will not be precise as the PDP-11 line clock interrupts sixty times per second. Furthermore, on RSTS/E the finest granularity for timing is one second; thus setting delays would result in delays of one second increments.

In general, not all of the result fields need be specified except for the call completed strings; Kermit will time out after a while if it can't match a response with any definitions.

Further information can be found in the sections on SET MODEM, DIAL, REDIAL and SET PHONE.

SET DTR

The SET DTR command is very similar to the DISCONNECT (or HANGUP) command. SET DTR, where supported, raises DTR for a predetermined amount of time, whereas the DISCONNECT (or HANGUP) command drops DTR. The SET DTR is only functional on RSTS/E, which by default keeps DTR low until either RING INDICATOR or CARRIER DETECT goes high. This is opposite of the behavior on RT11 and RSX11M/M+, both of which normally assert DTR. The SET DTR command raises DTR for at least 30 seconds (depending on the version of RSTS/E) and is useful for making connections to front end switches (such as MICOM and GANDALF). On RT11, SET DTR is identical to the HANGUP command; it simply drops DTR for two seconds. In this case (RT11 and TSX+) this command is only supported on RT11 5.2 and TSX+ 6.0 with the XL/XC and CL drivers, respectively. This command is a no-op on RSX11M/M+ and P/OS. For further information on modem support, see the later section regarding such.

SET DUPLEX

Syntax: `SET DUPLEX {FULL, HALF}`

The DUPLEX parameter controls whether an outgoing link (set via the SET LINE command) is a full duplex link (the default) or a half duplex link. All it does for half duplex is to cause all characters typed after the CONNECT command to be echoed locally.

SET END-OF-LINE

Syntax: `SET END-OF-LINE <octal ASCII value>`

The END-OF-LINE parameter sets the ASCII character which will be used as a line terminator for all packets SENT to the other KERMIT. This is normally not needed for most versions of KERMIT.

SET ESCAPE

Syntax `SET ESCAPE` <octal ASCII value>)

This command will set the escape character for the CONNECT processing. The command will take the octal value of the character to use as the escape character. This is the character which is used to "escape" back to Kermit-11 after using the CONNECT command. It defaults to control (octal 34). It is usually a good idea to set this character to something which is not used (or at least not used very much) on the system being to which Kermit-11 is CONNECTing.

SET FILE

Syntax: `SET FILE {NOSUPERCEDE, SUPERCEDE, TYPE` *file-type*}

The SET FILE command allows you to set various file related parameters.

SET FILE TYPE ASCII

File type ASCII is for text files. SET FILE TYPE TEXT is the same.

SET FILE TYPE AUTO

Kermit-11 will normally try to decide if a file must be sent in binary mode based on the file attributes and filetype. If, for instance, the directory entry for `FUBAR.TXT` showed it to be RMS (or FCS) fixed length records, Kermit-11 will switch to binary mode and send it verbatim. If the receiving Kermit is Kermit-11, then the sending Kermit will send attribute data over also. The file types shown in Table 13-1 also will normally be sent as binary files unless you use the SET FILE TYPE NOAUTO command.

SET FILE TYPE BINARY

File type BINARY is for non-text files. Note that binary files which are generated on a PDP-11 system cannot be transferred to another (non PDP-11) system without losing file attributes. This means that (for example), an RSM11 indexed file cannot be transmitted with Kermit-11 at this time. You can not have parity set to anything but NONE

`*.TSK`	RSX, IAS, and RSTS tasks
`*.SAV`	RT11 and RSTS save images
`*.OBJ`	Compiler and macro-11 output
`*.STB`	TKB and LINK symbol tables
`*.CRF`	TKB and LINK cross reference files
`*.TSD`	'Time shared DIBOL' for RT11
`*.BAC`	RSTS Basic-plus 'compiled' files
`*.OLB`	RSX, IAS, and RSTS object libraries
`*.MLB`	RSX, IAS, and RSTS macro libraries
`*.RTS`	RSTS/E run time systems
`*.EXE`	VMS executable

Table 13-1: Kermit-11 File Types

to use binary file transfer (see HELP SET PARITY) unless the other Kermit can process eight bit quoting. Two Kermit-11's connected to each other will use binary transmission automatically via the Kermit attribute packets, preserving file attributes where it makes sense (i.e. RSTS/E and RSX only).

SET FILE TYPE DECMULTINATIONAL

PDP-11 Kermit normally strips the high bit of every character on both transmission and reception of files (unless the SET FILE TYPE FIXED command was given). The SET FIL DEC command will cause Kermit-11 to leave all data intact but still obey the host file system when reading or writing files. In other words, Kermit will write sequential implied carriage control files with eight bit data if this command is used.

SET FILE TYPE FIXED

This is the same as SET FILE TYPE BINARY.

SET FILE TYPE NOAUTO

SET FILE NOAUTO disables Kermit-11 from trying to base binary transmission mode on file attributes or filetype.

SET FILE SUPERCEDE

Syntax: `SET FILE {SUPERCEDE, NOSUPERCEDE}`

SET FILE [NO]SUPERCEDE allows Kermit-11 to accept or reject files received (from either the RECEIVE or GET commands) on a per file basis. The default is SUPERCEDE. By doing SET FILE NOSUPERCEDE Kermit-11 will always check to see if the file to be created is already there (independent of version number) and reject it to the sending server if it exists. This presumes that the Kermit sending the file understands the protocol to reject one file of a (possibly) wildcarded group of files. The main use of this is to resume getting a group of files, as in `GET KER:K11*.*` or `GET KER:MS????.*` having lost the connection after transferring some of the files. If this is set, then any files already transferred will not be transferred again.

SET HOME

SET HOME resets the default device and UIC (or PPN) to nothing, ie, all file opens and creates use your default disk (`SY:`) and your UIC (or PPN).

SET IBM-MODE

Syntax: `SET IBM {ON, OFF}`

The SET IBM ON (or OFF) will instruct Kermit-11 to wait for an XON following each packet sent to an IBM host in linemode. Since the default for IBM mode may not always be appropriate for your IBM compatible system, you can always use the SET HANDSHAKE XON and SET DUPLEX HALF to avoid the parity setting implied by using IBM mode.

SET LINE

Syntax: `SET LINE` *device-designator*

The SET LINE command sets the terminal name up for use with the connect command. To use this you must have access to that device. On many systems terminal lines other than your own are protected from access, and may require special procedures to access them. The form of the device name is TTnnn:, where 'nnn' is a decimal number for RSTS and an octal number for RSX-11M/M+. For RT-11, the device name is simply the MT unit number shown by the SHO TER command, as in '5' for DZ11 unit 0 line 4. If the system is running RT-11 version 5 you can do a SET LIN XL:. At worst case, Kermit-11 can use the console port on RT-11. For more information see the notes later on for RT-11 If you are running `K11POS.TSK` for P/OS on the PRO/350, Kermit-11 will set the line to `XK0:` and the speed to 9600 baud when Kermit starts. To override the line or speed, set HELP SET LINE and HELP SET SPEED. Examples:

`Kermit-11>SET LINE TT55:`	(for RSTS and RSX-11M/M+)
`Kermit-11>SET LINE 5`	(for RT-11 and MT service)
`Kermit-11>SET LINE XK0:`	(for P/OS, done implicitly)
`Kermit-11>SET LINE XL:`	(for RT-11 and XL handler)

See HELP CONNECT, HELP SET DUPLEX and HELP SET SPEED for more information. Also, for TSX+, see notes regarding TSX later in these notes. The RT-11 XL handler has notes later on also.

SET LOGFILE

Syntax: `SET LOGFILE` *filespec*

The SET LOGFILE command creates a debug dump file for you. It must be used BEFORE any SET DEBUG commands can be used. See HELP DEBUG for further information about debugging modes.

SET MODEM

The SET MODEM command defines the type of MODEM use for dialing out on the line set with the SET LINE command, or, in the case of the PRO/350, the XC or XK port. There are only a few modems defined at this time, they are:

VADIC	Generic RACAL-VADIC autodial
VA212PA	Stand alone VADIC VA212
VA212PAR	Rack mounted VADIC VA212
VA4224	Rack mounted VADIC VA4224 .v22bis
HAYES	Hayes smartmodem
DF100	DEC DF112
DF200	DEC DF224
DF03	DEC DF03
MICROCOM	

The DIAL command is then used after the SET MODEM command. For example, on a PRO/350 running P/OS:

```
Kermit-11>set prompt PRO>
PRO>set modem va212pa
PRO>dial 5374411
Modem in command mode
Modem dialing
Connection made, type CONNECT to access remote
PRO>con
Enter class ? VX785A
Class start
Username: BRIAN
Password: _____
```

...*and so on*

SET PACKET-LENGTH

Syntax: `SET PACKET-LENGTH` *length*

You can alter the default transmitted packet length with the SET PACKET-LENGTH command. This should not normally be needed unless the line is very noisy, at which time you should probably give up anyway.

SET PARITY

Syntax: `SET PARITY {EVEN, ODD, MARK, NONE, SPACE}`

This is used with the SET LINE and CONNECT commands to specify the type of parity for the remote link. It defaults to NONE and can be any of ODD, EVEN, MARK or SPACE.

All parity generation is done via software, no special hardware is used. The use of software parity generation is restricted to 8 bit links only. The character format, if parity is set to anything but NONE, will be 7 bits of data followed with high bit set or cleared to indicate the parity. If you set parity to anything but NONE (the default), Kermit-11 will be forced to request 8bit prefixing from the other Kermit-11, which is a method by which Kermit can 'prefix' eight bit characters with a shift code. You MUST use parity (even if MARK or SPACE) when using Kermit-11 with the IBM CMS Series/1 or 7171 3270 emulator, or in linemode through a 3705 front end.

SET PAUSE

Syntax: `SET PAUSE` *seconds*

PAUSE tells Kermit to wait the specified number of seconds between each packet being sent to the other Kermit. This may be useful under situations of heavy system load. This may be automatically computer by Kermit-11 in a future release as a function of line speed.

SET PHONE

Syntax: `SET PHONE {NUMBER, TONE, PULSE, BLIND}`

The SET PHONE NUMBER command allows you to associate a phone number with a symbolic name for later use with the DIAL command. These definitions could be placed in your `KERMIT.INI` file, and then referenced later. Example:

```
Kermit-11>set pho num work 5374411
Kermit-11>set pho num market 16174671234
Kermit-11>dial work
```

The other two SET PHONE options, SET PHONE [TONE][PULSE] and SET PHONE BLIND are not useful unless the appropiate dial formatting string and character sequences for selecting PULSE or TONE, and BLIND dialing are present in the modem definition macros in `K11DIA.MAC`. The format effector for TONE/PULSE is `%M` and the effector for BLIND is `%B`. Currently (in 3.54) only the VA4224 has entries for these options.

SET POS

Syntax: `SET POS {DTE, NODTE}`

The SET POS command allows options SPECIFIC to P/OS to be altered. The most useful option is the SET POS [NO]DTE command. This allows Kermit-11 to use PRO/Communications version 2 for terminal emulation, if this product has been installed on the PRO/350. Of course, if this option is chosen, control is returned to the PRO with the EXIT key (F10) rather than with Control \C.

SET PROMPT

Syntax: `SET PROMPT` *prompt*

The SET PROMPT command is useful if you are using two Kermit-11's to talk to each other. By using the SET PROMPT command, you can change the prompt from 'Kermit-11>' on either (or both) Kermit to something that would indicate which system you are currently connected to. Examples:

```
Kermit-11>set prompt Kermit-11/1170>
Kermit-11>set prompt Fubar>
Kermit-11>set prompt ProKermit-11>
```

SET RECEIVE

Currently the SET RECEIVE and SET SEND basically work the same in that they only alter the END-OF-LINE character and the START-OF-PACKET value, as in:

```
Kermit-11>set rec start 2
Kermit-11>set rec end 12
```

The command SET RECEIVE PACKET-LENGTH command is discussed below.

SET RECEIVE END-OF-LINE

This instructs Kermit-11 to expect something other than the default carriage return (octal 15) at the end of a packet. Kermit-11 will ignore packet terminators. The SET SEND END command is of more use in conditioning outgoing packets.

SET RECEIVE START-OF-PACKET

The normal Kermit packet prefix is Control-A (ASCII 1); this command changes the prefix Kermit-11 expects on incoming 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 and the prefix it uses on outgoing packets.

SET RECEIVE PACKET-LENGTH

Syntax: `SET RECEIVE PACKET-LENGTH` *length*

This command has two functions. The first, and normal one, is to reduce incoming packet lengths in the event that normal sized Kermit packets can not be passed through the communications circuit. There could be, perhaps, some 'black box' somewhere in the link that has a very small buffer size; this command could be used to reduce the size that the SENDING Kermit will use.

The other use is to enable a protocol extension to Kermit called 'Long Packets'. The actual protocol is documented elsewhere, let's just say that this is a way for two Kermit's to use packet sizes far greater than the normal ('Classic') packet size if 90 characters or so. The main use of this feature is in file transfer over links that introduce considerable delay, it is not uncommon for packets to incur an one to two second delay. The net result is a VERY slow running Kermit with an effective speed of perhaps 300 to 600 baud rather than 1200 or 2400 baud. By making the packets longer, we raise the effective speed of such a circuit. The main restriction on the packet size chosen is the link, a given circuit may not pass 500 character packets. Also, BOTH Kermits must support this extension to the protocol, they will always negotiate it before any file transfer. See the notes at the end of this document for more information.

It is HIGHLY recommended that you use the CRC block check, as the default type one checksum could be inadequate for such long packets, as in:

```
Kermit-11>SET BLO 3
```

SET RECORD-FORMAT

Syntax: `SET RECORD-FORMAT {STREAM, VARIABLE}`

Kermit will, by default, create RMS11 variable length implied carriage control records for text files. You can override this and change it to create stream ascii records with the SET RECORD-FORMAT STREAM command. This is useful for RSTS/E systems if you need file compatability with BASIC Plus. This command would be most useful in a `KERMIT.INI` file, which is executed by KERMIT when Kermit starts.

SET RETRY

Syntax: (SET RETRY)*number*

SET RETRY value tells Kermit to try that many times on a NAK'ed packet before giving up. This should only be needed if the line is extremely noisy or the PDP-11 host is running very slowly due to the system load.

SET RSX

The SET RSX command is intended to deal with the peculiarities often found with RSX systems. There are currently three SET RSX commands, as in:

`Kermit-11>SET RSX FASTIO`	Default for packet reading, waits for <CR>.
`Kermit-11>SET RSX CHARIO`	Read one char at a time for packet reading.
`Kermit-11>SET RSX TC.DLU n`	Alters the TC.DLU setting.
`Kermit-11>SET RSX CONNECT ALT`	Uses a new (v2.33) connect driver which bypasses TTDRV flow control.
`Kermit-11>SET RSX CONNECT DEF`	Use old connect code (2.32)

The SET RSX command is subject to change and the above options may be removed in the future. Note the the SET RSX CHARIO may be needed when transfering files with parity enabled. This command alters the method by which a packet is read; instead of waiting for a carriage return, Kermit reads the typeahead byte count and then issues a read for that many characters. This is the same method Kermit-11 ALWAYS uses under P/OS.

SET RT-11 CREATE-SIZE

Syntax: (SET RT-11 CREATE-SIZE)*number*

The SET RT-11 CREATE value command was added to assist those RT-11 users with very small disks to be able to get files with sizes greater that half of the available contiguous space available. While this is NOT a problem going from one Kermit-11 to another Kermit-11 since the PDP-11 Kermit supports a subset of the protocol known as 'ATTRIBUTES', other Kermits may not support the exchange of file sizes (most do not). Thus if your largest contiguous space is 300 blocks and you want to get a 250 block file, the command:

```
Kermit-11>set rt-11 cre 250
```

would be needed, as RT-11 by default only allocates 50 percent of the available space.

SET RT-11 FLOW-CONTROL

Syntax: `SET RT-11 {FLOW-CONTROL, NOFLOW}`

Note that for the connect command under RT-11 you will most likely need xon/off flow control to be generated by Kermit-11. This is enabled with the SET RT-11 FLOW command. This is by default NOFLOW since the modem the author uses, a Vadic 212PA, can't handle XONs and XOFFs while in command mode. The solution here is to escape back to Kermit command mode after the remote system has been logged into, and then type SET RT-11 FLOW.

The effect of SET RT-11 FLOW is for Kermit-11, when in connect mode, to send an XOFF to the host every eight characters. When the loop in the connect module finds no more data in the input buffer, it sends up to 2 XON characters (in case the first XON got lost) to tell the remote system to start sending again. The reason for doing so is that the RT-11 multiple terminal service is very slow about handling input interrupts and does not do any of it's own flow control when it's internal ring buffer gets full. This has been tested at line speeds up to 4800 baud without losing data. This setting should not be needed for use with the XC/XL handlers.

SET RT-11 FLOW has NO effect on packet transmission, since the Kermit packet size is never mode than 96 characters, and the RT-11 input buffer is 134 characters in size.

The SET RT-11 [NO]FLOW command replaces the older SET RTFLOW [ON][OFF].

SET RT-11 VOLUME-VERIFY

Syntax: `SET RT-11 {VOLUME-VERIFY, NOVOLUME}`

Normally RT-11 Kermit-11 will check the directory header of a disk to verify that it most likely contains a valid RT-11 file structure before trying to read the directory. If for some reason your disk does not contain the standard data at offset 760 in the header, Kermit-11 will reject the disk. The SET RT-11 NOVOL command will instruct Kermit-11 to bypass that check.

SET SEND

The SET SEND command controls what Kermit-11 will be doing for outgoing packets in that you may want to alter the packet terminator and/or the start of packet character (by default, 15 octal and 1 octal respectively. See HELP SET RECEIVE for more information.

The only extra option for SET SEND is SET SEND [NO]XON. If the command SET SEND XON is give, then every packet sent will be prefixed with an XON character. This could be useful in situations where flow control is erratic. The actual intent of this option was to try to circumvent a firmware bug in the DHV11 when used under RSTS/E.

SET SPEED

Syntax: `SET SPEED` *speed*

SET SPEED value sets the line speed for the device specified via the SET LINE command, and used for the CONNECT command. Changing the speed of a terminal line requires privilege for RSTS and RSX-11M/M+. The SET SPEED command will only function with a DH11, DHV11, DZ11 or DZV11 multiline interface. Example:

```
Kermit-11>set speed 1200
```

1200 Baud would be a normal speed to use with a VA212PA or a DF03.

Please note that Kermit-11 CAN NOT change the speed of a DL11 type interface, nor can it change the speed of a PDT-150 modem port. For a PDT-150 modem port, use a command of `/M/S:`*nnnn* to change the speed to *nnnn* for the `SPEED.SAV` program.

SET TIMEOUT

Syntax: `SET TIMEOUT` *seconds*

The timeout value tells Kermit how long to wait to get a packet from the other Kermit. If system loads are high, it may be desirable to increase this beyond the default of 10 seconds.

SET TERMINAL

Syntax: `SET TERMINAL {TTY, VT100}`

The SET TERMINAL command simply controls the way which Kermit-11 prints packet counts while send or receiving a file (or group of files). The simplest way is the default, SET TER TTY. Using SET TER VT100 will cause Kermit to display headers for the numbers printed, at a possible cost in packet speed due to screen control overhead. On the PRO/350, VT100 is assumed. On RSTS/E v9.0 and later, the executive is queried for the terminal type.

SET UPDATE

Syntax: `SET {UPDATE` *number*`, NOUPDATE}`

The SET UPDATE command controls the frequency at which the packet count display is updated. The default is 1, displaying each packet. A SET UPD 0 will disable all packet count logs, whereas a SET UPD N will update the display every N packets. The SET NOUPDATE command is the same as SET UPDATE 0.

13.6.1. The DIAL Command

The DIAL command is new for version 3.29 of Kermit-11. The DIAL command is used to dial a number on an attached modem of known type (see SET MODEM). To find out the current known modems, use the SET MODEM ? command. The following example shows a RACAL-VADIC VA212 modem connect to the `XK:` port on a PRO/350 running P/OS version 2.

```
Kermit-11>set prompt PRO>
PRO>set modem va212pa
PRO>dial 5374401
Modem in command modem
Modem dialing
Connection failed, !BUSY
```

```
PRO>dial 5374411
Modem in command modem
Modem dialing
Connection made, type CONNECT to access remote
PRO>con
Enter class ? VX785A
Class start
Username: BRIAN
Password: .....................
```

See SET MODEM for more information.

13.7. System Manager's Notes

13.7.1. Odds and Ends

There are a few odds and ends that should be made aware to the system manager of any PDP-11 system regarding Kermit-11. They are as follows, grouped by operating system. Please note that installation instructions are in `K11INS.DOC` and that additional information may be in Kermit-11's online help command.

Restrictions

Prior to version 2.21, Kermit-11 did not support 8-bit prefixing. Prior to version 2.23, Kermit-11 did not support repeat character encoding.

The PRO/RT-11 version of Kermit-11 will request 8-bit prefixing due to the fact that the XC handler does not support 8BIT data. For most Kermits this should not be a problem. The XC handler always strips bit 7 from the character being sent, so the PRO/RT-11 version of Kermit will request prefixing of such. It does so internally by setting PARITY to SPACE (always clear the high bit, bit seven).

Note that this implies that a SET PARITY SPACE command will force Kermit-11 to request '8bit' prefixing in order to transfer binary files across a seven bit link.

P/OS

Kermit-11 will run on under P/OS on the Pro/350, the executable file is called `K11POS.TSK`. It does NOT run from a menu, the normal way to run it is via the RUN command in DCL. It will support the Kermit-11 attribute packets, thus a PRO/350 connected to a PDP-11 host can transparently handle binary and other types of files. The P/OS Kermit-11 can be run either as a local Kermit or a Kermit server. This has been tested under P/OS version 2 connected to both a PDP-11/23+ and PDP-11/70 RSTS/E host.

When Kermit-11 is started on the PRO, it will automatically do a `SET LINE XK0:` and a SET SPEED 9600. You can, of course, change the speed to whatever you need with the SET SPEED command. The line should be left as `XK0:`.

The top row function keys are mapped internally. Kermit-11 maps F5 (break) into a true BREAK (a space of 275 ms), F6 (interrupt) to Control-C, F10 to Control-Z, F11 to escape (octal 33) and F12 to backspace (octal 10). The incoming escape sequence DECID is intercepted to allow Kermit-11 to send back a device response of VT100.

RSTS/E

Kermit-11 runs on version 7.2 or later of RSTS/E. Due to options present in version 8, binary file transfers will not be possible under version 7.2 of RSTS/E. This is due to the use of 8 bit mode for the terminal link to allow all characters to be passed. The so called '8BIT' terminal setting was new as of version 8.0-06 of RSTS/E.

Any RSTS/E system running Kermit-11 will need the sysgen option for multiple private delimiters in the terminal driver. This special mode is needed since the 'normal' RSTS/E binary terminal mode has a 'feature' that disables binary mode whenever the terminal times out on a read. Since timeouts are essential to Kermit error recovery, binary mode can not be used for i/o.

Certain functions of Kermit-11 require that the system manager install Kermit with temporary privileges, these commands are the SYSTEM, WHO and REMOTE HOST commands. Kermit-11 does NOT need these to operate correctly.

Kermit-11 can only be built (from source, not from HEX files) under RSTS/E version 8.0 or later due to the use of RMS11 v2.0 and new assembler directives.

Support for the server remote login is only available under RSTS/E 9.0 or later. Also, a REMOTE LOGIN command to a RSTS/E server will fail unless the user has the WACNT privilege. While the LOGIN program will skip the password lookup if WACNT is present, Kermit will require a password.

RSX-11M/M+

Kermit-11 can not be installed non-checkpointable due to an apparent RMS11 bug. In other words, don't try to install the task '/CKP=NO'.

To use the connect command effectively, typeahead support is needed in the terminal driver. For RSX-11M+, set the typeahead buffer size high, as in SET /TYPEAHEAD=TT22:200. Also, if your connect line is TT22: (as above), use the mcr command SET/SLAVE=TT22:

Kermit-11 can only be built under RSX-11M version 4.1 or later, or under RSX-11M Plus version 2.1 or later due to the use of RMS11 v2.0 and new assembler directives.

There is a SET RSX command, see HELP SET RSX for further information.

As a side issue, please note that the file `K11POS.TSK` is quite usable under RSX, the difference being that `K11RSX.TSK` has DECNET support and RMS-11 overlayed in the task image (besides which, due to the lack author's systems running RSX may not be up to date) linked into it, whereas K11POS has NO Decnet support but IS linked to the RMS11 library RMSRES (v2), thus K11POS saves disk space as well as supporting named directories, ala VMS style.

RT-11

Kermit-11, as of version 2.20, has been tested under RT-11 version 5.0 under the FB and XM monitors using a DZ11 line for the link, and also on a PDT-150 using the modem port for the link. It has additionally been run under Micro-11's and the PRO/350 using the XL and XC handlers respectively.

Kermit-11 requires .TWAIT support as well as multiple terminal support (unless the XL/XC handler is used). The use of multiple terminal support allows Kermit-11 to use any type of interface sysgened, including the DZ11 and DZV11. It is possible under version 5 of RT-11 to use the XL: handler instead of the multiple terminal support. The use of the XL: driver will result in much faster file transfer at high baud rates. Note that XL: must be set up at system startup or at some time later to set the proper speed, CSR and vector.

For those users who do not have multiple terminal support and do not have the XL handler, Kermit-11 will force the use of the console for data transfers. This will require that Kermit-11 request eight bit prefixing from any other Kermit wishing to send binary data files. Additionally, you can force console mode by doing a SET LINE TT:

Please note that the device name syntax for terminal lines follows the MT unit numbers, thus if a SHO TER gave unit 5 for DZ11 line 0 the the device name would be SET LINE 5. If you use the XL handler, you would say SET LINE XL:. To force the console to be used, you would SET LINE TT:.

Additionally, Kermit-11 for RT-11 looks for its help file, `K11HLP.HLP`, on `DK:` first and then on `SY:` if the first one fails.

Full wildcarding is supported for RT-11, in the form *.type, name.*, *.* and the % character to match any single character.

Kermit-11 can only be built on RT-11 version 5.0 or later due to the use of new assembler directives.

Please note that for the connect command under RT-11 and the use of the MT service, you will most likely need xon/off flow control to be generated by Kermit-11. This is enabled with the SET RTFLOW ON command. This is by default OFF since the modem the author uses, a Vadic 212P, can't handle XONs and XOFFs while in command mode. The solution here is to escape back to Kermit command mode after the remote system has been logged into, and then type SET RTFLOW ON.

Due to overlaying constraints, the RT-11 Kermit-11 will not accept wildcards for the RENAME and DELETE commands and the REMOTE server equivalents.

The executable files are `K11XM.SAV` for the XM system and PRO/350, and K11RT4 for the FB system.

As a final (I hope) RT-11 note, see the RT-11 v5.1 Release Notes page 9-2 and chapter 12. The discussion relevant here regards the use of the XL/XC handlers.

Note that the default XL: handler vector (DL-11, DLV-11) is 300 and the CSR is 176500. For the Micro-11, PDP-11 and LSI-11, when the DL11/DLV11 interface is installed the field service representative will inform you what the CSR and VECTOR are. If they are NOT 176500 and 300, then to use the XL: handler you will need, prior to running Kermit-11, to set them. Suppose the DL vector is 400 and the CSR is 176510. Then the following DCL commands would set the addresses for RT-11:

```
.SET XL CSR=176510
.SET XL VECTOR=400
```

You SHOULD NOT ever alter these settings for XC: on the PRO/3xx. The ONLY settings you can alter for the PRO/3xx is the speed, as in DCL command SET XC SPEED=nnnn. Kermit-11 CAN NOT alter the XC: speed itself. As noted previously in this document, Kermit-11 executes the Kermit-11 command SET LIN XC: implicitly if it finds itself running on a PRO/3xx system.

Note that if your modem requires DTR to be present, you must use either an interface that asserts it (as does the PDT and PRO communications port), force it high internally to the modem, or build a cable to force it high. See HELP MODEM for more information.

TSX+

While most of the above notes for RT-11 apply for TSX+, there are a few differences of note. The first, in that TSX+ is a timesharing system, allows the Kermit user to log in normally from another system running Kermit (as in a Rainbow) and give the TSX+ Kermit the SERVER command and commence file transfer operations from the other system (ie, the Rainbow). If you are dialing INTO a TSX+ system, you need to give the TSX command:

```
.SET TT 8BIT
```

to be able to transfer data to your local (PC, other PDP-11,...) system without incurring the overhead of the Kermit protocol known as eight bit prefixing. If this is not possible, due to your local system requiring parity, or some other intervening device adds parity, then you should give Kermit the command SET PARITY SPACE, to let Kermit know that it can't send binary data as-is.

To use Kermit-11 to dial out from the TSX+ system, the following commands are needed. Note that TSX+ commands will be preceeded by the normal RT-11 prompt, the ever present DOT ('.'), whereas Kermit-11 commands will be prefixed by the default Kermit-11 prompt, 'Kermit-11>':

```
.SET CL LINE=n             Where 'n' is the unit number
.SET CL NOLFOUT
.SET CL SPEED=n            Where 'n' is the speed for that unit
.ASS CL XL
Kermit-11>SET LIN XL:
Kermit-11>CONNECT
```

As of Kermit-11 version 3.44, you may use CL directly in the SET LINE command, as in:

```
.SET CL3 LINE=3
.R K11XM
Kermit-11>SET LIN CL3
Kermit-11>SET SPEED 1200
Kermit-11>CONNECT
```

A sample command file in actual use is:

```
SET CL3 LINE=3
SET CL3 NOLFOUT
SET CL3 TAB
SET CL3 FORM
SET CL3 SPEED=2400
ALLOCATE CL3:
R K11XM
DEALLOC CL3
SET CL3 LFOUT
SET CL3 LINE=0
SH CL
```

If you are running PRO/TSX+, then Kermit will make the assignment of LINE 3 to either CL0 or CL1 if you are running Kermit from the console, ie, LINE 1. The speed will default to the last SET SPEED or the speed set at system boot.

Lastly, TSX+ needs PLAS support to use `K11XM.SAV`, see the installation notes for further data.

RSTS/E version 9.x

RSTS/E does not control modems signals in the manner that RSX or VMS does. VMS always asserts DTR whereas RSTS/E will not assert DTR until the terminal driver can see RCD (also known as DCD) which is pin 8 (eight) for the RS232 connection. To connect directly to a modem (like a VADIC 212, sorry, no DEC modems here) we must do one of two things:

1. Force the modem (via strapping options or whatever) to assert RCD (DCD) pin 8, thus RSTS/E will see carrier and raise DTR (pin 20 for RS232)

2. Set the terminal to LOCAL (RSTS/E V9 syntax 'SET TER TTxx:/NODIAL/PERM') and break pin 20 (DTR) and connect pin 20 to 8 on the modem side. This will cause the modem to be able to dial out and allow RSTS/E to connect to it. You will also need to have the modem assert RCD, pin 8. Keep in mind that the Kermit-11 command DISCONNECT (or HANGUP) will not function if a line is set to NODIAL (INIT SET syntax 'LOCAL'). This has been tested on a Racal Vadic VA212.

3. Break pin 8 (RCD) and loop DTR (pin 20) on the CPU side to RCD (pin 8) on the CPU side. Then use the command SET DTR in Kermit-11 to get RSTS to raise DTR and thus loop it's DTR signal back to RCD. See the next note regarding this.

For those of you who have port switches such as the Gandalf type, there is one additional problem. For Gandalf, suppose you want to connect a DZ11 line to to an AMTB2. You will have a problem, in that the Gandalf AMTB2 wants to see RCD (DCD) asserted to make a connection. What you may need to do is this:

Make a cable for the DZ11 to AMTB2 port as follows:

```
            CPU side                           AMTB2 side
                            20--|
                            8---|-----------8
                            7---------------7
                            3---------------2
                            2---------------3
```

Note that 20 is tied to 8 on the CPU side. Also, 2 is swapped for 3.

Then, the Kermit-11 command SET DTR, which forces RSTS to raise DTR for 30 seconds, will cause the DTR signal to loop back to the RCD (DCD) signal and thus tell RSTS that there is carrier detect which will raise DTR (the chicken or egg question) and get things rolling. The Kermit-11 HANGUP (or DISCONNECT) command will drop DTR and force the modem to break the connection.

RSX and Modems

While the author's experience on RSX is limited, the following notes may be of use. Dialing out on a LOCAL line will often require that the modem assert internally DTR. If a line is set REMOTE on RSX, the driver will assert DTR and RTS. For a modem, like a VA212PAR strapped at the factory defaults, this will cause the modem to assert DSR and RCD. On the VADIC in particular, the modem will drop RCD during a DIAL command unless the modem is configured to assert RCD continuously. For dialing out, ideally the modem should be able to assert RCD via an option or internally settable strap or switch. If this is not possible, an alternative is to break line 8 (RCD) and jumper DTR (20) to RCD (8) on the CPU side. This will force RSX to always see carrier detect and allow a dial sequence to complete. The Kermit-11 command DISCONNECT (or HANGUP) will still disconnect the modem as the modem will drop from the line when it sees DTR go low (assuming the modem is not strapped to assert DTR internally).

13.8. Typical Kermit-11 Transfer Rates

Some sample timings for Kermit-11 and long packet support. The packet size in the RSTS/E to P/OS was 500 bytes, the size from RSTS/E to RSTS/E was 700 bytes. These sizes are somewhat arbitrary, they depend more on the system's buffering capabilities than anything else.

Host buffering capabilities:

P/OS	500 (estimated)
RSTS/E 9.0 or later	up to 7000, given sufficient system pool
RSX-11M+	255 (I/D space CPU only)
RSX-11M	34
RT-11	134 (could be larger with mod to XC/XL)

As it can be seen, large packets make sense only for RSTS/E, P/OS and RSX-11M+ if one wishes to avoid XON/XOFF overhead at high speeds. It should be possible to run larger packets on M+ and RT-11 at lower speeds.

File transferred: `K11POS.TSK`, size 102,400 bytes (200 disk blocks). Actual data packet characters AFTER prefixing was 120,857.

```
Time    Speed  Data rate    Comments
seconds baud

1436    1200   84/sec       11/44 to PRO/350, 'Classic' Kermit
                            local phone call
1237    1200   97/sec       11/44 to PRO/350, 500 Char packets
                            local phone call

2915    1200   41/sen       11/44 to PRO/350, 'Classic' Kermit
                            local call, 1 second ACK delay.
1492    1200   81/sec       11/44 to PRO/350, 500 Char packets
                            local call, 1 second ACK delay.

304     9600   397/sec      11/44 to 11/44, 'Classic' Kermit,
                            connected locally via Gandalf switch.
245     9600   493/sec      11/44 to 11/44, 700 char packets,
                            connected locally via Gandalf switch.
```

The last two timings are much lower than the line speed due to the fact the the PDP 11/44 is running 100% busy trying to keep up with character interrupts using a normal terminal driver. A special purpose driver, such as the XK driver found on P/OS, would have lower overhead and allow somewhat faster data rates.

Long packets were chosen for Kermit-11 due to the lack of suitable interrupt driven i/o (at this time) under one of the operating systems, RSTS/E. The Sliding Windows would likely function better in those situations where the circuit delay is much higher, or when the circuit can not accommodate large packet sizes.

13.9. Common Problems

Connection Fails

Check modem control signals. RSX needs TC.DLU set to two to talk to a dial out modem, otherwise you will need to strap or jumper signals in the modem to have carrier detect set high. RSTS/E also should have the modem assert carrier detect. If not, see the previous notes about modems. If all else fails, put a breakout box in the line and observe what signals are present.

File Transfer Fails.

If the file transfer aborts on retries immediately, there may be a parity problem. If the problem shows up on binary files, try a SET PAR SPACE command to Kermit; that will force eight bit data to be prefixed into seven bits. If you instead get a retry about once every 10 seconds, the other Kermit is not responding and your Kermit is timing out. Check to see if your connection is still present, and try the SET PARITY command.

If you are sending binary data between unlike Kermits, you will most likely have to give the proper command to each to prepare them for the binary data; this is the SET FILE command; for Kermit-11 it's SET FIL BIN (or SET FIL TYP FIX); for VMS Kermit it's SET FIL TYP FIX.

If your Kermit's packets are being echoed back, try a SET SEND START value command for your Kermit, and a SET REC START samevalue for the other Kermit. This will force Kermit to ignore any echoed packets as they won't have the default start of packet character (a CONTROL A, octal 1).

14. Apple II Kermit

Authors: Antonino N. J. Mione (Stevens Institute of Technology),
Peter Trei (Columbia University),
Ted Medin (NOSC), Bob Holley (SERDAC)

Version: 3.85

Date: 1988 July

Kermit-65 Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
`^X/^Y` interruption:	Yes
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	No
Alternate block checks:	No
Terminal emulation:	Yes (VT52, VT100)
Communication settings:	Yes
Transmit BREAK:	Yes
IBM communication:	Yes
Transaction logging:	No
Session logging (raw download):	Yes
Raw upload:	No
Act as server:	Yes
Talk to server:	Yes
Advanced commands for servers:	Yes
Long packets:	Yes
Sliding windows:	No
Local file management:	Yes
Handle file attributes:	No
Command/init files:	Yes
Printer control:	Yes

Kermit-65 is a program that implements the Kermit file transfer protocol for the Motorola 6502 processor family (hence the name, Kermit-65) on the Apple II microcomputer system. It is written in 6502 assembly language and should run on any Apple II or compatible running DOS 3.3 or PRODOS 8. This section will describe the things you should know about the file system in order to make effective use of Kermit, and then it will describe the special features of the Kermit-65 program.

14.1. Supported Systems and Devices

There are several different Apple II's which can run Kermit-65. Kermit will have no problems running on an Apple II, II+, //e, //c or //gs system. Of the different communication devices available for the Apple II, Kermit-65 supports the ones shown in Table 14-1.

It is possible that other cards may have operational characteristics very similar or identical to one of the devices above. If this is the case, it may work using one of the currently available device drivers. The user may want to try each of the above options to see if any of them work. Kermit-65 must be told in which slot the card resides. This may be done with the 'SET' command (documented below).

AE Serial Pro (super serial driver - sw 1 & 3 open 2 & 4 closed)
AIO II (Uses the Apple Com Card driver??? - untested)
ALS dispatcher (Uses the Apple Com Card driver)
Apple Com Serial Card
ASIO (Uses the Apple Com Card driver??? - untested)
Apple Super Serial Card & //c Serial Port
Apple //gs Serial Port
CCS 7710 Serial Card
CCS 7711 (Uses the Apple Com Card driver??? - untested)
D.C. Hayes Micromodem.
Microtek sv-622 Card
Prometheus Versacard (Uses the Apple Com Card driver)
SSM AIO (Uses the Apple Com Card driver??? - untested)

Table 14-1: Apple II Communication Cards Supported by Kermit-65

14.2. The DOS 3.3 File System

Items of importance which will be discussed in this section include filenames and file characteristics.

Apple DOS Filenames

Filenames under Apple DOS may contain almost any ASCII character (including space). It is not recommended that special characters, (i.e. control characters or spaces) be used in a filename to be transferred by Kermit-65 since they may cause problems when parsing the filename. Filenames may be up to 40 characters in length.

Apple DOS File Characteristics

All files in Apple DOS have a file type associated with them which is contained in the directory entry for the file but is not part of the filename itself. There are four types of files in DOS 3.3. They are:

1. APPLESOFT BASIC
2. INTEGER BASIC
3. BINARY
4. TEXT

All file types have their data stored in eight-bit bytes although not all of them need the eighth bit. The two file types containing basic programs required the eighth bit due to the nature of the data being stored. BINARY files are images of memory copied into a file. Often, these are machine code programs. These files require all eight bits. TEXT files normally contain only printable or carriage control characters. They are stored in the form of seven-bit ASCII characters but the eighth bit should always be set since Apples manipulate all text internally as 'Negative ASCII'. When transmitting non-text files the user must insure that both Kermits are handling eight-bit data so that no information is lost. If an eight-bit data path is not available (i.e. the remote Kermit needs to do parity checking with the eigth bit), then eight-bit quoting should be used. Of course, BINARY files as well as Apple BASIC files will not have much meaning on a different system. If the user desires to edit a BASIC file on a mainframe, for instance, s/he must convert it to a TEXT file before sending it over. After receiving the file back on the Apple, the user may convert it back to BASIC once again. The reason BASIC files would be meaningless to a different machine is that the Apple stores BASIC keywords as single character tokens to save space and processing time. To convert a BASIC program to and from a TEXT file, consult the Apple DOS 3.3 Manual. File information can be obtained by issuing the CATALOG command. For example:

```
]CATALOG
```

```
DISK VOLUME 010

  *A 002 HELLO
   B 078 KERMIT
   A 002 READER
   T 005 TESTFILE

]
```

When Kermit-65 is receiving a file, the file it creates on diskette will be of the type indicated by the FILE-TYPE parameter. The file will always be left in an unlocked state after it is closed by Kermit-65. When sending a file, Kermit-65 will use the FILE-TYPE parameter to determine how to detect an End-of-file condition. Thus, it is important to have this set properly in all cases.

Recommendations for Archiving Files

When using a large system for archiving purposes, there is no reason to convert Apple Basic programs into text files before sending them if there is no need to edit them on the mainframe. The FILE-TYPE parameter must always be set correctly when sending and receiving files. The procedure for archiving files is:

1. Run Kermit on remote system.
2. `SET FILE-TYPE TEXT` (*or* `APPLESOFT` *or* `...`) on Kermit-65.
3. Send the files.

14.3. The PRODOS File System

The PRODOS system is essentialy the same as the DOS system with the exception that performance has been improved, hardware usage has been expanded and file names have different syntax. File names are the major importance to the Kermit system. File names have the following syntax:

```
/volname/subdirectory1/.../subdirectoryn/filename
```

where "volname" is the volume name where the file is located. Subdirectory(n) is a subdirectory on the volume and may be omitted. Filenames are much more restrictive than DOS filenames. PRODOS filenames are limited to 15 characters with no imbedded spaces and few special characters, and must begin with an alphabetic character. /volname/sub ... may be omited from the filename by use of the PREFIX command.

Binary file transfer using PRODOS has its dangers when creating new files. PRODOS keeps the file’s size and starting location in the directory which is of course not transferred. Therefore a new binary file will have its starting location 0 which can cause some interesting problems if you try and BRUN the file. Basic files all start at $801 (it says here) so Kermit creates new basic files with a starting address of $801.

14.4. Program Operation

Prior to using Kermit-65 for transferring files, the modem interface must be set to handle data in a certain manner. First, the data format should be 8 data bits and 1 stop bit. Second, the card should be set to no parity. The baud rate (if adjustable) must be set to whatever rate the modem can handle. For the D.C. Hayes Micromodem, these parameters are set correctly by default, so very little has to be done. For the Apple Super Serial Card these are set from within Kermit-65 except the interrupt switch (sw6-2) which must be set for interupts on. For the Microtek SV-622, all applicable parameters are set by Kermit-65. Some mainframes may need parity checking (i.e. most IBM machines). In this case some parity setting (other than none) will usually work. When talking with such mainframes, binary and basic files on the Apple cannot be transferred unless Eighth-bit-quoting is acceptable to the host. If you have the parameters set correctly then the "connect" command will start Kermit talking out the communication port.

File transfer is very dependent upon parity. Make sure the host and local parity are the same. Following are a couple of site's method for file transfer.

We have an IBM 3033 and 4381 and use both 3705/3725 and 7171 or Series/1 front ends. The differences in front ends as far as any microcomputer Kermit is concerned duplex (local-echo on for the 3705, local-echo off for the 7171 or Series/1), parity (the two front ends might use different parity, e.g. Mark for the 3705 and Even for the 7171), and flow control (None for the 3705, XON/XOFF for the 7171).

In Kermit-65, IBM mainframe users need to set the following parameters:

BAUD	Whatever is supported.
PARITY	EVEN, ODD, or MARK, whatever your front end requires.
FLOW	XON for the 7171, NONE for the 3705.
FLOW DELAY	00
LOCAL-ECHO	OFF for 7171, ON for 3705

In Kermit-65, SERDAC VAX 8800 users need to set the following parameters:

BAUD SERDAC Dial-up & 300, 1200, or 2400 baud FIRN Dialup: (the highest your modem and the dial-up connection will support)

Ethernet Hardwire: 300, 1200, 2400, or 4800 baud.

PARITY NONE

FLOW XON

FLOW DELAY 00 (higher for printers, logging, or "slow" Apples)

LOCAL-ECHO OFF

NOTE: If you want to do a binary file transfer (Apple binary or BASIC files) via a FIRN Network connection to the SERDAC VAX 8800, you must SET PARITY SPACE before the transfer is initiated; that will insure that eight-bit quoting is used. If you dial directly into the VAX 8800, SET PARITY NONE; eight-bit quoting (which is less efficient) is not required.

Conversing With Kermit-65

Kermit-65 reads file `KERMIT.INIT` from the default drive when started. The lines of this file are executed one at a time starting at the begining. This file should be an ASCII text file and contain commands to set up Kermit's parameters as desired. It will also execute Kermit's other commands. However, any command which reads a file (like MODEM) or leaves local mode (like CONNECT) will terminate reading of this file and continue with the command specified. Use your favorite editor to produce this file. Here's a sample:

```
set display 80 3
set keyboard 2e
set baud 4800
modem
```

Kermit-65's prompt is "`Kermit-65>`". To run Kermit-65 and issue commands to it, type "`brun kermit`". Example:

```
]BRUN KERMIT

NOSC/STEVENS/CU - APPLE ][ KERMIT-65 - VER 3.84

Kermit-65>send testfile
```

(file is sent...)

```
Kermit-65>status
```

(performance statistics are printed...)

```
Kermit-65>(other commands...)
            .
            .
            .
Kermit-65>exit
]
```

Like many Kermit programs, Kermit-65 uses a DEC-20 style command parser. During interactive operation, you may use the ?-prompting help feature ("?") and recognition (ESC) features while typing commands. A question mark typed at any point in a command displays the options available at that point; typing an ESC character causes the current keyword to be completed (or default value to be supplied). If you have not typed sufficient characters to uniquely specify the keyword (or if there is no default value) then a beep will be sounded and you may continue typing. Keywords may be abbreviated to any prefix that is unique.

Remote and Local Operation

Kermit-65 is normally run in local mode. It may be run as a remote Kermit as well although there is no advantage to doing things that way. Kermit-65 supports User-mode commands for talking to a Server, and it does support a limited server mode.

14.5. Kermit-65 Commands

The SEND Command

Syntax: `SEND` *filespec*

The SEND command causes a file to be sent from the Apple to the remote system. The Filespec is the name of the file on the Apple diskette to be sent. The parser will not accept control characters and certain special characters in a filename (like comma), so the user may have to rename the file before it is sent. The user may also have problems in filename compatibility with remote Kermits. If the remote Kermit does not have the facilities to beat the filename into a format that its system likes, the user may have to rename the file before sending it. Thanks to Dick Atlee, wildcards are now acceptable when sending files (they have always been acceptable when receiving files). The ‘‘*’’ is a multiple character wildcard and the ‘‘=’’ is a single character wildcard.

The default disk drive is used for file transfers this can be changed with the ’SET DEFAULT-DISK’(DOS) or ’SET PREFIX’(PRODOS) command (explained below). As a file is being sent, the screen displays ’RECEIVING NUMBER OF BYTES’ and ’SENDING NUMBER OF BYTES’ followed by the hexidecimal number of bytes transferred since start of transmission. If a packet must be transmitted several times and it reaches the maximum retry count, the transfer will fail and the ’Kermit-65>’ prompt will return. If the remote Kermit sends an error packet, the text of the packet will be displayed on the screen, the transfer will fail, and the prompt will return. Currently, a packet can be retransmitted manually by typing anything on the keyboard. If a ’Q’ is typed, the entire transmission will be aborted.

The RECEIVE Command

Syntax: `RECEIVE [`*filespec*`]`

The RECEIVE command tells Kermit-65 to receive a file or file group from the other system. If only one file is being received, you may include the optional filespec as the name to store the incoming file under; otherwise, the name is taken from the incoming file header. If the name in the header is not a legal filename, Kermit-65 will attempt to change it into something legal. If FILE-WARNING is on and an incoming file has a name identical to a file already existing on the diskette, Kermit-65 will issue a warning to the user and attempt to modify the filename to make it unique. Currently, a packet can be retransmitted manually by typing anything on the keyboard. If a 'Q' is typed, the entire transmission will be aborted. Filespec is required when xmodem protocol is used.

The TAKE Command

Syntax: `TAKE` *filespec*

The TAKE commands tells kermit-65 to execute commands from the specified file similarly to the KERMIT.INIT file. See discussion on KERMIT.INIT above for details.

The TYPE Command

Syntax: `TYPE` *filespec*

The TYPE commands tells kermit-65 to print to the screen from the specified file. Text files only and works best with 80 characters per line or less.

The GET Command

Syntax: `GET` *remote-filespec*

The GET command requests a remote Kermit server to send the file or file group specified by *remote-filespec*. This command can be used with a Kermit server on the other end. The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. So if the remote Kermit supports wildcards you can specify them in the *remote-filespec*. If the remote Kermit is not capable of server functions, then you will probably get an error message back from it like "Illegal packet type". In this case, you must connect to the other Kermit, give a SEND command, escape back, and give a RECEIVE command. Currently, a packet can be retransmitted manually by typing anything on the keyboard. If a 'Q' is typed, the entire transmission will be cancelled.

The CONNECT Command

Syntax: `CONNECT`

Establish a terminal connection to the remote system using all the current SET paramaters for terminal type, speed, parity, etc. Get back to Kermit-65 by typing the escape character followed by the letter C. The escape character is Control-`@` by default. When you type the escape character, several single-character commands are possible. These are shown in Table 14-2.

You can use the SET ESCAPE command to define a different escape character. When CONNECTed, Kermit-65 will be passing characters entered on the keyboard to the remote system, and passing characters from the remote system to the Apple screen. Incoming characters are interpreted according the selected terminal type (see SET

?	List all the possible single-character arguments.
B	Send a BREAK signal.
C	Close the connection and return to Kermit-65.
D	Drop the phone line to the remote and return to Kermit-65.
E	rEfresh the screen (useful for clearing garbage on screen).
K	TOGGLE Keypad application-mode on/off.
P	Toggle the Printer on/off.
R	pRint the screen, >= //e required
S	Show Status of the connection.
W	sWap the del and backspace key.
0	Send a null (ASCII 0).
^@	(or whatever the Connect-Escape character is): Send the Connect-Escape character itself.

Table 14-2: Kermit-65 Single-Character CONNECT Escape Commands

TERMINAL).

On an Apple II+ with an incomplete keyboard, special characters can be typed by prefixing regular characters with a right-arrow. On uppercase-only screens, uppercase characters are shown in inverse and lowercase characters are displayed as normal uppercase characters.

Here are the rules for using the special 2/2+ input, to get all printable ASCII characters, and how they appear on the screen. Special meanings are applied in various contexts to certain characters. The left and right arrow keys do special things, and sometimes the escape key does as well. For letters, the keyboard is always in either default UPPERCASE mode or default lowercase mode. When in UPPERCASE, all letters typed are sent out as uppercase. In lowercase, all letters are sent as lowercase. To reverse the case for the next character only, hit the right-arrow ("prefix") key. To switch the default case, hit the prefix-key twice in a row. For funny characters, the prefix key is also used to get the unusual punctuation characters which are not on the Apple keyboard. Table 14-3 shows the Apple II/II+ keyboard escapes; the letter "p" represents the prefix character.

To Get	Type	Appearence
Left Square Bracket	p(	[
Right Square Bracket	p)	]
Left Curly Bracket	p<	{
Right Curly Bracket	p>	}
Underline	p-	_
Backslash	p/	\
Tilde (wiggle)	p^	~
Vertical Line	p.	|

Table 14-3: Apple II/II+ Keyboard Escapes

The left-arrow key sends a rubout (ASCII 127). With left-arrow and right arrow doing special things, it's a little hard to enter their characters (^H and ^U respectively). There is therefore an escape from prefix mode sequence. If you type prefix-ESC, the next character is sent without any interpretation. If you have the capability for upper/lower case, etc, then use the 'SET KEYBOARD' and 'SET DISPLAY' commands to specify complete keyboards.

The HELP Command

Syntax: `HELP`

Typing HELP alone prints a brief summary of the Kermit-65 commands.

The MODEM Command

Syntax: *MODEM*

This command is designed for the hayes smart modem. Typing MODEM causes the file `KERMIT.MODEM` in the default drive/path to be used as a menu. You will be able to select any line in the file to be sent to the modem. Sorry, you can't back up to a previous menu, you will have to Quit and execute MODEM again. A "connect" response from the smart modem will cause Kermit leave the modem command and execute the CONNECT command. For a Hayes smart modem this file should have commands using text status responses (not numbers). One command per line with comments allowed after the first space (blank). Use your favorite editor to produce this ASCII text file. Since the attention Hayes command (AT) requires a delay the ''&'' character becomes the time delay for Kermit. Each ''&'' causes a delay of one second on a 6502 chip. If you have a //gs or an accelerator board you may have to use the SET TIMING command to produce a one second delay. If you really need to send the "&" character to the modem then the "\" is the escape character. Put a "\" before any character and that character will be sent asis. Of course two "\"'s will produce one "\". Normally Kermit will wait for 27 seconds (again on a 6502 chip) for the modem to respond, but any character typed on the keyboard will terminate this wait. If may hear the busy signal and there is no sense waiting any longer, so hit (crash-not so hard) any key on the keyboard.

Following is an example of the `KERMIT.MODEM` file:

`+++&&ATH`	*Get the Hayes Smartmodem's attention and then hang up.*
`ATDP1234567`	*Call your local BBS with pulse dialing.*
`ATDT8901234`	*Call your work dialup phone with touch tone dialing.*

The CATALOG Command

Syntax: `CATALOG`

Typing CATALOG produces a catalog (directory) listing of your default drive.

The DELETE Command

Syntax: `DELETE` *filespec*

Typing DELETE causes the file specified to be deleted.

The SERVER Command

Syntax: `SERVER`

Typing SERVER alone turns Kermit into a file server to a remote Kermit. Currently server mode will handle remote "send", "get", "remote" and "fin" commands. Variants of the above commands will probably work but file serving is very limited at present. Because the Apple requires knowledge of file types you can use the "remote Kermit" (or whatever the remote Kermit's syntax is) command to set the file-type on the server. Yes, the server will execute any command so you can really get the server into trouble (this is not a BBS). You must have the appropiate file type set before transfering files. You can exit server mode by typing Control-C (`^C`) when not doing file transfers or the remote can of course terminate via the "fin" command.

The REMOTE Command

Syntax: `REMOTE [`*option character-string*`]`

The only option currently is "kermit". This command submits the command "character-string" to the remote Kermit's command processor. Long replys are not paged so you will have to use ^S to stop the screen. The obvious usage is for setting and showing parameters on the remote Kermit.

The EXIT and QUIT Commands

Syntax: `EXIT` *or* `QUIT`

Exit from Kermit-65. You can restart the program, provided you haven't run anything else, by typing 'CALL 4096'.

The SET Command

Syntax: `SET` *parameter* `[`*option* `[`*value*`]]`

Establish or modify various parameters for file transfer or terminal connection. You can examine their values with the SHOW command. The following parameters may be SET:

APPLICATION-MODE	Set VT100 gs keypad in/out of application mode.
BAUD	Which baud rate should the com card use?
CURSOR-KEYS-VT100	In VT100 mode cursor keys give VT100 sequences.
DEBUGGING	TERSE or VERBOSE packet information.
DEFAULT-DISK	Which Diskette drive is used for DOS 3.3 file transfer?
DISPLAY	Which type of screen display is being used?
ESCAPE	Character for terminal connection.
FILE-TYPE	Type of Apple file being sent/received.
FILE-WARNING	Warn users if incoming file exists?
FLOW	Should xon/xoff flow control be used to the remote?
KEYBOARD	II+ or //e keyboard.
KEYPAD	Is there a gs style keypad?
LOCAL-ECHO	Full or half duplex switch.
PARITY	Character parity to use
PREFIX	Which default prefix to use with PRODOS?
PRINTER	Should the printer be used for the display?
PROTOCOL	Which protocol is to be used for file transfer.
RECEIVE	Various parameters for receiving files
SEND	Various parameters for sending files
SLOT	Which slot # is communication device in?
TIMER	Should Kermit observe the receive timeout value?
TIMIMG	Change the time loop for 1 ms. delays.
TERMINAL	Which type of terminal should Kermit emulate?

SET APPLICATION-MODE

Syntax: `SET APPLICATION-MODE {ON, OFF}`

For VT100 emulation with a gs-style keypad you can set the keypad in or out of application mode. Some computer systems set this via escape sequences so it may not be necessary to use this command.

SET BAUD

Syntax: `SET BAUD` *value*

Value is the baud rate for your communication card. For the super serial and the microtek it can be 300 to 19200. The actual values will depend upon the com card you are running with.

SET CURSOR-KEYS-VT100

Syntax: `SET CURSOR-KEYS-VT100 {ON, OFF}`

In VT100 emulation the cursor keys can also emulate the VT100 cursor keys.

SET DEBUGGING

Syntax: `SET DEBUGGING {TERSE, VERBOSE, OFF}`

Record the packet traffic on your terminal. Options are: TERSE, Show packet info only (brief). VERBOSE displays packet field descriptions with packet info (lengthy). OFF disables display of debugging information (this is the default).

SET DEFAULT-DISK

Syntax: `SET DEFAULT-DISK {SLOT, VOLUME, DRIVE}` *value*

This DOS command will tell Kermit-65 which disk drive should be used for file transfers. The three parameters which may be set separately are SLOT, VOLUME and DRIVE. The value for SLOT ranges from 1 to 7. The value for DRIVE is either 1 or 2. The value for VOLUME ranges from 0 to 255.

SET DISPLAY

Syntax: `SET DISPLAY {2E, 2P}` or `SET DISPLAY 80-COL` *number*

This command will tell Kermit-65 which kind of screen display you want to use. If you have an Apple II or II+ without an 80 column card, use the first syntax. If you have any kind of an Apple with an 80 column card, enter: SET DISPLAY 80, followed by a space and the slot number where the card resides (if you don't know the slot number, or the card is built-in to the set, try 3).

SET ESCAPE

Syntax: `SET ESCAPE` *hexidecimal-number*

Specify the control character you want to use to "escape" from remote connections back to Kermit-65. The default is 0 (Control-@). The number is the hex value of the ASCII control character, 1 to 37, for instance 2 is Control-B, B is Control-K.

SET FILE-TYPE

Syntax: `SET FILE-TYPE {APPLESOFT, INTEGER, TEXT, BINARY, OTHER` *hex-value*`}`

This will inform Kermit-65 what type of file is being sent or received. It is important that this is set correctly since Kermit-65 must create a file of the appropriate type when receiving (and it has no way of knowing what kind of file it is). When Kermit-65 is sending, it must also know the type of file since that tells it how to detect the actual end-of-file. The keywords for this parameter are listed below. OTHER includes an added hex-value so that the user may specify the hex value of the file-type. This has meaning only in PRODOS and allows the user to specify any of the many different file types used in PRODOS, see Tables 14-4 and 14-5 (thanks to Phil Chien, M L Stier et al).

APPLESOFT	The file being transfered is an Applesoft Basic program.
INTEGER	The file being sent/received is an Integer Basic program.
TEXT	The file being sent/received is an ASCII Text file.
BINARY	The file being sent/received is a Binary image.
OTHER	The type of file being sent/received is specified by the hex-value.

SET FILE-WARNING

Syntax: `SET FILE-WARNING {ON, OFF}`

This tells Kermit-65 whether to warn the user about incoming filenames conflicting with existing files or not. If there is a conflict Kermit-65 will attempt to change the file name to something unique.

SET FLOW

Syntax: `SET FLOW {OFF, XON, DELAY` *number*`}`

SET FLOW allows one to use the XON/XOFF protocol when connected to a remote site. Delay timings are part of

? List of most of the prodos file types.

Num		Name	OS	
$00	r		typeless	
$01	r	BAD	both	BAD blocks file
$02	r	PCD	SOS	Pascal CoDe file
$03	r	PTX	SOS	Pascal TeXt file
$04	r	TXT	both	ASCII text file
$05	r	PDA	SOS	Pascal DAta file
$06	r	BIN	both	BINary file
$07	r	CHR	SOS	CHaRacter font file
$08	r	PIC	both	PICture file
$09	r	BA3	SOS	Business BASIC (SOS) program file
$0A	r	DA3	SOS	Business BASIC (SOS) data file
$0B	r	WPD	SOS	Word Processor Document
$0C	r		SOS	SOS system file
$0D	r		SOS	SOS reserved file type
$0E	r		SOS	SOS reserved file type
$0F	r	DIR	Both	subDIRectory file
$10	r	RPD	SOS	RPS data file
$11	r	RPI	SOS	RPS index file
$12	r		SOS	Applefile diskcard file
$13	r		SOS	Applefile model file
$14	r		SOS	Applefile report format file
$15	r		SOS	Screen library file
$16	r		SOS	SOS reserved file type
$17	r		SOS	SOS reserved file type
$18	r		SOS	SOS reserved file type
$19	r	ADB	ProDOS	AppleWorks Database file
$1A	r	AWP	ProDOS	AppleWorks WordProcessing file
$1B	r	ASP	ProDOS	AppleWorks Spreadsheet file
$1C-$5F	r			Reserved
$60-$6F	r		ProDOS	PC Transporter (Applied Engineering) reserved filetyp
$60	r	PRE	ProDOS	ProDOS preboot driver
$61-$6A	r		ProDOS	Reserved
$6B	r	NIO	ProDOS	PC Transporter BIOS and drivers
$6C	r		ProDOS	Reserved
$6D	r	DVR	ProDOS	PC Transporter device drivers
$6E	r		ProDOS	Reserved
$6F	r	HDV	ProDOS	MSDOS HardDisk Volume
$70-$9F	r			Reserved
$A0	r	WPF	ProDOS	WordPerfect document file
$A1	r	MAC	ProDOS	Macrofile
$A2	r	HLP	ProDOS	Help File
$A3	r	DAT	ProDOS	Data File
$A4	r			Reserved
$A5	r	LEX	ProDOS	Spelling dictionary
$A6-$AB	r			Reserved

Table 14-4: PRODOS file types, part 1

this command. Using delay times is probably a desperation move to keep the screen/printer from losing characters. Setting the timings will have to be set by experience. Perhaps the best way to set the timings is to bring the values down until you get failures and then double the timing figure. Both LOG and SET PRINTER will probably depend on flow control.

OFF Turn off flow control

Num		Name	OS	
$AC	r	ARC	ProDOS	General Purpose Archive file
$AD-$AF	r			Reserved
$B0	r	SRC	ProDOS	ORCA/M & APW source file
$B1	r	OBJ	ProDOS	ORCA/M & APW object file
$B2	r	LIB	ProDOS	ORCA/M & APW library file
$B3	r	S16	ProDOS	ProDOS16 system file
$B4	r	RTL	ProDOS	ProDOS16 runtime library
$B5	r	EXE	ProDOS	APW shell command file
$B6	r	STR	ProDOS	ProDOS16 startup init file
$B7	r	TSF	ProDOS	ProDOS16 temporary init file
$B8	r	NDA	ProDOS	ProDOS16 new desk accessory
$B9	r	CDA	ProDOS	ProDOS16 classic desk accessory
$BA	r	TOL	ProDOS	ProDOS16 toolset file
$BB	r	DRV	ProDOS	ProDOS16 driver file
$BC-$BE	r			Reserved for ProDOS16 load file
$BF	r	DOC	ProDOS	document file
$C0	r	PNT	ProDOS	//gs paint document
$C1	r	SCR	ProDOS	//gs screen file
$C2-$C7	r		Reserved	
$C8	r	FNT	ProDOS	Printer font file
$C9	r		ProDOS	finder files
$CA	r		ProDOS	finder icons
$CB-$DF	r			Reserved
$E0	r	LBR	ProDOS	Apple archive library file
$E1	r			Unknown (unlisted)
$E2	r	ATI	ProDOS	Appletalk init file
$E3-$EE	r	Reserved		
$EF	r	PAS	ProDOS	ProDOS Pascal file
$F0	r	CMD	ProDOS	added command file
$F1-$F8	r		ProDOS	User defined filetypes
			(popular ones include:)	
$F1	r	OVL	ProDOS	Overlay file
$F2	r	DBF	ProDOS	Database file
$F3	r	PAD	ProDOS	MouseWrite file
$F4	r	MCR	ProDOS	AE Pro macro file
$F5	r	ECP	ProDOS	ECP batch file
$F6	r	DSC	ProDOS	description file
$F7	r	TMP	ProDOS	temporary work file
$F8	r	RSX	ProDOS	linkable object module
$F9	r	IMG	ProDOS	ProDOS image file
$FA	r	INT	ProDOS	Integer BASIC program
$FB	r	IVR	ProDOS	Integer BASIC variables file
$FC	r	BAS	ProDOS	AppleSoft BASIC program
$FD	r	VAR	ProDOS	AppleSoft BASIC variables file
$FE	r	REL	ProDOS	ProDOS EDASM relocatable object module file
$FF	r	SYS	ProDOS	ProDOS8 system file

Table 14-5: PRODOS file types, part 2

XON Turn on xon/xoff flow control with the remote

DELAY *number* Delay the micro until XOFF takes effect

Delay followed by a number (including 0) delays the program for that many milliseconds after the XOFF is given to the remote. This delay allows the XOFF to take effect before the program continues.

NOTE: Except for printing and logging, most Apples will not require you to use a flow delay, even at rates up thru

19200 baud; for proper screen control, however, certain older Apple IIe's may require a fairly high delay (120-160 dec), even at 300 baud.

SET KEYBOARD

Syntax: `SET KEYBOARD {2P, 2E}`

SET KEYBOARD tells Kermit-65 if the user has a full keyboard (2E) or not (2P). If the user is on an Apple II+, this should be set to 2P (which is the default). When set to that, character translations are available by using the right-arrow key as a prefix character, as shown in Table 14-3.

SET KEYPAD

Syntax: `SET KEYPAD {ON, OFF}`

SET KEYPAD tells Kermit-65 if the user has an Apple//gs-style keypad available. This is automatically set on a gs but must be set manually on other machines. With keypad set ON then "VT100 keypad on an Apple keyboard" (see below) will not be used but the actual keypad will be.

SET LOCAL-ECHO

Syntax: `SET LOCAL-ECHO {ON,OFF} [Default: OFF]`

This command tells Kermit-65 to echo to the screen characters you type on the keyboard (LOCAL-ECHO = ON), or to let the remote system echo the typed characters (LOCAL-ECHO = OFF). If, when CONNECTed to the remote, you see a duplicate of every character you type, escape back to Kermit-65, and SET LOCAL-ECHO OFF. If, when CONNECTed to the remote, you see nothing echoed to the screen, escape back to Kermit-65, and SET LOCAL-ECHO ON.

SET PARITY

Syntax: `SET PARITY {NONE, EVEN, ODD, MARK, SPACE} [Default: NONE]`

This command tells Kermit-65 which parity you want to use while communicating with the remote. Most remotes use NONE; some use EVEN, a few may use the other possible values. If you have a choice of parity to use with a remote machine, if possible, choose NONE.

SET PREFIX

Syntax: `SET PREFIX string [Default: boot volume]`

This command allows you to specify a ProDOS file prefix.

SET PRINTER

Syntax: `SET PRINTER {ON, SLOT}` *number*
or `SET PRINTER OFF`

This allows one to turn the printer on for printing what is displayed on the screen. With all the different printers and printer cards there will be a lot of variability here but flow control (XON/XOFF) is probably required when you are connected to a remote site. The printer can also be toggled on/off via the ESCAPE character followed by the command "P".

Remember when you use your printer there are a lot of variables here. What was being sent to the screen now is being sent to your printer. If you were emulating the VT52 your printer may not know how to handle the escape sequences, tabs, etc. It may be you can tell the host you are a tty or some such device that will give produce control codes that your printer can handle. Some printers may require the flow control and delay to get readable printing.

ON	Turn the printer on, slot number is required.
OFF	Turn the printer off.
SLOT *number*	Printer card is in slot "number".

SET PROTOCOL

Syntax: `SET PROTOCOL {KERMIT, XMODEM}`

SET PROTOCOL tells kermit-65 which protocol to use for file transfer. NOTE: When XMODEM is used you will probably want to change the carrage return and carrage return/line feed translation in the send/receive parameters. eg. "SET SEND CR<->CR,LF OFF" and "SET RECEIVE CR<->CR,LF OFF".

SET SLOT

Syntax: `SET SLOT` *number*

This option tells Kermit-65 in which slot the communication device is located. The range for the number parameter is 1-7.

SET TIMER

Syntax: `SET TIMER {ON, OFF}`

SET TIMER will turn on or off the timeout checking for receive file transfers. Since there is no clock for exact timing a loop of instructions has been set up assuming a 1 megacycle CPU. CPUs which run faster may have to make allowances via the SET RECEIVE TIMEOUT command or the SET TIMING command.

SET TIMIMG

Syntax: `SET TIMIMG { number }`

Kermit uses a timing loop with the rom address $fca8 to produce a 1 ms. delay. If you have a machine that runs faster than the 6502 chip you may have to increase this number to get the 1 ms delay.

SET TERMINAL

Syntax: `SET TERMINAL {MONITOR, NONE, VT100, VT52}`

When TERMINAL is NONE, then all incoming characters (except nulls) are passed directly to the display.

MONITOR emulation simply displays all the characters received from the remote (except nulls) without any formating of the screen (40 or 80 characters per line). Control characters are displayed inverse.

VT100 Emulation

The Kermit-65 VT100 emulator is a small but working set of a true VT100 terminal. It appears to work with most of the standard full screen editors and processors on BSD UNIX and VAX/VMS machines. An Apple//e, //c, or //gs is probably required with the Apple 80 column text card. Sorry, but the II and II+ will probably have to use one of the other terminal options. The VT100 keypad has also been defined for the application mode via the OA/CA/game button. Figure 14-1 shows the Apple keypad looks like to EDIT (VMS), an Figure 14-2 shows the layout on an Apple//gs keyboard. When using EVE (VMS) the meaning of the keys will of course change.

As you can see the keypad is physically laid out like the VT100 keypad except for the lower right corner. Notice that above the keys are the VT100 labels while within the box (key) is the Apple key label. Also the arrow keys work as VT100 arrow keys with the OA/CA/game button.

```
               PF1     PF2     PF3     PF4     .(period)
             ------- ------- ------- ------- -------
            |       |       |findnxt|delline|select |
            |   6   |   7   |   8   |   9   |   0   |
            | gold  | help  | find  |undelln| reset |
             ------- ------- ------- ------- -------
                  7       8       9       -
                ------- ------- ------- -------
               | page  |section|append |delword|
               |   Y   |   U   |   I   |   O   |
               |command| fill  |replace|undelwd|
                ------- ------- ------- -------
                     4       5       6       ,
                   ------- ------- ------- -------
                  |advanc|backup |  cut  |delchar|
                  |  H   |   J   |   K   |   L   |
                  |bottom|  top  | paste |undelch|
                   ------- ------- ------- -------
                        1       2       3      ENTER
                      ------- ------- ------- -------
                     | word  |  eol  | char  | enter |
                     |   N   |   M   |   ,   |   .   |
                     |chgcase|deleol |specins|  sub  |
                      ------- ------- ------- -------
                            0
-------------------------------------
                           bline      |
     space bar                        |
                          openline    |
-------------------------------------
```

Figure 14-1: VT100 Keypad on an Apple Keyboard

VT52 Emulation

SET TERMINAL VT52 will turn on the VT52 terminal emulation. One thing that is required is your 80-column card must handle the $16 command in order for reverse scrolling to work. The Apple//e 80 column card handles this fine. The VT52 keypad has been defined using the open/closed Apple. For II or II+ one will have to have a game paddle or joy stick (key shift mod too????) and use the buttons. When a button/open/closed Apple is pushed then the keys starting with 6,7,8 & 9 form the top of the keypad. Key 6 is the blue key key 7 is the red key etc. The keys directly below the 6,7,8 & 9 and shifted one-half key to the right form the second row of the keypad etc. Every thing is fine until you get to the last row on the keypad. There the sp bar is 0 and the other two keys are moved to the upper right as the 0 & - keys. This way the arrow keys are available as VT52 keys with the OA/CA/game button combination (thanks to Dick Atlee for this idea). With those two exceptions the keypad is physically similar to a VT52 keypad. Remember the open/closed Apple or the game button must be pushed (like the cntl key) to get the keypad emulation. Figure 14-3 should clear up the questions.

The arrow keys work as VT52 arrow keys with the OA/CA/game button.

```
   PF1     PF2     PF3     PF4
 ------- ------- ------- -------
|       |       |findnxt|delline|
| CLEAR |   =   |   /   |   *   |
| gold  | help  | find  |undelln|
 ------- ------- ------- -------
   7       8       9       -
 ------- ------- ------- -------
| page  |section|append |delword|
|   7   |   8   |   9   |   +   |
|command| fill  |replace|undelwd|
 ------- ------- ------- -------
   4       5       6       ,
 ------- ------- ------- -------
| advanc|backup |  cut  |delchar|
|   4   |   5   |   6   |   -   |
| bottom|  top  | paste |undelch|
 ------- ------- ------- -------
   1       2       3       ENTER
 ------- ------- ------- -------
| word  |  eol  | char  |       |
|   1   |   2   |   3   |       |
|chgcase|deleol |specins|       |
 ------- ------- -------| enter |
        0          .    | ENTER |
 -----------------------|  sub  |
|     bline     |select |       |
|       0       |   .   |       |
|  openline     | reset |       |
 -------------------------------
```

Figure 14-2: VT100 Keypad on an Apple//gs or Equivalent Keypad

THE SHOW COMMAND

Syntax: `SHOW [`*option*`]`

The SHOW command displays various information:

ALL	All parameter settings (this is quite long).
BAUD	Baud rate of the com card.
CURSOR-KEYS-VT100	Are the cursor keys emulating the VT100 cursor keys?
DEBUGGING	Debugging mode.
DEFAULT-DISK	Which Diskette drive is used for file transfer?
DEVICE-DRIVER	Which communication device is being used?
DISPLAY	Which screen display is being used?
ESCAPE	Character for terminal connection.
FILE-TYPE	Of Apple DOS/PRODOS file being sent/received.
FILE-WARNING	Warn users if incoming file exists?
FLOW	Should XON/XOFF flow control be used to the remote?

```
^ is up arrow    BLUE     RED     GRAY     ^    .(period) ENTER
                 ------- ------- ------- ------- ------- -------
                |       |       |delline|  up   |select | enter |
                |   6   |   7   |   8   |   9   |   0   |   -   |
                | gold  | help  |undelln|replace| reset |  sub  |
                 ------- ------- ------- ------- ------- -------
  v is down arrow    7       8       9       v
                    ------- ------- ------- -------
                   | page  |findnxt|delword| down  |
                   |   Y   |   U   |   I   |   O   |
                   |command| find  |undelwd|section|
                    ------- ------- ------- -------
                        4       5       6       ->
                       ------- ------- ------- -------
                      |advanc|backup |delchar| right |
                      |   H   |   J   |   K   |   L   |
                      |bottom|  top  |undelch|specins|
                       ------- ------- ------- -------
                           1       2       3      <-
                          ------- ------- ------- -------
                         | word  |  eol  |  cut  | left  |
                         |   N   |   M   |   ,   |   .   |
                         |chgcase|deleol | paste |append |
                          ------- ------- ------- -------
                              0
    ---------------------------------------
                               bline       |
         space bar                         |
                              openline     |
    ---------------------------------------
```

Figure 14-3: VT52 Keypad on an Apple Keyboard

KEYBOARD	II+ or //e keyboard.
KEYPAD	Does a gs-style keypad exist?
LOCAL-ECHO	Full or half duplex switch.
PARITY	Character parity to use
PREFIX	Which default prefix to use with PRODOS?
PRINTER	Should the printer be used for the display?
PROTOCOL	Which protocol is to be used for file transfer.
RECEIVE	Various parameters for receiving files
SEND	Various parameters for sending files
SLOT	Which slot # is communication device in?
TIMER	Is the receive timeout on or off?
TERMINAL	Which terminal (if any) should Kermit emulate?

The above options are analogous to the equivalent SET commands.

The STATUS Command

Syntax: `STATUS`

Give statistics about the most recent file transfer. This includes information such as number of characters sent/received, number of data characters sent/received, and last error encountered.

The LOG Command

Syntax: `LOG` *filespec*

When connected to a remote site, log the remote session's output to the specified file. The file type and file warning protocols are observed. This command is dependent upon the flow control (XON/XOFF) working. Without flow control there is little possibility of getting a correct copy of the terminal session. The logging begins when you connect to the remote and is terminated when you escape back to the local Kermit with the ESCAPE character followed by the "C" command.

14.6. Standard Installation

To bootstrap Kermit to the Apple, get the files APP3*xx*.1 thru APP3*xx*.2 on a DOS 3.3 diskette, where *xx* is the current version. Make sure the diskette is a master diskette and empty. Then:

    EXEC APP3xx.1,R25

You will be asked several questions about your hardware and the program should execute and produce a binary, `kermit.help` and `kermit.init` with some starting instructions. If you want to run Kermit on PRODOS simply use the PRODOS conversion routines to move the binary, `kermit.help` and `kermit.init` to PRODOS. If you want other options as a regular thing then change file `kermit.init` with your changes. Then make sure the file is on the same disk as the binary.

Files Supplied for Kermit-65

The following files should be supplied on the distribution tape:

APPAAA.HLP	List of files (like this one)
APP385.1	Easy install file 1 (an exec file for DOS 3.3)
APP385.2	Easy install file 2 (an exec file for DOS 3.3)
APPACC.HEX	Apple com card hex
APPACC.M65	Apple com card source
APPCCS.HEX	CCS 7710 com card hex
APPCCS.M65	CCS 7710 com card source
APPCPS.HEX	CPS com card hex
APPCPS.M65	CPS com card source
APPGS.HEX	GS serial port hex
APPGS.M65	GS serial port source
APPHMM.HEX	Hayes micro modem card hex
APPHMM.M65	Hayes micro modem card source
APPLE.DOC	Complete documentation (it says here)
APPLE.MSS	Scribe text formatter source for documentation
APPMAI.HEX	Main kermit pgm hex
APPMAI.M65	Main kermit pgm source
APPMAK.UNX	Make file for UNIX cross assembly (to assemble Kermit)
APPMSV.HEX	Microtec com card hex
APPMSV.M65	Microtec com card source
APPSSC.HEX	Super serial com card hex

`APPSSC.M65`	Super serial com card source
`APPXAS.1`	65c02 cross assembler for UNIX system part 1
`APPXAS.2`	65c02 cross assembler for UNIX system part 2
`APPXAS.3`	65c02 cross assembler for UNIX system part 3

The syntax of the filenames may vary. On UNIX systems, the filenames will be in lowercase. On VM/CMS systems, the period will be replaced by a space.

Alternate Installation

The main problem exists in getting the hex files onto your diskette as a text file. But again that is a test of your creativity. If you have a version of Kermit running then GET or RECIEVE the file as a text file and you are in business. Since Kermit has been separated into two assemblies then two hex files will have to be present on the diskette. Get the main hex file `APPMAI.HEX` and select which com card hex you will need. First "`exec APPMAI.HEX`". Your Apple (or compatable) will go into monitor and show you `*`'s for several minutes. This is the monitor loading the hex into binary. If you get beeps from the monitor its probably because you didnt get a good copy of the text file. Now EXEC the com card driver you are going to use. You will have to get back into basic(aha another test for you,try "3d0G") to do this. And you will see the monitor loading the com driver. The order of EXEC's is important. The com card should be loaded last. Next get back into basic and do a "`bsave kermit,A$1000,L$6e00`". You may have to specify the drive to do this binary save, with a slot or drive on the end of the BSAVE (aha another test). You now run Kermit via "`brun kermit`".

If you want to customize Kermit for your equipment, the recommended method is to use file "kermit.init" OR do all your SETs, etc, and then do an "exit". Now you should be back in BASIC. At this point do a "`bsave` *name*`,A$1000,L$6e00`" and when you do a "`brun` *name*" all your setups will be remembered. NOTE: If you save your current settings via "`bsave kermit ...`" you may find that moving that binary to another type of Apple (e.g. from a //e to an //e+) will not be possible. So make sure you keep the original binary to move between machine types.

Since the org is now $1000 if you have been using Kermit and then went back to basic for some trivial thing a "CALL 4096" should start up Kermit without having to reload it.

In summary:

1. `EXEC APPMAI.HEX`
2. Choose the com card driver you will use. For example `APPSSC.HEX`.
3. `3D0G`
4. `EXEC APPSSC.HEX`
5. `BSAVE kermit,A$1000,L$6e00`

And you should be in business. Remember there is the command HELP and whenever you are into a command a "?" will give you the posible options available at that point of a command. The escape key will finish typing an option if it is possible. The syntax of all the commands and options only requires enough characters to make that command or option unique.

14.7. Problems

Installation

NOTE: When using the super serial driver you must have the cards sw6-2 on. This allows the card to use interupts. The rest of the switches are set from within Kermit. It appears that you can run your Apple 2 with sw6-2 on and in 99% of the cases will cause no problems. This is because the OS runs with interupts locked out ("sei" in assembly language) and the program must explicitly give a "cli" for interupts to work (the super serial driver does).

The AE Serial Pro must have switches 1 & 3 open and 2 & 4 closed. This appears to dissagree with the documentation since those settings turn off irq interupts and turn on nmi interupts. So watch this it may get corrected in later versions.

The Microtek driver is a super serial look alike which does not run with interupts. If you have trouble with the super serial driver you might try the MSV driver. For you people with the MSV-622c card, you might try running a jumper from the UART 6551 pin 26 to the card edge pin 30. This will enable interupts just like the SSC sw2-6, and then you can use the super serial driver.

The Prometheus card will work with the Apple com driver. However you will have to set the switches on the card for baud etc. Evidently this card can not be programed by the software. If that is not true then here is an oportunity for you to write a better driver. If you do please pass it on for other Prometheus users.

Unconfirmed reports have it that the Apple Cat will work with the Apple com driver. Would appreciate a confirmation.

Some have noted the Apple com card must be initialized via the "IN#x" before starting Kermit. Ike has now updated this driver and the initialization is now done within the Apple com driver. Thanks Ike.

Usage

There is the command HELP and whenever you are into a command a "?" will give you the posible options available at that point of a command. The escape key will finish typing an option if it is possible. The syntax of all the commands and options only requires enough characters to make that command or option unique.

When using flow control you may appear to hang. Type a `^Q` (Control-Q) and that may free you up.

Remember when you use your printer there are a lot of variables here. What was being sent to the screen now is being sent to your printer. If you were emulating the VT52 your printer may not know how to handle the escape sequences, tabs etc. It may be you can tell the host you are a tty or some such device that will give carrage returns etc that your printer can handle. Some printers may require the flow control and delay to get readable printing.

File Transfer Errors

"File Transfer Errors," was added to this document by the Southeast Regional Data Center (SERDAC), ’88 July 17.

In spite of the fact that successful Kermit file transfers are almost always error free, there are a number of circumstances which can corrupt, prevent, or interrupt/abort a transfer. In the case of an actual abort, there may be data loss or corruption, and an incomplete file may not have a correct end-of-file. These circumstances may be roughly divided into two groups: (1) problems due to file or disk errors, and (2) problems due to delays or failures in Kermit packet exchange.

Common problems in category (1) include the following:

(a) improper file specification (b) wrong file type (c) protected file(s) (d) disk problems

(1a) problems can occur when you specify, to either the Apple or host Kermit, a non-existent or improperly located file. Misspelling and/or incorrect (sub)directory specification are popular villains! If you are commanding either Kermit to SEND a file (SEND filespec), the problem will be fairly obvious. On the Apple II, you’ll see an error message like: "FILE NOT FOUND." On the VAX/VMS 8800, for example, you’ll see the message: "%KERMIT32 ....., file not found for ’filespec’". In either case, the transfer will not take place. If you’re using Kermit-65 to GET (GET filespec) files from the VAX/VMS Kermit server, and the requested file does not exist in your VAX default directory, you should see a Kermit-32 generated "REMOTE MESSAGE %KERMIT32 ....., file not found for ’filespec’" appear in the transmission status display, and then the Kermit-65 message "CANNOT RECEIVE

FILE-HEAD". Transfer of the questionable file will not take place.

(1b) problems can occur if you forget to specify, to either the Apple or host Kermit, what type of file you wish to transfer. If you are using Kermit-65 to send files to a host, you are fairly well protected against this error. If you attempt to send a file whose CATALOG type does not match the FILE-TYPE parameter setting, you will receive a "INCOMPATIBLE FILE FORMAT" error message, or something similar, and the transfer will not take place. If, however, you are receiving (via RECEIVE or GET) a file whose native type does not match the FILE-TYPE setting, the file WILL be received. It will be mis-typed (according to the FILE-TYPE setting), though, and any later attempt to use it on the Apple will probably be unsuccessful.

The same sort of circumstances generally apply for a host Kermit. With the VAX 8800, for example, when Kermit-32 is sending a file, you generally need not worry about setting its file type. When Kermit-32 is receiving a file, however, properly setting its file type is very critical. If you wish to put Kermit-32 in server mode to receive multiple files, set the file type BEFORE using the SERVER command, and make sure that you only send it the appropriate type of files during that server session. You cannot switch file types DURING a given server session!

NOTE: One other way you can get into trouble with "wrong file type" is by trying to send a file which is mixed--mostly text, but with some imbedded characters that are not true 7-bit ASCII (i.e., ASCII codes 00-127). This often happens when you are trying to transfer a file which is word processor output. Most word processing software claims to allow you to output a true ASCII or text file, but in some cases it really does not, and in others the choice of output options is confusing. If you have set up either Kermit program to send/receive a text file, and you try to transfer illegal ASCII characters (codes 128-256), your transfer may "hang" or be aborted. At the very best, if the transfer "works," the suspect characters will later probably be meaningless or confusing to the destination machine.

(1c) problems can occur in two ways on the Apple II. If your default drive disk is write protected, and you attempt to receive a file, you will receive a "WRITE PROTECTED" error message, and no transfer will take place. If you have set Kermit-65's FILE-WARNING parameter to OFF (normally NOT a good idea), and you attempt to receive a file that already exists in a locked state on your default diskette, you will receive a "FILE LOCKED" error message (if the file is very short, you may have to check with a Kermit-65 STATUS command to see the error message), and no transfer will take place.

Similar problems may occur on the host because of various file protection schemes. On the VAX/VMS 8800, for example, Kermit-32 cannot send out a file that you are unauthorized to read. And, it cannot receive a file unless you are authorized to write to that filename and its (sub)directory. If you use Kermit-32 to attempt to SEND (SEND filespec) a protected file, you should see a "%KERMIT32 ....., insufficient privilege or file protection violation for 'filespec'" error message, and no transfer will take place. If you have Kermit-32 in server mode, and you are trying to GET a protected file from it, or you are trying to SEND it a file whose space is protected, you should see a similar Kermit-32 generated REMOTE MESSAGE appear in the transmission status display, and then, on GET, the Kermit-65 message "CANNOT RECEIVE FILE-HEAD". Transfer of the protected file will not take place.

(1d) problems are most likely to occur because of Apple II diskette or drive problems. The following conditions will generate "DISK I/O" or "I/O ERROR" messages when Kermit-65 transfer commands are entered: bad diskette in default drive, no diskette in default drive, default drive door open, and/or unINITialized disk in default drive.

If any of those errors are detected before the attempted transmission of a given file, the transfer of that file will not begin. If any are detected DURING a file transmission, the file transfer will likely abort; at best transmitted data will be incomplete. Data which does reach the destination end of an aborted transfer should be considered very suspect; the disk problem should be corrected and the transfer should be repeated! (The best chance you have for salvaging text file data in an abort is if the file destination is the host machine and you have told its Kermit to save incomplete files, e.g., on the VAX-8800, you need to SET INCOMPLETE KEEP).

One other Apple II disk problem can be encountered while you are using Kermit-65 to receive files. If you exceed the storage capacity of your diskette during a RECEIVE or a GET, you should see a "DISK FULL" error message. Data that has been received up to the point of the overflow will be automatically DELETEd. Make CERTAIN that you do not try to receive any more files until you have DELETEd some files from the problem diskette, or until you have replaced it with one that has adequate capacity to receive the complete file. NOTE: See Section 1.5.4.

It is less likely that (1d) problems would occur because of host machine disk problems. The most likely circumstance you might encounter on the VAX/VMS 8800, for example, would be in receiving a large file and, in the process, exceeding your VAX disk quota. In such a case, you should see an appropriate Kermit-32 generated REMOTE MESSAGE appear within the Kermit-65 transmission status display. If this happens, delete some files from your VAX (sub)directories, and/or have your VAX disk quota increased BEFORE you try the transfer again. If you have issued a SET INCOMPLETE KEEP command to Kermit-32, there may be some chance of salvaging text file data that arrived before the disk quota overage, but the best thing you can do is to repeat the transfer!

As a general rule, if some disk or file error prevents a transfer from beginning, to get it to "go," you will need to correct the error and repeat all the steps that preceded it.

If you are still commanding the host Kermit, and you see an error message, you will have to get the host Kermit's prompt back and give it an acceptable command. If you have commanded the host Kermit to SEND or RECEIVE, and are back commanding Kermit-65 when you notice the error, you will have to correct the problem, CONNECT back to the host, get the host Kermit prompt (with the VAX/VMS 8800, try typing RETURN or CTRL-Y), and repeat the SEND or RECEIVE command, before returning back to Kermit-65 to command it again.

If you have placed the host Kermit in server mode, and are giving Kermit-65 commands when you notice an Apple disk/file error prevents a file transfer from starting, chances are good that you won't have to CONNECT back to the host. It is also important to note that within a single server session, when you are transferring multiple files, all files transferred PRECEDING an error (or abort) are probably good. To repeat the transfer, correct the error, and give Kermit-65 the appropriate command to transfer the file that messed up. The first time you do it, you may get back a message like "REMOTE MESSAGE %KERMIT-32..... protocol error" This is just the host server trying to get back "on track" after the error. When the Kermit-65> prompt returns, enter the transfer command again, and it will probably be accepted.

If the second attempt should fail, wait for the Kermit-65> prompt, enter: FINISH, wait for the prompt again, and enter: CONNECT. If you do not see the host operating system prompt ($ on the VAX 8800), type a few RETURNs (or on the VAX/VMS a CTRL-Y). Re-invoke the host Kermit and put it back in server mode.

If disk or file errors prevent a transfer from completing, recovery will depend on the error, whether you had the host Kermit in server mode or not, and on your desire for accuracy.

Some disk/file error aborts are "fatal" (e.g., Apple DISK FULL, and uploading to the VAX 8800 w/o having commanded Kermit-32 to SET INCOMPLETE KEEP). The destination file will be destroyed. The transfer of the file will have to be repeated again from the beginning. Again, unless you have set the host Kermit for server mode, you will have to CONNECT back, get the host Kermit prompt, and re-command it. If you were in a server session, though, you can probably repeat the transfer of the interrupted file without going back to the host (see recovery procedures above).

Other disk/file errors that interrupt/abort a transfer may leave salvageable text data at the transfer destination. The best policy, though, is to repeat the transfer of the incomplete file (see recovery procedures above).

Common problems in category (2) include the following:

(a) bad parity

(b) noisy communications line

(c) timeout due to delays, "disaster," etc.

(d) Kermit-program incompatibility

(e) user error

(2a)

Parity settings are very critical to correct transfers. If you do not inform Kermit-65 of the correct parity being used by the remote host machine or the communications path to it, "checksum" error checking calculations will be wrong, and packets will be consistently rejected when they arrive at their destination. In particular, most binary file Kermit transfers won't get very far if parity is not set correctly.

[NOTE: If you want to do a binary file transfer (Apple binary or BASIC files) via a FIRN Network connection to the SERDAC VAX/VMS 8800, you must SET PARITY SPACE before the transfer is initiated; that will insure that eight-bit quoting is used. If you dial directly into the VAX/VMS 8800, SET PARITY NONE; eight-bit quoting (which is less efficient) is not required].

(2b)

Line noise can be the root cause for a variety of file transfer problems. The beauty of a "packetized protocol transfer" scheme like Kermit is that ordinarily, the scheme will overcome an occasional burst of line noise. A packet which arrives out of sequence, or which does not have the same checksum "bit count" as when it was sent, will get retransmitted, and the noise induced data error will correct itself.

Sometimes, however, bad line noise can outwit even the cleverest aspects of Kermit. There are some times where severe noise can corrupt the "checksum" error checking and lead to undetected transmission of a bad character (assuming that the severe line noise exists, chances of this happening for one character are, for Kermit-65 error checking, less than two percent).

If line noise is bad enough and persistent enough, it is also a cause for several problems that will eventually "hang" or totally confuse and abort a transfer:

Each transfer is preceded by the Kermit-to-Kermit exchange of several short "initialization packets. These tell the controlling programs critical things to expect about the upcoming transfer. If line noise prevents the packets from arriving, or scrambles them up, the transfer probably can't get started correctly.

One of the biggest vulnerabilities of the Kermit scheme is that each arriving packet must be acknowledged (ACK) by the receiver, and that the sender must actually receive back the acknowledgement (likewise, if an expected packet does not arrive, there often must be a negative acknowledgement (NAK)). Since the ACK/NAK packets are very short, they are rather vulnerable to severe noise. If too many of them are scrambled or lost, the transfer can get out of synch, and the transferring programs can lose track of where they are.

One other place Kermit is vulnerable is in the beginning of a data packet. The first several bytes of these longer packets are reserved for control information: packet type, byte count, sequence number, etc. If line noise repeatedly coincides with the transmission of this control information, it is very easy for the transfer to get confused--particularly if the packet numbering gets garbled.

If you detect frequent line noise after you've connected to a host, but before you begin transfers (you will probably see extraneous junk characters appearing on your screen), you're probably in for trouble. Once transfers actually begin, line noise problems are often characterized by incrementing of the RETRY counter on the Kermit-65 transmission status display, and/or by long pauses in incrementing of the status display byte counter.

To minimize line noise, first see if there are any obvious loose connections in your equipment (telephone line connection to wall box, telephone line to modem, modem cabling to serial connector, or, if appropriate, cabling from hardwire port to serial connector). If not, you may want to hang up and redial to get another telephone connection (almost every connection is unique, and you may get a better one than you had). Many line noise problems will clear up with those simple remedies, but some may be beyond your control!

If all else fails, you may also try shortening the maximum length of your data packets (SET SEND/RECEIVE PACKET-LENGTH) to possibly lessen the effects of persistent noise.

(2c)

A Kermit transfer consists of a regular and predictable exchange of initialization, data, and, ACK/NAK packets. If something (line noise, busy computer, user error, etc.) interrupts or delays this regular exchange, there must be a way for a Kermit program on at least one end to figure out something is wrong and try to get the packet exchange back on track again.

This is usually done with a timer and retry mechanism. If a Kermit does not receive an expected packet, within its timer's time limit (a timeout), it will resend its last sent packet to try to "wake up" the other Kermit (effectively by asking it to send its last packet again). This resending is repeated ("retried") a number of times before the program assumes it cannot get things on track again. Each packet resent by Kermit-65 is counted as a RETRY on its transmission status display. If Kermit-65's retry count exceeds 20, it will try to issue an error message according to what kind of packet it was waiting for and/or it will say MAX RETRY COUNT EXCEEDED. The transfer will then be aborted.

Very frequently, timeouts are caused by unexpected delays in the remote computer, or in the network thru which you connect to it. If you know that the host machine or network is very busy, and you repeatedly have aborted transfers due to timeouts, you may be able to alleviate the problem by increasing the value of the default Kermit-65 receive timeout parameter (SET RECEIVE TIMEOUT).

Other common ways that Kermit-65 can timeout and abort are: (1) if the host machine "goes down" during a transfer, (2) if the telephone, network, or hardwire connection is completely broken during a transfer, (3) if you forgot to "start up" the host Kermit and give it a transfer command (SEND, RECEIVE, or SERVER) BEFORE giving Kermit-65 a transfer command, and (4) if (2a), (2b), (2d), or (2e) problems occur and critical initialization packets are never received.

In cases (1) and (2), you will eventually probably see a CANNOT RECEIVE DATA or MAX RETRY COUNT EXCEEDED message from Kermit-65. Cases (3) and (4) may result in a CANNOT RECEIVE INIT message.

(2d)

To do effective Kermit transfers, there must be two Kermit programs working-- one on either end of a "computer connection." In addition, the two Kermits must be able to "talk to" each other in a prescribed, standard way. Although there are specific standards for writing all Kermit programs, most of them have been written by volunteers and are in the "public domain." The protocol requirements and resultant programs are generally rather complex, and it is all too easy to inadvertently program in a subtle error in a given Kermit version. Additionally, there are many "levels of ability" of Kermit programs: some can operate in server mode, some cannot. Some can transfer binary files; some cannot, etc. Unless the Kermit programs you are using are both error free, and both have the same capabilities for the transfers you wish to perform, you are in trouble!

If there is a systematic "bug" in one of the Kermit programs, or if you are asking one Kermit to do something the other can't do, there will usually be a problem with packet exchange; in many cases the requested transfer will not even get started. You may see a Kermit-65 error message, on the transmission status display, saying that a packet was not received, or a REMOTE MESSAGE saying a packet was unexpectedly received, or one that the command cannot be executed by the other Kermit. In some cases, you may see no explanatory error messages at all; the transfer will just "hang" and will probably eventually "timeout" and abort (MAX RETRY COUNT EXCEEDED).

(2e)

If you've read about category (1) errors above, you can see that there are a variety of things you can do to with files or disks to mess up a Kermit transfer. You can also wreak havoc by issuing improper or illegal commands to Kermit programs. Before trying to transfer a lot of files, or trying out a new type of transfer, be sure you understand the procedure you need to follow and the various Kermit commands that will be involved.

New Kermit users often try to command their local Kermit program (e.g., Kermit-65) to send or receive a file, without having first invoked and commanded the host Kermit.

Another common error is to issue improper commands to a remote server. For example, when VAX/VMS Kermit-32 is in server mode, and you are requesting files from it via Kermit-65 commands, you cannot use a RECEIVE command; you must instead use GET.

As with Kermit program incompatibilities, illegal or inappropriate commands will often cause a problem with packet exchange; in many cases the requested transfer or action will not even get started. You may see a Kermit-65 error message, on the transmission status display, saying a packet was not received, a REMOTE MESSAGE that a packet was unexpectedly received, or one that the command cannot be executed by the other Kermit. In some cases, you may see no explanatory error messages at all; the transfer will just "hang" and will probably eventually "timeout" and abort (MAX RETRY COUNT EXCEEDED).

Except for the fact that you will probably never note a category (2) "packet exchange" error while you are "talking to" the remote system or commanding its Kermit, and that the remedies you must employ to correct the errors will be different, recovery procedures to get your file transferred correctly will be much the same as those we described at the end of the discussion on category (1) "disk/file" errors. Make sure to read that discussion for more details than we have included below.

In short, if an error prevents a given transfer from actually beginning, you will need to correct the error and repeat all the steps that preceded it. This will be more difficult if you are transferring only one file-- having commanded the remote Kermit to SEND or RECEIVE. If you have placed the remote Kermit in server mode, and an error prevents the transfer of one file, all files transferred up to that point are probably OK, and you can usually correct the problem, and get a transfer started again without having to reCONNECT back to the host.

If you are transferring a text file, and an abort occurs in mid-transfer, some data may be salvageable in the destination file, but the best rule with any type of file is to repeat the transfer, in which case the recovery procedures in the last paragraph apply.

14.8. Customizing Kermit-65

The source code to Kermit-65 is in 6502 Assembler. It has been formatted for a cross assembler which runs on a unix 2's complement machine. Files `appxas.1` thru `appxas.3` are the cross assembler for UNIX. Get the files on a UNIX system and then look at the documentation at the start. They will easily make you a xasm for Kermit. The file `appmak.unx` is the makefile to use with the xasm to reassemble all of Kermit's parts.

Kermit-65 3.xx has been separated into two assemblies, the main routines and the com card routines for the devices shown in Table 14-1. A vector has been set up in low memory for the two assemblies to communicate. Look at the working com drivers for tips on how to incorporate your version of the com driver. some things to note: It is probably best to buffer the input from the remote and to get input characters from the remote every chance you get. Note the Microtek SV-622 driver, whenever the input is checked for a character and has a character the character is put into the buffer immeadiatly. Also when the output is checked for ready to output, if the card is not ready to output then it is checked for a character to input. All this should help prevent losing characters.

Communications card vector area:

```
address size   module    function
1003    byte    main      This is the baud rate index as follows:
```

```
                                 3 - 110
                                 4 - 135.4
                                 5 - 150
                                 6 - 300
                                 7 - 600
                                 8 - 1200
                                 9 - 1800
                                10 - 2400
                                11 - 3600
                                12 - 4800
                                13 - 7200
                                14 - 9600
                                15 - 19200
                                for example:
                                if index is 6 then line should be 300 baud
1004     byte                   unused
1005     word    driver         Address of a null terminated string.
                                address should point to a capitalized
                                string of the drivers id
1007     byte    main           Com slot in the form $n0 where n is slot #.
1008     byte    main           Force initialization flag when 0.
                                init routine should always initialize when
                                this flag is 0 & then set flag non-zero.
1009     word    main           Address of the end of Kermit main routine.
100b     byte    main           Flow control is on when high bit is set.
100c     word    driver         Address of the end of the com driver.
1020   3 bytes   driver         Jump to initialization routine.
1023   3 bytes   driver         Jump to command routine.  A reg has command
                                  0 - hang up the line
                                $0b - set baud rate
                                $0c - set break on the line
                                $91 - do xon on the line
                                $93 - do xoff on the line
                                routine returns false (P reg zero flag)
                                if unable to do command.
1026   3 bytes   driver         Jump to check for input from the line.
                                routine returns false (P reg zero flag)
                                if no character on line
1029   3 bytes   driver         Jump to get input character from line.
                                routine returns character in A reg
102c   3 bytes   driver         Jump to put character in A reg on line.
102f   3 bytes   driver         Jump to reset com driver.
1040   3 bytes   main           Jump to Apple ROM wait rtn. microsecs delay
                                =1/2(26+27A+5A*A) where A is the accumulator
1043   3 bytes   main           Jump to routine to print null-terminated string.
                                X reg contains least significant byte of address
                                Y reg contains most significant byte of address
                                routine does not issue a carriage return.
1046   3 bytes   main           Jump to routine to read the keyboard.
                                A reg contains the character read
1049   3 bytes   main           Jump to routine to print carrage rtn & line feed.
104f   3 bytes   main           Jump to routine to set characters parity.
                                A reg contains the character before and after.
```

All the routines should return with the "rts" instruction. Routines which can return a true/false indication should return with the P reg zero flag set appropiately. That is: a "beq" instruction will branch on a false indication and the "bne" will branch on a true indication. The com driver should start its routines above the main routines and tell where the end of the com driver is via location $100c. If your com driver gets too large then the bsave address would have to be changed when you are saving the binary to diskette.

15. CP/M-80 KERMIT

Program: Bill Catchings, Columbia University, with contributions from Charles Carvalho (ACC), Bernie Eiben (DEC), Nick Bush (Stevens), John Bray (University of Tennessee), Bruce Tanner (Cerritos College), Greg Small (University of California at Berkeley), Kimmo Laaksonen (Helskini University of Technology), Brian Robertson (Aberdeen University), A.J. Cole (Leeds University), John Shearwood (Birmingham University), Tony Addyman (Salford University), Godfrey Nix and Martin Carter (Nottingham University), Ian Young (Edinburgh University), Chris Miles (Manchester University), Richard Russell, Dave Roberts, and many, many others.

Language: 8080 Assembler, M80, or MAC80

Version: `4.09`

Date: 11th January, 1988

Documentation:
Bertil Schou, Loughborough University (with lots of help from Jon Warbrick of Plymouth Polytechnic); Christine Gianone and Frank da Cruz, Columbia University; Charles Carvalho, ACC; many others.

KERMIT-80 Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Partial, Auto-receive only
Login scipts:	No
Transfer text files:	Yes
Transfer binary files:	Yes
Wildcard send:	Yes
File transfer interruption:	Yes
Filename collision avoidance:	Yes, poor
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	No
Alternate block checks:	Yes
Terminal emulation:	Yes, VT52 and others
Communication settings:	Yes; duplex, parity
Support for dial-out modems:	No
Transmit BREAK:	Yes; most versions
IBM communication:	Yes
Transaction logging:	No
Debug logging:	No
Session logging:	Yes
Raw file transmit:	Yes
Act as server:	No
Talk to server:	Yes; SEND, GET, FIN, BYE
Advanced commands for servers:	No
Command/init files:	Yes
Command macros:	No
Local file management:	Yes; DIR, ERA, TYPE, PRINT, COPY
Handle file attributes:	No
Extended packets:	No
Sliding Windows:	No
Printer control:	Yes, better, but not perfect

15.1. Summary of CP/M

There are essentially two versions of CP/M - Versions 2.2 and 3.0 (sometimes also called CP/M PLUS.)

CP/M-80 Version `2.2` is run in a single 64 Kbyte "page", usually the largest amount of memory on Z80 or 8080 systems. The BIOS (Basic input/output system), BDOS (Basic Disk Operating System) and CCP (Command console processor) all share memory with any transient program the user may wish to run. Some basic commands are available through the CCP, like DIR, ERA etc,while others are loaded from disk into the transient program area and run as a program, like PIP or STAT.

CP/M Version `3.0` (or CP/M PLUS) effectively removes the requirement of having the CCP and BDOS along with a chunk of the BIOS code being resident in the single 64k byte page of memory. This allows even more space for programs in the TPA, but still a little less than the maximum of 64k. It is substantially different from CP/M version `2.2`, with lots of added features. Kermit-80 uses very few additional version `3.0` features, and only where absolutely necessary.

CP/M file specifications are of the form `DEV:XXXXXXXX.YYY`, where

`DEV:` is a *device name*, normally the A: or B: floppy. If omitted, the device name defaults to your connected diskette.

`XXXXXXXX` is a *filename* of up to 8 characters.

`YYY` is the *file type*, up to 3 characters.

File names and file types may contain letters, digits, and some special characters, including dash, dollar sign, and underscore, but no imbedded spaces. Upper and lower case letters are equivalent.

"Wildcard" file-group specifications are permitted in file names and file types (but not device names) within certain contexts; a "`*`" matches a whole field, a "`?`" matches a single character, including space. Examples: "`*.F??`" specifies all files whose *types* start with F and are 1, 2, or 3 characters long; "`F?.*`" specifies all files whose names start with F and are no more than two characters long (before the trailing spaces).

The five CP/M commands are:

DIR *file* Lists the the names of the specified files. The default file specification is "*.*". Example: "`DIR B:*.FOR`".

ERA *file* Erases (deletes) the specified file(s); wildcards allowed.

REN *new old* Changes the name of a file from *old* to *new*, e.g. "`REN NEW.FOR=OLD.FOR`".

SAVE Saves the specified number of memory blocks into a file. (Not on CP/M Plus systems)

TYPE *file* Types the specified file on the screen, e.g. "`TYPE FOO.TXT`".

The most important programs are:

STAT Gives statistics on disk usage; sets and displays IOBYTE. (Not on CP/M Plus systems)

PIP <u>P</u>eripheral <u>I</u>nterchange <u>P</u>rogram. Copies files. In response to the "`*`" prompt, give a command of the form

```
disk:outfile=disk:infile
```

Wildcards ("`*`" for a whole field or "`?`" for a letter) can be used. Examples: "`A:=B:*.*`" to copy a whole disk, "`A:=B:*.FOR`" to copy all the Fortran programs from disk B to disk A. If the disk specification is omitted, your "connected" disk is assumed. Command line arguments are also accepted, e.g. "`PIP A:=B:*.*`".

There are equivalent commands for CP/M Version 3.0, but are not loaded into memory in the same way as for CP/M Version 2.2. For further information on CP/M, consult your microcomputer manual or a CP/M handbook.

15.2. Kermit-80 Description

Since Kermit-80 runs on a standalone micro, it is always in control of the screen -- it is always *local*. Thus, it always keeps the screen updated with the file name and the packet number, whether sending or receiving.

Kermit-80 is capable of an imprecise or "fuzzy" timeout on an input request, and can break deadlocks automatically. In most cases, this is not important, because the KERMIT on the other side is most likely able to handle the timeouts. The timeouts done by Kermit-80 are fuzzy because they depend on the speed of the processor and other factors that can vary from system to system.

If, despite the timeout capability, the transmission appears to be stuck (and you can tell that this has happened if the screen fails to change for a while) you can type carriage return to have the micro do what it would have done on a timeout, namely NAK the expected packet to cause theforeign host to send it again (or, if the micro is sending, to retransmit the last packet). Micro/micro or micro/IBM-mainframe transfers could require this kind of manual intervention.

File transfers may be interrupted in several ways.

Control-C
: This will return you to Kermit-80 command level immediately, so that you can connect back to the remote system, or take any other desired action.

Control-X
: When sending a file, this will terminate the sending of the current file with a signal to the KERMIT on the other side to discard what it got so far. If there are more files to be sent, KERMIT-80 will go on to the next one. When receiving a file, KERMIT-80 will send a signal to the remote KERMIT to stop sending this file. If the remote KERMIT understands this signal (not all implementations of KERMIT do), it will comply, otherwise the file will keep coming. In any case, the remote KERMIT will go on to the next file in the group, if any.

Control-Z
: Like Control-X, except if a file group is being transmitted, this will stop the transmission of the entire group. If only a single file is being transmitted, it works exactly like Control-X.

Carriage Return
: If you type a carriage return Kermit-80 will resend the current packet. You may do this repeatedly, up to the packet retry limit (somewhere between 5 and 16 times) for a particular packet.

Kermit-80 Commands

KERMIT-80 uses the DECSYSTEM-20 keyword style command language. Each keyword may be abbreviated to its minumum unique length. "?" may be typed to request a menu of the available options for the current field at any point in a command. ESC may be typed at any point in a command to fill out the current keyword or filename; if sufficient characters have not been typed to identify the current field uniquely, KERMIT-80 will sound a beep and allow you to continue from that point.

BREAK
: Send a Break condition to the remote computer. This is only possible if your system is capable of sending breaks. It is intended to be used with PAUSE, STRING etc and the TAKE command to do weird and wonderful things, like automatic logging on to a remote host.

BYE
: When talking to a remote Kermit Server, this command shuts down the server and logs it out, and also exits from Kermit-80 to CP/M command level.

CONNECT
: Establish a "virtual terminal" connection to any host that may be connected to the serial port, i.e. pass all typein to the serial port and display all input from the serial port on the screen. Also, emulate a DEC VT52 to allow cursor control, screen clearing, etc., if VT52-EMULATION is ON (see below), in which case you should also set your terminal type on the remote host to VT52. (Some versions emulate other terminals.)

  Warning: VT52 emulation is only successful if your system or its attached terminal can do the same sort of functions as a genuine VT52. Things to beware of are cursor addressing, clear to end of page and end of line, clear screen, home cursor, and clear-and-home functions. The useability of VT52 emulation depends entirely on how many of the VT52 functions can be emulated by your terminal.

The escape character differs from micro to micro; when you issue the CONNECT command, the micro will print a message telling you how to get back. The escape sequence is generally an uncommonly-used control character, like CTRL-backslash or CTRL-rightbracket, followed by a single letter "command":

C Close Connection, return to `Kermit-80>` command level.
S Display Status of connection, but maintain remote connection.
`?` List available single-character commands.
0 (zero) Send a null (0) character.
B Send a BREAK signal. Most systems provide this function.
D Drop the line. Used on the Apple with modem. Automatically closes the connection after dropping the line. The TORCH system acknowledges this command but does nothing.
P Toggle printer on or off. Allows you to copy whatever goes to the screen to the printer.
S Temporarily suspend logging to the log file.
Q Restart logging to the log file
`^]` (or whatever - a second copy of the escape character) Send the escape character itself to the remote host.

DIRECTORY
This provides a directory listing of the specified files. If no files are specified, all files on the default disk are listed. File sizes, in K, are included. You may interrupt the listing at any time by typing any character. The listing (even if interrupted) concludes with a display of the amount of free storage left on the disk. You can inhibit the display of file sizes by SET DIRECTORY OFF.

ERASE *filespec*
This executes the CP/M ERA command on the specified file(s). The names of the files being erased are not displayed.

EXIT Quit back to CP/M. The return is made by a JMP 0 (Warmstart)

FCOPY *source destination*
Copy a named file to another file, either on the same drive or another drive.

FINISH Like LOGOUT, but shuts down the remote server without logging it out. Leaves you at Kermit-80 command level; subsequent CONNECT commands will put you back at host system command level.

GET *filespec (local_filespec)*
When Kermit-80 is talking to a Kermit Server on the host, you should use the GET command to request the server to send files to you, for example:

```
get hlp:k*.hlp
```

You may specify a local filename if you want to save the remote file under a different filename. Limitation: If you request an alternate block check type using the SET BLOCK command, the GET command will not communicate it to the remote server. If you want to have type 2 or 3 block checks done when getting files from the server, you have to issue the appropriate SET BLOCK command to the remote KERMIT before putting it in server mode.

HELP List all these commands, with a short description on what the commands do. A question mark will do the same. If you have already typed a command but do not know what the parameters are, type a space (to indicate the end of the command) and a question mark. You will be informed of what Kermit can expect at that stage.

INPUT *time_delay string*
Setup a string and time delay for your CP/M system to expect from the host, then wait for that sting to be sent to your CP/M-80 system.

LOG *filespec*
When CONNECTed to a foreign host as a terminal, log the terminal session to the specified diskette file. This functionality depends to some extent on the remote host's ability to do XON/XOFF flow control, and does not guarantee a complete transcript (after all, that's what the KERMIT protocol is for). The log file is closed when the connection is closed by typing the escape character followed by the single-character command "C".

LOG (append)

It is possible to temporarily suspend logging during connect state. Typing an escape sequence can turn file logging on (<escape-character> R for Resume) or off (<escape-character> Q for quiet).

Re-entering connect state will re-open the previously opened log file and append to that file.

LOGOUT
Like BYE, but leaves you at Kermit-80 command level.

PAUSE *delay_period*
If this command is issued your CP/M system will wait a while before proceeding with another command. This is intended for use in TAKE commands, where you may want to pause for a while before proceeding with the rest of the TAKE file. The actula delay is very variable between systems, and values should be determined on a trial and error basis.

PRINT Print a file to the console and printer. Output to the printer is buffered by the Kermit maintained printer buffer This routine is identical to TYPE but characters are echoed to the printer as well as to the screen. Suspending and aborting output is as described in TYPE.

RECEIVE *filespec*
Receive file(s) from the remote Kermit, and save them under the names provided in the file headers supplied by the remote host. If a local filespec is given, the file is saved under the given filename. If the names aren't legal, use as many legal characters from the name as possible (see the description of SET FILE-WARNING below). If there's a conflict, and FILE-WARNING is ON, warn the user and try to build a unique name for the file by adding "&" characters to the name.

SEND *filespec*
Send file(s) specified by *filespec* to the remote Kermit. The *filespec* may contain CP/M wildcards.

SET *parameter* [*value*]
Set the specified parameter to the specified value. Possible parameter settings:

AUTORECEIVE
ON (or OFF). Allows several files to be received without having to type RECEIVE on the receiving machine. The routine simply looks for activity on the serial line, and if so fudges a RECEIVE command. The packet sent by the sender will be lost.

BAUD-RATE *value*
Change the baud rate of the communications port. This command only works on some systems. *value* is the numeric baud rate (300, 9600, etc.) desired. Type SET BAUD followed by a question mark for a list of supported baud rates. On systems that do not support this command, you must set the port baud rate from CP/M or other setup mechanism outside of KERMIT-80.

BLOCK-CHECK-TYPE *option*
The options are:

1-CHARACTER-CHECKSUM
Normal, default, standard 6-bit checksum.
2-CHARACTER-CHECKSUM
A 12-bit checksum encoded as two characters.
3-CHARACTER-CRC-CCITT
A 16-bit CCITT-format Cyclic Redundancy Check, encoded as 3 characters.

BUFFER-SIZE *value*
This allows you to set a buffer size during transfer of data. On some systems it takes so long that the remote end times out while the local system is reading or writing to disk. The size is the number of 128 disk sectors (nominal) and can be from 1 (128 bytes) to 64 (8 kbytes).

CASE-SENSITIVITY ON (or OFF)
Kermit-80 usually maps lower case characters to upper case character, but there may be times when lower case should be left alone. SET CASE-SENSITIVITY ON will do this-but only for some functions.

CP/M-80 filenames will still be mapped to uppercase characters.

DEBUG ON (or OFF). Enables/disables displaying of packets on the screen during file transfer. Not performed if the QUIET option has been set for the terminal (ie SET TERMINAL QUIET)

DEFAULT-DISK *drive letter*
This allows you to set the default disk as source and destination of file transfers. In addition, issuing this command causes you to switch to the specified disk and log it in, write-enabled. The colon must be included in the disk name (A:). The selected disk appears in your KERMIT-80 prompt, for instance

```
Kermit-80 14A:>
```

DIRECTORY-FILE-SIZE ON (or OFF).

By setting DIRECTORY-FILE-SIZE OFF you can get an abreviated listing of your disk drive. File sizes are not calculated, and five files are shown on a line. Setting this option ON will show file sizes of each file.

Both options will list the free space remaining.

ESCAPE Change the escape character for virtual terminal connections. Kermit-80 will prompt you for the new escape character, which you enter literally.

FILE-MODE *option*
Tells KERMIT-80 what kind of file it is sending, so that KERMIT can correctly determine the end of the file. SET FILE BINARY means to send all the 128-byte blocks (ie logical CP/M sectors) of the file, including the last block in its entirety; SET FILE ASCII is used for text files, and transmission stops when the first Control-Z is encountered anywhere in the file (this is the CP/M convention for marking the end of a text file).

SET FILE-MODE DEFAULT tells Kermit to attempt to determine the file type by examining the file being transmitted. If a Control-Z appears before the last block of the file, it is assumed to be BINARY; if, when the first Control-Z is encountered, the remainder of the file contains only control-Z's, it is assumed to be a text file. Unfortunately, not all programs fill the remainder of the last record of a text file with Control-Z's, so this algorithm is not always successful.

If binary transmission is used on a text file, or a compressed file (eg a .DQC file) some extraneous characters (up to 127 of them) may appear at the end of the file on the target system.

If ASCII transmission is used on a binary file, any 8th bits set will be stripped and a warning sent to the console. When the first control-Z is encountered, the file is assumed to be at the end, even if it is not.

FLOW-CONTROL ON (or OFF)
Sets XON/XOFF flow control on or off. If set ON the host is expected to respond to an XOFF or XON sent by Kermit-80. If set off, no flow control is assumed and any XON/XOFF is ignored.

IBM ON (or OFF)
Allow the transfer of files to and from an IBM mainframe computer. This makes Kermit-80 wait for the IBM turnaround character (XON), ignore parity on input, add appropriate parity to output, and use local echoing during CONNECT. As distributed, KERMIT-80 uses MARK parity for IBM communication. If you don't give this command, IBM mode is OFF. Since IBM VM/CMS KERMIT does not have timeout capability, SET IBM ON also turns on the "fuzzy timer" automatically.

LOCAL-ECHO ON (or OFF)
When you CONNECT to a remote host, you must set LOCAL-ECHO ON if the host is half duplex, OFF if full duplex. OFF by default.

LOGGING ON (or OFF)
Cease or resume logging whenever connect mode is entered. This is really only applicable after a LOG command is no longer required.

NO-EXIT
This command is applicable only for Kermit initiated with a command tail. For example, if Kermit was initiated by:

KERMIT ;SEND HELLO;NO-EXIT

Kermit would first seek out and execute the KERMIT.INI file (if present), then send file HELLO to a remote system. Usually Kermit would exit back to CP/M, but NO-EXIT over-rides this.

Note the leading semicolon. This clears leading spaces from the first command.

PORT *port name*
: Allows you to switch between different communication ports. This command is not available on all systems. Type `SET PORT ?` for a list of valid options for your system. (Note: If your system does not support several ports, this command will return a "Not implemented" error if you try to set a port.)

PRINTER
: ON (or OFF)
Turns copying of CONNECT session to printer on and off. It is also possible to toggle the printer on/off from the connect state, by typing <escape character> followed by P.

PARITY *option*
: Sets parity for outgoing characters to one of the following: NONE, SPACE, MARK, EVEN, or ODD. On input, if parity is NONE, then the 8th bit is kept (as data), otherwise it is stripped and ignored. The parity setting applies to both terminal connection and file transfer. If you set parity to anything other than none, KERMIT-80 will attempt to use "8th bit prefixing" to transfer binary files. If the other KERMIT is also capable of 8th bit prefixing, then binary files can be transferred successfully; if not, the 8th bit of each data byte will be lost (you will see a warning on your screen if this happens).

RECEIVE *parameter* [*value*]
: Set a RECEIVE parameter.

  PAD-CHAR
  : Set the PAD character to use while receiving files. Currently a dummy, as for SET SEND PAD-CHAR.

  PADDING [value]
  : Set the number of PAD characters to use while receiving files. Same as SET SEND PADDING.

  START-OF-PACKET [value]
  : Set the default start of Packet character for receiving files. Apply the same rules and considerations as for SET SEND START-OF-PACKET.

SEND *parameter* [*value*]
: Set a SEND parameter.

  PAD-CHAR
  : Set the Pad character to be used while sending files. It is currently a dummy entry, and does not do anything.

  PADDING [value]
  : Set the number of PAD-CHARS to be used while sending files. This too does nothing.

  START-OF-PACKET
  : Set the default start of packet character to another character than control-A. This may be necessary on systems (including intervening networks) that trap control-A characters. Choose a control character not otherwise used, ie not carriage return (13D, ODH), line feed (10D, OAN), tabs (09D, 09H), backspace (08H), and bell (07H) or any other used between you and your remote system.

TACTRAP
: set the TAC intercept character. If you are attached to a TAC it will swallow the intercept character (commercial AT sign by default) so Kermit sends it twice. With this command you can set the intercept character (ie the one to send twice) to another character.

TERMINAL *option*
: Select one of the following terminal characteristics:

  OFF
  : sets emulation off, and its up to the attached terminal to respond to escape sequences

sent from the remote host system.

DUMB Like off, but carriage return and line feed characters are the only control characters accepted. All other control characters are simply ignored. (Really a "Glass TTY").

EXTERNAL
Emulation is provided for by a routine in the system dependent part of Kermit. Attempting to set this option without having and externally supplied routine will returna "Not Implemented" error.

VT52 When connected as a terminal to a foreign host, the micro emulates a VT52. VT52 emulation is set by default, except on micros that already have terminal functionality built in, such as the DEC VT180 and DECmate (these act as VT100-series terminals). Some systems emulate other terminals, like the ADM3A; see table 15-5.

QUIET Do not display any file transfer information onto the console. This mode is useful if you console takes a long time to update the display. Only the file name is displayed. DEBUGging information is not displayed even if selected.

REGULAR
Inverse of QUIET. All packets etc displayed, as ususal.

TIMER ON (or OFF)
Enable or disable the "fuzzy timer". The timer is off by default, because in the normal case KERMIT-80 is communicating with a mainframe KERMIT that has its own timer. Mainframe KERMIT timers tend to be more precise or adaptable to changing conditions. You should SET TIMER ON if you are communicating with a KERMIT that does not have a timer. You should SET TIMER OFF if you are communicating over a network with long delays.

USER *new user number*
Sets another user number to be active. Acceptable user numbers are 0 to 31, though it is recommended to use user numbers 0 to 15 only. This is really only useful for Winchester Systems with high disk capacities.

WARNING ON (or OFF)
Warn user of filename conflicts when receiving files from remote host, and attempt to generate a unique name by adding "&" characters to the given name. ON by default.

SHOW Display all setable parameters. You will get a page or so of the status af all parameters that can be set using the SET command.

STATUS The same function as Show.

STRING *line of text*
Send a character string to the host. This simply copies the string to the correct line, and assumes all appropriate parameters have been set to be used eg baudrate, parity etc. It is intended as an option for the TAKE command.

TAKE *filespec*
Take characters and commands from the specified file as if they were entered from the keyboard. This is useful if you want to set up a batch job. A command file can send, get, receive, set functions etc automatically.

An automatic "TAKE KERMIT.INI" is executed from the default drive when Kermit-80 is loaded. This can be used to set defaults of band rate, parity, filetype, default drive etc.

If KERMIT.INI does not exist, control is given directly to the user.

TRANSMIT *filespec wait character string*
Send the specified file to the system on the other end of the connection as though it were being typed at the terminal, one line at a time. Each line sent is terminated with a carriage return, and any line feeds are stripped from the file sent. After each line has been sent Kermit waits for a character string from the host (eg a acrriage return). If not specified, a carriage return is assumed. No KERMIT protocol is involved. An asterix (star) is sent to the console for every line sent, to indicate how the transfer is progressing. This is useful for sending files to systems that don't have a KERMIT program. During transmission, you may type one of these single-character commands:

Control-C
Cease transmission, and drop into terminal emulation mode.

<CR>
Re-transmit the previous line.

TYPE Type a file to the console. Typing any character other than Control-C while the file is being displayed will suspend the output. Another character will resume output. A Control-C will abort the rest of the output.

VERSION
Show the name, edit number, and edit date of several of the modules that make up Kermit-80.

15.3. Kermit-80 Flavors

Many of the systems supported use an external terminal, rather than a built-in console. Kermit may be further customized for these systems by defining (at assembly time) the terminal type to be used. If the terminal type is unknown or does not match any of the existing terminal options, the generic "CRT" option may be selected. In this case, Kermit cannot do fancy screen control during file transfer; it simply types the file names, packet numbers, and messages in sequence across and down the screen. This works best if you can put your micro or terminal in "autowrap" mode; otherwise the packet numbers will pile up in the rightmost column; the filenames and messages will always appear on a new line, however. If no specific terminal has been selected, Kermit cannot do VT52 emulation; it can act as a "dumb terminal" (sometimes called a "glass TTY"), or else its own built in terminal firmware provides cursor control functions independent of the Kermit program.

15.3.1. Generic Kermit-80

"Generic Kermit-80" is an implementation of Kermit that should run on any 8080-compatible CP/M 2.2 system with no modification at all, or perhaps only a minor one. Unlike other Kermit-80 implementations, it contains no system-dependent manipulation of the serial port. All I/O is done with standard CP/M BIOS calls, and I/O redirection is done using the CP/M IOBYTE function, which, according to the Digital Research *CP/M Operating System Manual*, is an optional feature of any particular CP/M implementation. If your system does not provide the IOBYTE function, Generic Kermit-80 will not work; furthermore, not all systems that implement IOBYTE do so in the same way. The SET PORT command may be used to select the devices to be used for input and output. Table 15-1 lists the options to the SET PORT command and their effects.

SET PORT *xxx*	input from	output to
CRT	CRT:	CRT:
PTR	PTR:	PTP:
TTY	TTY:	TTY:
UC1	UC1:	UC1:
UR1	UR1:	UP1:
UR2	UR2:	UP2:

Table 15-1: Kermit-80 SET PORT Options

The default is SET PORT PTR. In all cases, the console (CON:) and list (LST:) devices used are those selected when Kermit is started.

The reason all Kermit-80 implementations aren't generic is that a good deal of speed is sacrificed by getting all services from the operating system. While a specific implementation of Kermit-80 may be able to operate at 4800, 9600, or even 56 Kilo baud, generic Kermit will fail to work on some systems at speeds in excess of 1200 baud. In addition, many features of Kermit require more specific knowledge of the hardware involved. Generic Kermit cannot send a BREAK signal, or change the baud rate, for example.

15.3.2. CP/M 3 Kermit

CP/M-3 Kermit (also known as CP/M-Plus Kermit) is a version of generic Kermit-80, and should run on most CP/M-3 (CP/M-Plus) systems. It uses the auxilliary port (AUX:) to communicate to the remote Kermit. The SET BAUD and SET PORT commands are not supported; nor can a BREAK be sent. Like generic Kermit-80, a terminal may be selected at assembly time.

15.3.3. System-Specific Versions

There are also many versions of Kermit-80 tailored to specific systems. Most of these operate uniformly, but some of them take advantage (or suffer limitations) of the specific system. Here are some of the special features for particular systems:

Amstrad: -- Two versions:

PCW 8256
: The PCW 8256/8512 with the serial inerafce attached.

CPC 6128
: The 664 with add on memory and 6128 are both supported. Both systems must run CP/M Plus, so the 664 will need an add on RAM pack and CP/M upgrade. A high speed transfer rate of 38k baud can be used between Amstrad computers.

ACCESS:
: Access Matrix computer using port J5. Supports SET BAUD-RATE for rates of 300-9600 baud.

Apple II -- four variations:

APMMDM:
: Apple with Z80 Softcard and Micromodem II in slot 2 Dialout capability provided in connect command; user is prompted for phone number if carrier is not present. During connect mode, `^]D` drops carrier. BYE command also causes carrier to be dropped.

AP6551:
: Apple with Z80 Softcard, and one of several 6551-based communication cards; the slot number is a compile-time parameter (default is slot 2). SET BAUD-RATE supported; speeds are 110-19200 baud.

APCPS:
: Apple with Z80 Softcard and CP Multi-Function Card. The slot number is again a compile-time parameter. SET BAUD-RATE is supported for baud rates from 50 baud to 19200 baud.

AP6850:
: Apple II with Z80 Softcard and a 6850-based USART in slot 2-the slot being a compile-time parameter. SET BAUD-RATE is not supported.

BBC:
: Acorn Computers BBC Computer with Acorn Z80 second processor running CP/M-80. Supports SET BAUD-RATE and can send breaks.

BigBoard II:
: Uses serial port A. To use port B, change mnport, mnprts, and baudrt and reassemble. Can generate BREAK. SET BAUD-RATE supported; speeds are 300-38400 baud.

Cifer:
: Originally coded for Cifer 1886 using the VL: port set as TTYI: and TTYO: but works successfully on 18xx and 28xx series machines.

  There are now two versions, each with two variations: Either running CP/M Version 2.2 or 3.0, and either using the VL: or AUX: ports. The VL: port version can only use seven bits of data, so parity prefixing is required for binary file transfers. This restriction is removed by using the AUX: port. For those interested, the problem is due to the interprocessor link between the video and CPU (!) boards. The VL: port is on the video board, and the AUX: port on the CPU board, and the inter processor link can only transfer seven bits of data.

  Supports SET BAUD-RATE, and can generate breaks on some models with a BREAK key.

Comart:
Comart Communicator-Similar to Northstar equipment. Can generate BREAK.

Compupro:
Based on Kermit 3.x, and has been merged into V4.09

CPT-85xx word processors:
Can generate BREAK. SET BAUD-RATE supported; speeds are 50-9600 baud.

Cromemco:
Cromemco computers with TU-ART card. Supports SET BAUD-RATE (110-9600 baud).

DEC DECmate II word processor (with Z80 card):
Can generate BREAK.

DEC VT180 (Robin):
Three output ports, referred to as COMMUNICATIONS, GENERAL, and PRINTER. Can generate BREAK.

Digicomp Delphi 100:
SET BAUD-RATE supported; speeds are 50-19200 baud.

Discovery:
Action Computer Enterprises "Discovery" Multi-user Computer. Uses Port B on an 83U user board. Supports SET BAUD-RATE for 50-19200 baud. Can generate BREAK.

Epson:
Epson PX-8 with LCD display. Although it is quite different in displaying of Packet Information, it works as any other CP/M-80 Kermit. Supports SET BAUD-RATE and can generate BREAK.

Generic Kermit:
Two versions, one for CP/M version `2.2` and version 3. These systems use IOBYTE flipping (V`2.2`) and the AUX: device to communicate to the serial line. You may have to SET PORT xxx before the version `2.2` will work, as Kermit needs to know what device is the serial line.

Genie:
Eaca Video Genie.

Heath: Three Versions:

H8QUAD
for Heath-8 systems with the quad io board. This system has been derived from V3.x code. Note that this version will not run "as is" on H89 systems.

H89 For Heath-89 machines suports baud rates from 50 to 56,000 baud.

Z100
For Z-100 running CP/M-85. This version does not support setting of baud rates.

Intertec Superbrain: Two Versions:

BRAINA
For superbrain using AUX port. Breaks and SET BAUD both supported

BRAINM
As above, but using the MAIN port.

Ithaca:
Ithaca Intersystems based computer using the VIO card for all IO to the outside world. The system is strictly speaking a home-brew variant of the Ithaca machine, using an S100 cardcage without a front panel. It uses the Extended BIOS by EuroMicro of London. However, I see no reason for this version not running on a genuine Ithaca Intersystems machine. There are patches needed to the EuroMicro BIOS to make this version work.

Kaypro:
Should work on most Kaypro models, as well as some related systems (Ferguson BigBoard I, Xerox 820). For the newer Kaypros with multiple ports, Kermit uses the one labeled "serial data"; it cannot use the serial printer or internal modem ports (but it should be possible to modify the values for mnport, mnprts, and baudrt to do this). Can generate BREAK. SET BAUD-RATE supported; speeds are 50-19200 baud.

Lobo:

Lobo MAX-80. Supports SET BAUD-RATE and can generate BREAKS.

Merlin:
British Telecom Merlin M2215 (also Rair Black Box, possibly also the ICL PC?). Requires a terminal.

Micromate:
PMC 101 Micromate. Supports SET BAUD-RATE and can generate BREAK.

Micromint: Two versions

S6 The Ciarcia/Micromint sb-180 board with a 6Mhz procoessor. System requires a terminal.

S9 As above, but with a 9Mhz processor.

NCR:
Decisionmate 5. Uses the 2651 and is largely the same as the Ithaca Intersystems machine implementation.

Northstar: -- There are four versions available:

NORTHS:
Northstar Horizon with HS10-4 board. Supports SET BAUD-RATE and SET PORT.

HORIZON:
Northstar Horizon using the serial ports on the Mother board. Can generate BREAK.

BASICNS:
Basic Northstar Horizon using the printer port. Can generate BREAK.

ADVANT:
Northstar Advantage. Supports SET BAUD-RATE and can generate BREAK. Traps Control-0 in the system filter.

Morrow Decision I:
Uses the Multi-I/O board. Port 1 is the console, port 3 is the communications line. SET BAUD-RATE supported; speeds are 75-56000 baud.

Morrow Micro Decision I:

Nokia MicroMikko:
Will not echo control-O (which locks keyboard). SET BAUD-RATE supported; speeds are 75-9600 baud.

Ohio Scientific:
Doesn't have screen control.

Osborne 1:
Uses serial line, not internal modem. Left-arrow key generates <DEL> ("delete" or "rubout" character) during connect mode. SET BAUD-RATE supported; speeds are 300 and 1200 baud. Now supports multi-sector buffering.

Research Machines: Two Versions:

RM380ZM:
380Z and 5.25" disks supports SET BAUD.RATE

RM380ZF:
380Z and 8" disks, otherwise as above.

Sanyo:
Sanyo MBC-1100. This version derived from Kermit V3.x

ScreenTyper:
Details unkown.

TRS-80: Three versions:

TRS80LB:
TRS-80 with Lifeboat CP/M

TRS80PT:
TRS-80 with Pickles and Trout CP/M

TRSM4:
TRS-80 Model 4 with Montezuma CP/M

Teletek:
Teletek Systemaster. Supports SET BAUD.

Telcon:
TELCON ZOBRA portable computer.

Torch:
Torch Unicorn 5 initially, but the code runs on other Z80 based CP/N (as in Nut!) systems. It uses the BBC Computer as a "Base processor", and is similar to the BBC version. The base processors RS423 port is used rather than any built in Modem. (UK telecoms legislation effectively makes modem control software tricky business...). Two potential versions exist-one using cursor positioning codes for a MCP and CCCP ROM combination of revision less than 1.00, the other version uses the additional facility MCP/CCCP versions greater than 1. Supports SET BAUD-RATE and can generate BREAKs.

Note that binary files must be transferred using SET PARITY to anything other than NONE! Parity is neither generated nor checked.

US Micro Sales:
S-100-8 based computer.

Vector Graphics:
Vector

Xerox:
Xerox 820.

Z80MU:
Development Kermit on a PC running the Z80MU Z80 and CP/M 2.2 development system. Allows development of the system independent modules to be done on an IBM PC or clone. Also allows the generation of new .HEX files, that may then be KERMITed to the target system. Note: Not all the BDOS or BIOS routines are supported, so avoid "unusual" BIOS/BDOS calls. (For example, DIR from within Kermit will fail as one of the BIOS routines returning disk parameters is not supported.)

15.4. Installation of Kermit-80

Kermit-80 was written originally for the Intertec SuperBrain in lowest-common-denominator 8080 code with the standard assembler, ASM (single source module, no macros, no advanced instructions), so that it could be assembled on any CP/M-80 system (the 8080 assembler is distributed as a standard part of CP/M-80, whereas the fancier Z80 or macro assemblers are normally commercial products). It has since been modified to run on many other systems as well. Kermit-80 should be able to run on any 8080-, 8085- or Z80-based microcomputer under CP/M with appropriate minor changes to reflect the port I/O and screen control for the system (see below).

The proliferation of new systems supported by Kermit-80 made the program grow so large and complicated that it had to be broken up into system-independent and system-dependent modules, as of version 4 (this was done by Charles Carvalho of ACC). Each module is composed of multiple files. This has reduced the time and disk space necessary for assembly; Kermit-80 may once again be assembled on a CP/M system with roughly 250Kbytes of space. The majority of the code does not need to be reassembled to support a new system. Unfortunately, it can no longer be assembled with ASM, since ASM does not support multiple input files. To allow it to be assembled on any CP/M system, the public-domain assembler LASM is included in the distribution kit. Kermit-80 may also be assembled with Microsoft's M80 (not supplied). In theory, any 8080 assembler supporting the INCLUDE directive ought to work, as well.

All versions of Kermit-80 are assembled from the same set of sources, with system dependencies taken care of by assembly-time conditionals within the system-dependent module (eventually, the system-dependent module will itself be broken up into multiple files, one for each system). The most important system dependencies are terminal emulation (when CONNECTed to the remote host) and screen handling, which are dependent on the individual micro's escape codes (these features are table driven and easily modified for other CP/M systems), and the lowest level I/O routines for the serial communications port. The port routines are best done only with BDOS calls, but some systems do not allow this, primarily because the BDOS routines strip the parity bit during port I/O, and the parity bit is used for data when transmitting binary files.

Kermit-80's I/O routines must check the port status and go elsewhere if no input is available; this allows for virtual terminal connection, keyboard interruption of stuck transmissions, etc. On systems that fully implement I/O redirection via the optional CP/M IOBYTE facility, this may be done by switching the IOBYTE definition. On others, however, IN/OUT instructions explicitly referencing the port device registers must be used.

CP/M-80 KERMIT versions 3.8 and later include a "fuzzy timer" that allows a timeout to occur after an interval ranging from 5 to 20 seconds (depending upon the speed of the processor and the operating system routines) during which expected input does not appear at the port. In this case, retransmission occurs automatically. In any case, you may type a carriage return during transmission to simulate a timeout when the transfer appears to be stuck.

15.4.1. Organization of Kermit-80

Kermit-80 consists of two modules, each of which is generated from multiple source files. The first module contains the system-independent code; the second module is configured for a particular system and merged with the system-independent module to produce a customized Kermit-80.

The distribution kit contains:

- the system-independent module, `CPSKER.HEX`;
- the system-dependent modules, `CPV*.HEX` (see table 15-2 and 15-3);
- the source files, `CPS*.ASM` and `CPX*.ASM`,
- the public-domain CP/M assembler, `LASM.*`,
- the public-domain CP/M load/patch utility, `MLOAD.*`

Symbol	Filename	System
ACCESS	CPVACC	Access Matrix
ADVANT	CPVADV	Northstar Advantage
AP6551	CPVAPL	Apple II, Z80 Softcard, 6551 ACIA in serial interface
AP6850	CPVA65	Apple II, Z80 Softcard, 6850 ACIA in Serial Iiterface
APMMDM	CPVAPM	Apple II, Z80 Softcard, Micromodem II in slot 2
APCPS	CPVCPS	Apple II, Z80 Softcard, with CPS multifunction card
BASICNS	CPVBNS	Northstar Horizon (terminal required)
BBC	CPVBBC	Acorn "BBC" computer with Acorn Z80 second processor
BBII	CPVBB2	BigBoard II (terminal required)
BRAINM	CPVBRM	Intertec Superbrain using the main port
BRAINA	CPVBRA	Intertec Superbrain using the Aux port
CIFER2	CPVCIF	Cifer 1886 using the VL: Serial port and CP/M V2.2
CIFER3	CPVCI3	Cifer 1886 using the VL: Serial port and CP/M V3.0
CIFER2	CPVCA2	Cifer 1886 using the AUX: Serial port and CP/M V2.2
CIFER3	CPVCA3	Cifer 1886 using the AUX: Serial port and CP/M V3.0
CMEMCO	CPVCRO	Cromemco with TU-ART card. Terminal required)
COMART	CPVCOM	Comart Communicator (terminal required)
COMPRO	CPVPRO	Compupro with Interfacer 4 (or 3). Terminal required.
CPC	CPVCPC	Amstrad CPC 664 and 6128 and CP/M 3
CPM3	CPVCP3	"Generic": CP/M 3.0 (CP/M Plus) systems (terminal req'd)
CPT85XX	CPVCPT	CPT-85xx wordprocessor with CP/M
DELPHI	CPVDEL	Digicomp Delphi 100 (terminal required)
DISC	CPVDIS	Action Computer Enterprises "Discovery" (terminal req'd)
DMII	CPVDM2	DECmate II with CP/M option
GENER	CPVGEN	"Generic": CPM 2.2 systems with IOBYTE (terminal req'd)
GENIE	CPVGNI	Video Genie
H8QUAD	CPVH8Q	Heath-8 with Quad 8 i/o board
HEATH	CPVH89	Heath/Zenith H89
HORIZON	CPVHOR	Northstar Horizon (terminal required)
KPII	CPVKPR	Kaypro-II (and 4; probably supports all Kaypro systems)
LOBO	CPVLBO	Lobo Max-80

"symbol" is the symbol used to select the target system, in `CPVTYP.ASM`;

"filename" is the name under which the module is supplied in the distribution.

Table 15-2: Systems supported by Kermit-80 (Part 1)

Symbol	Filename	System
M2215	CPVMRL	British Telecom Merlin/Rair Black Box (terminal required)
MDI	CPVMDI	Morrow Decision I (terminal required)
MIKKO	CPVMIK	MikroMikko
MMATE	CPVMM	PMC 101 Micromate (terminal required)
MMDI	CPVUD	Morrow Micro Decision I (terminal required)
NCRDMV	CPVDMV	NCR Decision Mate V. (Terminal required?)
NORTHS	CPVNS	Northstar Horizon with HSIO-4 card (terminal req'd)
OSBRN1	CPVOSB	Osborne 1
OSI	CPVOSI	Ohio Scientific
PCI2651	CPVPCI	Ithaca Intersystems with VI0 card (terminal required)
PCW	CPVPCW	Amstrad PCW 8256/8512 with serial interface
PX8	CPVPX8	Epson PX-8
RM380ZM	CPVRMM	Research Machines 380Z with MDS (5.25" discs)
RM380ZF	CPVRMF	Research Machines 380Z with FDS (8" discs)
ROBIN	CPVROB	DEC VT180
S1008	CPVUSM	US Microsales S-100-8 (terminal required)
SANYO	CPVSAN	Sanyo MBC-1100
SB6	CPVSB6	Micromint SB-180 with 6Mhz CPU (terminal required)
SB9	CPVSB9	Micromint SB-180 with 9Mhz CPU (terminal required)
SCNTPR	CPVSCN	Screentyper
TELCON	CPVTEL	TELCON Zobra portable
TELETEK	CPVTET	Teletek Systemaster
TORCH	CPVTRC	Torch computers BBC-B with Z80 second processors
TRS80LB	CPVTLB	TRS-80 model II with Lifeboat 2.25C CP/M Display
TRS80PT	CPVTPT	TRS-80 model II with Pickles + Trout CP/M Display
TRSM4	CPVTM4	TRS-80 model IV
VECTOR	CPVVEC	Vector Graphics
XER820	CPVXER	Xerox 820
Z100	CPVZ00	Z-100 under CP/M-85
Z80MU	CPVZ80	Z80MU development system on a PC

"symbol" is the symbol used to select the target system, in `CPXTYP.ASM`;

"filename" is the name under which the module is supplied in the distribution.

Table 15-3: Systems supported by Kermit-80 (Part 2)

Symbol	Terminal type
CRT	Dumb terminal type. Does not do cursor addressing
ADM3A	Lear Seigler ADM 3A
ADM22	Lear Seigler ADM 22
AM230	Ampro 230
H1500	Hazeltine 1500
SMRTVD	Netronics Smartvid
SOROQ	Soroq IQ-120
TVI912	Televideo 912
TVI925	Televideo 925 or Freedom 100
VT52	Dec VT52 or equivalent (H19)
VT100	Dec VT100 or equivalent
WYSE	Wyse 100

"symbol" is the symbol used to select the target system, in CPXTYP.ASM;

"Terminal type" is the type of terminal "symbol" selects.

Table 15-4: Terminals supported by Kermit-80

15.4.2. Downloading Kermit-80

You'll need either a pre-configured .COM file or the system-independent module, CPSKER, in binary (.COM) or hex (.HEX) format and the system-dependent overlay for your system (from Tables 15-2 and 15-3). If your system is not listed in the table, get the generic CP/M 2.2 Kermit or the generic CP/M 3 Kermit. If you already have a version of Kermit on your micro and you want to install a new version, simply use your present version to get the new files. Transfer the files to your system and skip ahead to "merging the modules".

If you do not have a copy of Kermit on your micro, and you cannot borrow a Kermit floppy but you do have access to a mainframe computer with a copy of the Kermit-80 distribution, you should read this section.

There are several ways to get CP/M Kermit from a host system to your micro. The easiest is to "download" the necessary "hex" files into your micro's memory and then save them on the disk. If you have a terminal emulator program on your micro which can save a copy of the session to disk, connect to your host, and type the necessary files. Exit from the emulator, saving the session log, and edit the session log to extract the hex files. Skip ahead to "merging the files".

The following is a procedure which, though far from foolproof, should allow you to get a version of Kermit to your CP/M based micro. It depends upon the host prompt, or at least the first character of the host prompt, being some character that cannot appear in a hex file (the valid characters for hex files are the digits 0-9, the upper case letters A-F, the colon ‘‘:’’, carriage return, and line feed). As soon the prompt character is encountered, the transfer will terminate. If your host does not issue a prompt that will accommodate this scheme, you can achieve the same effect by adding an atsign ‘‘@’’ to the very end of the hex file before sending it from the host. The program below looks for an atsign (the normal DEC-20 prompt, hex 40). DECSYSTEM-10 users would look for a dot, hex 2E; VAX/VMS or UNIX users would look for a dollar sign, hex 24; UNIX C-Shell users would look for a percent sign, hex 26.

1. For CP/M 2.2 systems, connect to a floppy disk with plenty of free space. Run DDT and type in the following (the comments should not be typed in; they are there just to tell you what's happening): (Note that this wont work for CP/M Plus or 3.0 systems!)

```
-a100                  ;Begin assembling code at 100
     0100    LXI H,2FE       ;Where to store in memory
```

```
0103    SHLD 200        ;Keep pointer there
0106    MVI E,D         ;Get a CR
0108    MVI C,4         ;Output to PUNCH (send to HOST)
010A    CALL 5
010D    MVI C,3         ;Input from READER (read from HOST)
010F    CALL 5
0112    ANI 7F          ;Strip parity bit
0114    PUSH PSW        ;Save a and flags
0115    MOV E,A         ;Move char to E for echo
0116    MVI C,2         ;Output to screen
0118    CALL 5
011B    POP PSW         ;Restore A and flags
011C    CPI 40          ;(or 4E,24,26,etc) System prompt?
011E    JZ 127          ;Yes, have whole file in memory
0121    CALL 17A        ;No, store another byte
0124    JMP 10D         ;Read another byte
0127    MVI A,1A        ;Get a Control-Z (CP/M EOF mark)
0129    CALL 17A        ;Store it in memory
012C    LXI H,300       ;Get memory pointer
012F    SHLD 202        ;Store as DMA pointer
0132    LDA 201         ;Get 'HI' byte of memory pointer
0135    STA 200         ;and store it as 'LO' one
0138    XRA A
0139    STA 201         ;Zero 'HI' byte (slow *256)
013C    MVI C,16        ;Make NEW file
013E    LXI D,5C        ;With FCB1
0141    CALL 5
0144    CALL 15E        ;Write 128 bytes (sector)
0147    CALL 15E        ;Write another sector
014A    LXI H,FFFF      ;Get a 16-bit Minus One
014D    XCHG            ;into DE
014E    LHLD 200        ;Get 256-byte counter
0151    DAD D           ;Decrement
0152    SHLD 200        ;and store back
0155    MVI A,2         ;Check if
0157    CMP L           ; 256-byte counter down to offset
0158    JZ 183          ;Yes, we're done
015B    JMP 144         ;Keep writing..
015E    LHLD 202        ;Get file-pointer
0161    XCHG            ;into DE
0162    MVI C,1A        ;Set DMA-address
0164    CALL 5
0167    MVI C,15        ;Write sector (128 bytes)
0169    LXI D,5C        ;using FCB1
016C    CALL 5
016F    LHLD 202        ;Get file-pointer
0172    LXI D,80        ;128-bytes
0175    DAD D           ;added to file-pointer
0176    SHLD 202        ;and save
0179    RET             ;and return
017A    LHLD 200        ;Get Memory-pointer
017D    MOV M,A         ;Store character
017E    INX H           ;Increment Pointer
017F    SHLD 200        ;and save
0182    RET             ;and return
0183    MVI C,10        ;CLOSE file
0185    LXI D,5C        ;using FCB1
0188    CALL 5
018B    JMP 0           ;Force WARM BOOT
0179
-^C                     ;(Type Control-C) Return to CP/M
A>SAVE 1 FETCH.COM      ;Save program, we need to run it twice.
```

Figure 15-1: Bootstrap program for Kermit-80 and CP/M Version 2.2

Alternatively, an assembler source file for this program is distributed with CP/M Kermit as `CPKFET.ASM`. You might prefer to type the assembler version in and assemble and load it (ASM CPKFET, LOAD CPKFET, or MASM CPKFET, MLOAD CPKFET), to let the assembler and loader catch any typing errors.

2. Connect to your host using a terminal or a terminal emulation program. Ensure that your host does not have your terminal in "page mode" (does not pause at the end of each screenful).

3. Tell the host to display the first hex file (the system-independent module) at your terminal, e.g. give a command like `TYPE CPSKER.HEX`, *without a terminating carriage return*.

4. Return to your micro by switching the cable from the terminal to the micro, or by terminating the micro's terminal program.

5. Make sure your IOBYTE is set so that `RDR:` and `PUN:` correspond to the I/O port that is connected to the host (this would normally be the case unless you have done something special to change things).

6. Load the program you entered in the first step with DDT, and use it to capture the first hex file:

```
DDT FETCH.COM
-icpsker.hex            ;Setup FCB for file CPSKER.HEX
-g100,179               ;Execute the program.
```

   Now there should be a file `CPSKER.HEX` on your connected disk.

7. Return to the host, and tell it to display the second hex file (the system-dependent module for your configuration). Again, do not type the terminating carriage return.

8. Return to your micro, and run the capture program again:

```
DDT FETCH.COM
-icpxovl.hex            ;Setup FCB to create CPXOVL.HEX
-g100,179               ;Execute the program.
```

   Now there should be a file `CPXOVL.HEX` on your connected disk. Replace `CPXOVL.HEX` in this example with the appropriate overlay file for your system.

Merging the files:

1. For purposes of illustration, we will assume the system-dependent overlay is called "cpxovl.hex". The two hex files may be combined with MLOAD or DDT. If you already have a running Kermit, you can transfer `MLOAD.HEX` to your system and create `MLOAD.COM` by running LOAD. If you're bootstrapping Kermit, you could transfer `MLOAD.HEX` to your system the same way you got the other two `.HEX` files, but it's probably simpler to use DDT to get Kermit running, and get MLOAD later if you need it.

2. Using MLOAD, the two pieces may be easily merged:

```
A>mload kermit49=cpsker,cpxovl
(Some messages about program size, etc...)
A>
```

3. If you don't have MLOAD running, it's a bit more complex:

```
A>ddt cpsker.hex
NEXT  PC
3500 0100
-icpxovl.hex
-r
NEXT  PC
```

```
xxxx 0000
-^C
A>save dd kermit49.com
```

The page count ("dd") used in the SAVE command is calculated from the last address ("xxxx") given by DDT in response to the R command: drop the last two digits and add 1 if they were not zero, then convert from hexadecimal (base 16) to decimal (base 10): 684F becomes 69 hex, which is 105 decimal (5 times 16 plus 9) -- but 6700 becomes 67 hex, or 103 decimal (consult an introductory computing book if you don't understand number base conversion).

4. If you are using the Z80MU CP/M and Z80 development toolkit on an IBM PC or clone, then follow the same instructions as for a genuine CP/M system. When you have loaded your file, you will have to ship the .COM or two .HEX files to the target CP/M system. (Possibly using a previous issue of Kermit?)

5. Note that CP/M hex files have checksums on each line. If there were any transmission errors during the downloading process, MLOAD or DDT will notice a bad checksum and will report an error (something like "Illegal Format"). If you get any errors during loading, either fix the hex file locally with an editor, or repeat the transfer.

You now should have a running version of Kermit-80, called `KERMIT49.COM`.

Test your new Kermit by running it. If it gives you a prompt, it might be OK. (don't delete your old one yet...). Instead of a prompt, you could get one of two messages indicating that the configuration information is invalid:

```
?Kermit has not been configured for a target system
```

or

```
?Consistency check on configuration failed
```

Of course, neither of these messages should appear if you're building Kermit from the distribution kit. The first message indicates that the overlay was not found where the system-independent module expected to find it, probably because the overlay address is incorrect; the second indicates that the version of CPXLNK used in the system-dependent module is incompatible with the system-independent module.

Once you are satisfied that KERMIT40 works correctly, you should rename your old `KERMIT.COM` to something else, like `OKERMIT.COM`, and rename `KERMIT40.COM` to `KERMIT.COM`.

15.4.3. Assembling Kermit-80 from the sources

Kermit-80 is built in two pieces from the following files:

The system-independent files:

`CPSKER.ASM`	header file
`CPSDEF.ASM`	definitions for both KERMIT and KERSYS
`CPSMIT.ASM`	initialization, main loop, miscellaneous commands (BYE, EXIT, LOG, SET, SHOW, STATUS, and VERSION)
`CPSCOM.ASM`	second part of commands, status and set file
`CPSPK1.ASM`	part 1 of the KERMIT protocol handler (SEND, RECEIVE, LOGOUT,and FINISH commands)
`CPSPK2.ASM`	part 2 of the KERMIT protocol handler
`CPSREM.ASM`	REMOTE routines (FINISH, BYE and LOGOUT in CPXPK*.ASM)
`CPSSER.ASM`	SERVER routines (for the future)
`CPSTT.ASM`	the transparent commands (TRANSMIT, CONNECT)
`CPSCPM.ASM`	CP/M commands (DIR, ERA, USER, TYPE, PRINT, COPY)
`CPSWLD.ASM`	the wildcard handler
`CPSCMD.ASM`	the command parser
`CPSUTL.ASM`	utility routines and data
`CPSDAT.ASM`	data space and the overlay definitions
`CPXLNK.ASM`	linkage area description

The system-dependent files:

CPXTYP.ASM	system selection
CPXLNK.ASM	system overlay specification and jump table
CPXCOM.ASM	common routines for all systems
CPXSWT.ASM	system selector or switcher

One of:

CPXSYS.ASM	family file for some system-specific code
CPXTOR.ASM	family file for Torch, Superbrain, PCI2651 etc
CPXNOR.ASM	family file for Northstar and Comart machines
CPXMRL.ASM	family file for British Telecom merlin/Rair Black Box
CPXSB.ASM	family file for Micromint SB-180 systems
CPXCIF.ASM	family file for Cifer systems
CPXHEA.ASM	family file for Heath/Zenith systems
CPXAPP.ASM	family file for Apple II systems
CPXPCW.ASM	family file for Amstrad PCW 8256/8512 machines
CPXBBI.ASM	family file for BigBoard, Kaypro and Xerox 820 systems
CPXSYO.ASM	family file for Sanyo MBS-1100 systems
CPXTM4.ASM	family file for Tandy Model 4 with CP/M systems
CPXGNI.ASM	family file for Video Genie systems
CPXPRO.ASM	family file for Compupro systems
CPXZ80.ASM	family file for the Z80MU development system

and if you use a terminal,

CPXVDU.ASM	display codes for VDUs etc. Not always required

The system-independent module contains all of the system-independent files except for CPXLNK.ASM, which is assembled into the system-dependent module to provide the structures needed to connect the two modules. As distributed, the system-independent module is named CPSKER.HEX. If you have a copy of CPSKER.HEX, you do not need to reassemble the system-independent module to configure Kermit for your system.

The system-dependent module consists of CPXTYP.ASM, CPSDEF.ASM, CPXLNK.ASM, CPXSWT.ASM, CPSCOM.ASM, one of the family files CPXSYS.ASM, CPXTOR.ASM, CPXMRL.ASM, CPXSB.ASM, CPXCIF.ASM, CPXHEA.ASM, CPXBBI.ASM, CPXTM4.ASM, CPXGNI.ASM, CPXNOR.ASM, CPXAPP.ASM, CPXPCW.ASM, or CPXPRO.ASM, and possibly CPXVDU.ASM, if your system uses a terminal for the console. One copy of the system-dependent module is supplied already assembled for each supported system; the filename may be obtained from tables 15-2 and 15-3. If a terminal is required for a system, a CRT (glass TTY device) has been selected.

After assembling the two pieces separately, they are combined with DDT or MLOAD into a system-specific Kermit.

If you want to rebuild the system-independent module, the only change you may need to make is to select the assembler to be used, in CPSKER.ASM. Define one of MAC80, M80, or LASM to TRUE to select it as the assembler; the others should be defined FALSE.

Assuming you have the Microsoft Macro Assembler package (M80/L80), you'll need to do the following:

```
A>m80 cpsker=cpsker.asm
A>l80 /p:100,cpsker,cpsker/n/e
```

This will produce CPSKER.COM.

If you are using LASM instead, do this:

```
A>lasm cpsker
```

LASM will generate CPSKER.HEX and CPSKER.PRN. LASM allows options to be specified in the same way as the standard assembler, ASM, so the command

```
A>lasm cpsker.abz
```

will read the source files from drive A, send the .HEX file to drive B, and suppress the listing file.

If you are using the Z80MU development system on an IBM PC or clone, then assemble your files using either LASM and MLOAD or M80 and L80, as if you were using a genuine CP/M-80 system. Note that you will still have the problem of transferring your assembled files to the target CP/M system.

If you want to generate a system-dependent overlay for a particular system, or want to change the terminal supported, you'll need to check three areas in CPXTYP.ASM:

First, the overlay start ADDRESS. The symbol "ovladr" is EQUated to the address of "LNKFLG" in the system-independent module, as the starting address of the overlay (6000H for version 4.09). You'll need to know this value if you're building the overlay with M80/L80. You won't normally need to change this value.

Second, the assembler being used. Again, define one of MAC80, M80, and LASM to be TRUE to select it, and define the others to be FALSE. The two modules (system-independent and system-dependent) do not need to be built with the same assembler.

Third, the system configuration. Locate your system in tables 15-2 and 15-3, then define the appropriate symbol TRUE, and the rest FALSE. If the system comes with a builtin console terminal, define all the terminal switches FALSE. If the system uses an external terminal as the console, locate the terminal in table 15-5 and define the appropriate symbol TRUE, and the remainder FALSE. If the terminal is not listed in table 15-5, use the CRT switch; in this case, VT52 emulation is not supported.

In addition, there are a few general and system-specific symbols which may be altered to fit your system:

APSLOT	For Apple with 6551 ACIA, defines the slot number of the serial card
CPUSPD	Processor speed in units of 100KHz (currently used only for bbII and kpII for timing loops)
TAC	For users connecting through ARPAnet TACs: set to TRUE if you wish the default TACTRAP status to be ON. (This may be overridden with the SET TACTRAP command). If you're not connecting through a TAC, set tac to FALSE and ignore tacval.
TACVAL	For ARPANET TAC users: defines the default TAC intercept character (may be overridden with the SET TACTRAP command).

If you are just assembling an existing configuration, you'll need to edit CPXTYP.ASM only. If you are adding support for a new system, you should not modify CPSDEF.ASM or CPXLNK.ASM; if you do, you'll have to change the system-independent module also. Eventually, CPXSYS.ASM will be split into separate files, each of which will generate one or more related systems. When this happens, you'll want to pick the one closest to your system to use as a starting point.

After editing CPXTYP.ASM as necessary, assemble and link the overlay as follows:

- With M80 (where "xxxx" is the hex value of ovladr from CPXLNK.ASM):

```
A>m80 cpxtyp=cpxtyp.asm
A>l80 /p:xxxx,cpxtyp,cpxtyp/n/x/e
```

- With LASM:

```
A>lasm cpxtyp
```

With an IBM PC or clone using the Z80MU softwrae, follow the instructions as if you were using a real CP/M system.

The overlay (`CPXTYP.HEX`) may then be merged with the system-independent module as described above (creating a runnable Kermit from the distribution kit).

If you are using the Z80MU development system on a PC, and already have a running Kermit-80 v3.9 or later, you can merge the two `.HEX` files into a `.COM` file with LINK80 (TOPS 10/20), MLOAD (Z80MU), L80 (Z80MU), and transfer the new `.COM` file to your micro with Kermit:

- Z80MU on a PC and MLOAD:

```
@MLOAD KERNEW=CPSKER,CPXTYP
```

- Z80MU on a PC and C80:

```
@L80 /P:xxxx,CPXTYP,CPXTYP/N/X/E
```

producing `KERNEW.COM`.

Symbol	Terminal description
crt	Basic CRT, no cursor positioning
adm3a	ADM3A Display or lookalike
adm22	ADM22 Display or lookalike
am230	Ampro 230
h1500	Hazeltine 1500
smrtvd	Netronics Smartvid-80
soroq	Soroq IQ-120
tvi912	TVI 912
tvi925	TVI 925, Freedom 100
vt52	VT 52 or VT52 emulator such as Heath H19, H29, etc.
vt100	VT 100 or emulator (most ANSI terminals should work)
wyse	Wyse 100

Table 15-5: Terminals known to Kermit-80

15.5. Adding Support For A New System

Kermit-80 is built from a common set of source files; the system-dependent module makes heavy use of conditional assembly (this complication will be removed in future releases). The system dependencies arise from attempts to answer some questions:

1. *What kind of terminal is to be supported?*

   For many micros, the console is an integral part of the system, but others can use an external terminal. In either case, the commands to manipulate the screen (position the cursor, erase the screen, etc) must be defined.

2. *How is the serial line accessed?*

   For systems supporting the IOBYTE function, this is straightforward; the symbol "IOBYT" is defined TRUE. If the serial line is accessed with IN and OUT instructions, it may be possible to use the simple I/O routines provided. In this case, the symbol "INOUT" is defined TRUE, the MNPORT and MNPRTS are defined to be the data and control addresses, respectively, and bit masks for testing for "input data available" and "output buffer empty" must be defined. If the interface is strange, leave IOBYT and INOUT set to FALSE, and provide the I/O routines.

3. *What initialization is necessary?*

You may wish to set the baud rate or configure the serial line at startup. Examples for a number of devices are present.

4. *What special features are to be supported?*

   You may want to provide the capability to select one of several serial lines with the SET PORT command, or to change the speed of the serial line with the SET BAUD-RATE command. To do this, you'll need to build a command table, using the systems already supported as examples. The ability to send a BREAK signal is desirable. Again, examples for several different interfaces (ACIA, SIO, etc) are present.

5. *Do you want to design an external terminal type?*

   There is a jump entry in the overlay file to allow users to add their own termainl emulator. If you write the code for such an emulator, you must load this jump address with the address of your emulator, and SET TERMINAL EXTERNAL from within Kermit. All characters will be passed to this routine during connect mode.

15.6. Notes on New Features in Kermit-80 Version 4

- *Debugging aids:* SET DEBUG ON will add two fields to the SEND/RECEIVE display, labelled "Spack" and "Rpack". These display the last packet sent and received. Of course, this slows down the transfer, especially if the console is an external terminal. SET DEBUG OFF removes these fields. The VERSION command displays the name, edit number, and edit date of several of the modules that make up Kermit.

- *TAC support:* ARPAnet TACs (and many other communication devices such as terminal concentrators, modems, port contention units, network PADs, etc) use a printing character (like "@") as an intercept character, to allow commands to be issued to the TAC, or modem, etc. In order to send this character to the host, it must be typed twice. The command "SET TAC CHARACTER" to Kermit enables the "TACtrap" and asks the user to specify the TAC intercept character. This character will be automatically doubled when it appears in Kermit protocol messages (sent by the SEND or RECEIVE commands) or when it appears in a file being sent with the TRANSMIT command. It is not automatically doubled when typed by the user in CONNECT mode. "SET TAC ON" enables the TACtrap but does not change the TAC intercept character, which is initially "@". "SET TAC OFF" disables the TACtrap.

- *File buffering:* Previous versions of Kermit-80 buffered only one sector (128 bytes) at a time during file transfer operations. This version buffers 16Kbytes at a time, reducing the number of times the floppy drive must be spun up and down, and increasing the effective throughput of the link. If the disk transfer rate is too slow, however, the remote Kermit may time out and retransmit packets. This will show up on the screen in the "Retries:" field; if this occurs after disk activity, you may want to increase the timeout value on the remote Kermit, SET BUFFER <new value> while in Kermit,or reassemble Kermit with a smaller value for MAXSEC (in `CPSDEF.ASM`) This buffer is also used by the TRANSMIT command; the log file enabled by the LOG command is still written a sector at a time.

This section is intended for people wanting to implement their own versions of Kermit-80 for computers not already defined.

The system independent code communicates to routines for a specific system through a set of tables. These tables are defined in CPXLNK.ASM, and should not be modified between revisions of Kermit. If an entry is added or deleted, then the whole of Kermit-80 needs reassembling. Make sure that the changes to CPXLNK.ASM are duplicated in CPSUTL.ASM, which has the system independent equivalent of CPXLNK.ASM.

The following entries/definitions apply to revision 4.09. There have been three additional entries since revision 4.05.

The table is split into three sectors; The first section defines two byte "words" giving 16 bits of interface data; The second set is a set of jumps to various functions, and finally the third set a set of pure data bytes.

15.6.1. Interface Data.

LNKFLG Must be first entry in overlay at overlay address. Is a two byte address giving the size of the linkage table. This is used to check for consistency of overlay's

ENTSIZE
Length of entry table, also used for consistency checking after the overlay. Currently 6

SYSEDT The address of a dollar-terminated string giving the overlay revision level and date. Points to a string like: CPXSYS.ASM(33) 4-JUN-1986$

FAMILY The address of a dollar-terminated string giving the Family overlay revision level and date. If the system is in CPXSYS.ASM rather than a particular Family overlay, it is simply a pointer to $

15.6.2. Jump Table.

This is split into three main sectors-

1. Input/Output routines

2. Screen formatting routines

3. other system dependent routines

SELMDM
Parameters None
Returns None
Description selects the modem port. Most systems do nothing and simply return. HL,DE and BC registers preserved.

OUTMDM
Parameters None
Returns None
Description Output the character in E register to the communications line. BC,DE,HL registers preserved.

INPMDM
Parameters None
Returns Accumulator either 0 or character from comms line if available
Description Check modem for character and if so, return it in A. HL,DE,BC registers preserved, flags and accumulator lost.

FLSMDM
Parameters None
Returns None
Description Clear any pending characters in the input buffer from the modem. No registers are preserved.

SELCON
Parameters None

Returns None
Description Select the console. This is a null subroutine for most systems, but for IOBYTE systems selects the console.

<u>OUTCON</u>
Parameters Character in E
Returns None
Description Send the character in E to the console. Any quirks of system responding in an odd manner should be handled. No registers preserved.

<u>INPCON</u>
Parameters None
Returns Zero or character in A.
Description Get a character from the console or return a null if no character to be read. No registers are preserved.

<u>OUTLPT</u>
Parameters Character in E
Returns None
Description Send the character in E to the printer. The console is selected. Only DE registers are preserved

<u>LPTSTAT</u>
Parameters None
Returns 00H or 0FFH in A register
Description Test the printer to see if it is ready to receive a character to be printed. If a 00H is returned then the printer is ready to receive a character.

<u>EXTTER</u>
Parameters Character to be sent to the user supplied terminal emulator in the E register
Returns None
Description If the user has supplied a terminal emulator in the overlay code, EXTTER will be a JMP <non zero address>. If SET TERMINAL EXTERNAL has been set, all caharcters will be passed verbatim to this terminal emulator. If there is no external emulator, this code will never be called. The user should reset terminal conditions on initialisation of both the system and before CONNECT. All registers should be preserved.

<u>XBDOS</u>
Parameters Any required for calling BDOS
Returns Any expected from the called BDOS routine
Description This is an alternative entry to BDOS. This entry will also check the printer status etc. For full details see the code for the BDOS trap in CPSUTL.ASM.

2b)

<u>CLRLIN</u>
Parameters None
Returns None
Description Clear the current line on the terminal

<u>CLRSPC</u>
Parameters None
Returns None
Description Erase the current position (after a backspace)

<u>DELCHR</u>
Parameters None
Returns None

Description Make delete (7FH) look like a backspace. Some systems do a backspace, space, backspace automatically others have to simulate it

CLRTOP
Parameters None
Returns None
Description Clear the screen and place the cursor at the top LH corner

SCREND
Parameters None
Returns None
Description Place the cursor on the line for the Kermit-80 prompt after a file transfer. (Usually line 13)

SCRERR
Parameters None
Returns None
Description Move cursor to the error message field on the file transfer format screen

SCRFLN
Parameters None
Returns None
Description Move the cursor to the filename field

SCRNP
Parameters None
Returns None
Description Move the cursor to the packet count field

SCRNRT
Parameters None
Returns None
Description Move cursor to the retry count field

SCRST
Parameters None
Returns None
Description Move cursor to the status field

RPPOS
Parameters None
Returns None
Description Move to the receive packet field (debugging use)

SPPOS
Parameters None
Returns None
Description Move to the send packet field (for debugging use)

2c)

SYSINIT
Parameters None
Returns None
Description Initialize the system specific items. No registers are preserved. Any initialization is done once only when Kermit-80 is first loaded.

SYSEXIT

Parameters	None
Returns	None
Description	Program termination. De-initialize anything in preparation for a return to CP/M

SYSCON

Parameters	None
Returns	None
Description	Initialize anything before entering the connect state.

SYSCLS

Parameters	None
Returns	None
Description	System dependent close routine when exiting connect state

SYSINH

Parameters	None
Returns	None
Description	Help routine to test for any extensions to the escape menu during the connect state. If a system has any special feature it can use during connect mode, then it can be tested as <escape-character>xxx. This entry is a string for printing to the console for an <escape-character>? Often used for generating breaks or controlling a modem.

SYSINT

Parameters	None
Returns	None
Description	This is a test-and-jump on receipt of an escape sequence not understood by Kermit-80. If the character in A is not recognized by your version of Kermit=80, do a rskip

SYSFLT

Parameters	Character in E
Returns	Character in E. Either a 00H or anything else in A
Description	Test the character in E. If it may not be printed to the console, set A to zero. All other registers preserved. NB <XON>,<XOFF>,<DEL>,<NULL> are always rejected.

SYSBYE

Parameters	None
Returns	None
Description	System dependent processing for the BYE command. (eg hang up the phone)

SYSSPD

Parameters	Value from table in DE
Returns	None
Description	The system dependent code for baud rate change. DE contains the two byte value from the baud rate table. This value is also stored in "SPEED"

SYSPRT

Parameters	Value in DE
Returns	None
Description	The system dependent code for setting the port. The parameters are passed in DE, which are obtained from the port tables

SYSSCR

Parameters	String pointer in DE
Returns	None
Description	Setup the screen display for file transfer. The Kermit version string is pointed to by DE. If the

terminal is not capable of cursor addressing (eg dumb glass TTY) then only the screen is cleared and the version string is printed.

CSRPOS
Parameters Row number in B, column number in C
Returns None
Description Move the cursor to row B, column C where B=1,C=1 is top LH corner of screen. The routine should first end a "cursor position" leading string (up to four characters) then use the parameters given to complete the versions cursor position function

SYSSPC
Parameters None
Returns K bytes free in HL
Description Get the amount of free disk space on the selected disk drive. This could be in the system independent code. Automatically detects CP/M V2.2 or V3.0. No registers saved.

MOVER
Parameters Source Pointer in HL
Destination Pointer in DE
Byte count in BC
Returns None
Description Move (BC) bytes from (HL) to (DE) Z80 based systems do an LDIR, while 8080 systems do it as a loop. All registers destroyed

PRTSTR
Parameters $ terminated string pointed to by DE
Returns None
Description Print the string onto the console.

3)

PTTAB WORD Points to VT52 equivalent escape sequences.

SPDTAB WORD Address of baud-rate command table, or 0 if table does not exist

SPDHLP WORD Address of baud-rate help table, or 0 if SET BAUD-RATE is not supported.

PRTTAB WORD Address of port command table or 0 if SET PORT is not supported.

PRTHLP WORD Address of port help table or 0 if SET PORT is not supported

TIMOUT BYTE FUZZY-TIMER. Set to value suitable to your system (depends largely on CPU speed)

VTFLG BYTE VT52 emulation flag. Set to 0 if terminal emulates a VT52, 01 if emulation is required, or 0FFH if emulations not possible (eg for "CRT")

ESCCHR BYTE default escape character-usually control-] but sometimes control-\

SPEED WORD Storage space for baud-rate. Set to 0FFFFH as baud rates are initially unknown. Note that the STATUS routine only looks at the first (least significant) byte.

PORT WORD Storage space for port. Set to 0FFFFH as ports may not be implemented, and is initially unknown

PRNFLG	BYTE	Printer copy flag-if O no copy. Anything else => copy to printer
DBGFLG	BYTE	Debugging flag. If O then no debugging to be done. (ie writing of debugging info during a file transfer)
ECOFLG	BYTE	Local ECHO flag (default is off)
FLWFLG	BYTE	File warning flag. If set to 1 will not overwrite files already existing on disk with some-named files being transferred
IBMFLG	BYTE	IBM system is the host-assume IBM file transfers etc
CPMFLG	BYTE	Flag indicating type of CP/M files to be transferred. Default setting - DEFAULT
PARITY	BYTE	Type of parity in use 0 = Even parity 3 = Mark parity 6 = No parity (8th bit is data) 9 = Odd parity 12 = Space parity
SPSIZ	BYTE	Size of send packet
RPSIZ	BYTE	Size of receive packet
STIME	BYTE	Send timer (time-out)
RTIME	BYTE	Receive timer (time-out)
SPAD	BYTE	Send Padding (default=0)
RPAD	BYTE	Receive Padding (default=0)
SPADCH	BYTE	Send Padding character (default=NULL)
RPADCH	BYTE	Receive Padding character (default=NULC)
SEOL	BYTE	Send EOL character (default=CR)
REOL	BYTE	Receive EOL character (default=CR)
SQUOTE	BYTE	Send quote character (default=#)
RQUOTE	BYTE	Receive quote character (default=#)
CHKTYP	BYTE	Ascii value of checktype 31H="1"=checktype1 (6bits) 32H="2"=checktype2 (12bits) 33H="3"=CCITT checksum (CRC) Default is 31H("1")
TACFLG	BYTE	If set to on (non zero) send the TACCHR twice. This is for ARPA TAC users,

where the TAC swallows one "wakeup" character. If sent twice the TAC will pas one on and go back to normal mode.

<u>TACCHR</u> BYTE Desired TAC character. It is ignored if TAC trapping is turned off. Value to send twice if TAC interception is set on. Default=0, but set to commercial AT if the conditional assembly flag TAC is set true

<u>BUFADR</u> WORD Address of Multi-Sector buffering for I/O

<u>BUFSEC</u> BYTE The number of bytes the big buffers can hold. Default is 1. (0=256 sectors).

<u>FFUSSY</u> BYTE Indicates if funny characters may be used in CP/M file names (eg `<>.,;?#[]`) If zero, allow anything. Default is nonzero.

<u>BMAX</u> SPACE:(2bytes) Highest block number on selected disk drive

<u>BMASK</u> SPACE:(1byte) (Records/block)-1

<u>BSHIFTF</u> SPACE:(1byte) Number of shifts to multiply by rec.block

<u>NNAMS</u> SPACE:(1byte) Counter for file-names per line

15.7. Future Work

Work that needs to be done in future releases includes:

- Merge in support for additional CP/M-80 systems, particularly those for which support was recently added to the monolithic v3.x source.

- Break up `CPXSYS` into discrete source files, one for each system. These source files should serve as simple models for adding support for new systems to Kermit-80 -- only the very basic screen definitions, flags, i/o primitives, initializations, and so forth should appear in each system-dependent file.

- Addition of missing features -- compression of repeated characters during packet transmission, transmission of file attributes (particularly size, so that "percent done" can be displayed for both incoming and outbound files), advanced commands for servers (REMOTE DIRECTORY, etc), command macros and initialization files, login scripts, remote operation and server mode, etc etc. Any offers??

16. CP/M-86 KERMIT

Authors: Bill Catchings, Columbia University; Ron Blanford, University of Washington; Richard Garland, Columbia University.
Language: Digital Research ASM86
Version: 2.9
Date: December 1984
Documentation:
Frank da Cruz, Columbia

This version of KERMIT is designed to support any CP/M-86 system. So far it supports the DEC Rainbow-100 and the NEC Advanced Personal Computer (APC). It is very similar to CP/M-80 and MS DOS KERMIT.

CP/M-86 KERMIT-86 Capabilities At A Glance:

Local operation:	Yes
Remote operation:	No
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
`^X/^Y` interruption:	Yes
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	No
Alternate block checks:	No
Terminal emulation:	Yes, uses PC firmware (VT100)
Communication settings:	Yes; duplex, parity
Transmit BREAK:	Yes
IBM communication:	Yes
Transaction logging:	No
Session logging (raw download):	Yes
Raw upload:	No
Act as server:	No
Talk to server:	Yes; SEND, GET, FIN, BYE
Advanced commands for servers:	No
Local file management:	Yes
Handle file attributes:	No
Command/init files:	Yes
Printer control:	No

CP/M-86 KERMIT Description

Since Kermit-86 runs on a standalone micro, it is always in control of the screen -- it is always *local*. Thus, it always keeps the screen updated with the file name and the packet number, whether sending or receiving. Kermit-86 is capable of timing out an input request, and can thus break deadlocks automatically. In most cases, however, this is not desirable because the KERMIT on the other side is most likely better able to handle the timeouts; therefore, Kermit-86's timer is normally not used.

If despite the timeout capability, the transmission appears to be stuck (and you can tell that this has happened if the screen fails to change for a long while) you can type carriage return to have the micro do what it would have done on a timeout, namely NAK the expected packet to cause to foreign host to send it again (or, if the micro is sending, to retransmit the last packet). Micro/micro or micro/IBM-mainframe transfers could require this kind of manual intervention.

File transfers may be interrupted in several ways.

Control-C This will return you to Kermit-86 command level immediately, so that you can connect back to the remote system, or take any other desired action.

Control-X When sending a file, this will terminate the sending of the current file with a signal to the KERMIT on the other side to discard what it got so far. If there are more files to be sent, KERMIT-86 will go on to the next one. When receiving a file, KERMIT-86 will send a signal to the remote KERMIT to stop sending this file. If the remote KERMIT understands this signal (not all implementations of KERMIT do), it will comply, otherwise the file will keep coming. In either case, the remote KERMIT will go on to the next file in the group, if any.

Control-Z Like Control-X, except if a file group is being transmitted, this will stop the transmission of the entire group. If only a single file is being transmitted, it works exactly like Control-X.

Carriage Returns If you type carriage return repeatedly Kermit-86 will retry the current packet up to its retry limit (somewhere between 5 and 16 times) and then, if no valid response was received, return to Kermit-86 command level.

When KERMIT-86 is started, it looks for the file `KERMIT.INI`. If found, it executes KERMIT-86 commands from it before prompting you for commands. The KERMIT-86 prompt looks like this:

```
Kermit-86 B3>
```

in which "B" is your current default disk and "3" is the current default user number.

16.1. Kermit-86 Commands

KERMIT-86 uses the DECSYSTEM-20 keyword style command language. Each keyword may be abbreviated to its minumum unique length. "?" may be typed to request a menu of the available options for the current field at any point in a command. ESC may be typed at any point in a command to fill out the current keyword or filename; if sufficient characters have not been typed to identify the current field uniquely, KERMIT-86 will sound a beep and allow you to continue from that point.

CONNECT

Establish a "virtual terminal" connection to any host that may be connected to the serial port, i.e. pass all typein to the serial port and display all input from the serial port on the screen, using the system's own built-in support for ANSI (VT100-like) screen control. When you issue the CONNECT command, the PC will print a message telling you how to get back by typing an an escape sequence, an uncommonly-used control character, normally CTRL-backslash, followed by a single letter "command".

- C Close Connection, return to `Kermit-86>` command level.
- `?` List available single-character commands.
- B Send a BREAK signal.
- Q Quit logging the remote session.
- R Resume logging the remote session.
- L Toggle logging.
- `^\` (or whatever - a second copy of the escape character) Send the escape character itself to the remote host.

SEND *filespec*

Send file(s) specified by *filespec* to the remote Kermit, using the prevailing file mode (ASCII or BINARY; see SET). The *filespec* may contain CP/M wildcards.

RECEIVE

Receive file(s) from the remote Kermit. Store them under the names provided in the file headers supplied by the remote host. If the names aren't legal, use as many legal characters from the name as possible (see the description of SET FILE-WARNING below). If there's a conflict, and FILE-WARNING is ON, warn the user and try to build a unique name for the file by adding "&" characters to the name. You may also provide an optional file name in the RECEIVE command; if you do, the incoming file will be stored under the name you specify. If more than one file arrives, only the first will be stored under the given name, unless you included wildcard characters in the RECEIVE filespec; in that case, the filespec will be used as a mask for incoming filenames. For instance, you told the remote Kermit to send `*.ASM`, you could tell

KERMIT-86 to "receive `*.A86`", thereby changing the filetype of all the incoming files.

GET *filespec*
: When Kermit-86 is talking to a Kermit Server on the host, you should use the GET command to request the server to send files to you, for example: `get hlp:k*.hlp`

BYE
: When talking to a remote Kermit Server, this command shuts down the server and logs it out, and also exits from Kermit-86 to CP/M command level.

LOGOUT
: Like BYE, but leaves you at Kermit-86 command level.

FINISH
: Like LOGOUT, but shuts down the remote server without logging it out. Leaves you at Kermit-86 command level; a subsequent CONNECT command should put you back at host system command level.

EXIT
: Exit from KERMIT-86 back to CP/M.

QUIT
: Synonym for EXIT.

SET *parameter* [*value*]
: Set the specified parameter to the specified value. Possible settings:

BAUD
: Change the baud rate of the communications port. This command only works on some systems, and its actual operation can vary from system to system. Type SET BAUD followed by a question mark, and follow the directions. On systems that do not support this command, you must set the port baud rate from CP/M or other setup mechanism outside of KERMIT-86.

DEBUG
: ON or OFF. If ON, displays incoming and outbound packets during file transfer. OFF by default.

DEFAULT-DISK *disk/user*
: Specify default disk and user number for subsequent file reception and transmission. The specification following the command must be in one of the following forms:

`d:` = go to drive d (A through P) without changing user
`u:` = go to user u (0 through 15) without changing drive
`du:` = go to drive d and user u
`:` = go to the defaults when Kermit was loaded

Whenever a drive is specified, even if it is the same as the current default drive, the drive is logged in so that disks can be swapped without exiting Kermit to type control-C. Kermit restores the original drive and user upon termination.

ESCAPE
: Change the escape character for virtual terminal connections. Select a character in the control range that you will not be likely to need at the remote host; type the new character literally. Certain characters, like Control-X, cannot be specified.

FILE-TYPE
: Tells KERMIT-86 what kind of file it is sending, so that KERMIT can correctly determine the end of the file. SET FILE BINARY means to send all the 128-byte blocks of the file, including the last block in its entirety; SET FILE ASCII is used for text files, and transmission stops when the first Control-Z is encountered anywhere in the file (this is the CP/M convention for marking the end of a text file). If binary transmission is used on a text file, some extraneous characters (up to 127 of them) may appear at the end of the file on the target system. If ASCII transmission is used on a binary file, the entire file will not be sent if it happens to contain any data bytes that correspond to Control-Z. ASCII is the default.

FLOW-CONTROL
: Select the desired type of flow control to be used on the communication line. The choices are NONE and XON/XOFF. XON/XOFF is the default. If the remote system is not full duplex or cannot do XON/XOFF, you should use NONE.

IBM ON (or OFF)
: Allow the transfer of files to and from an IBM mainframe computer. This makes Kermit-86 wait for the IBM turnaround character (XON), ignore parity on input, add appropriate parity to output, and use local echoing during CONNECT. As distributed, KERMIT-86 uses MARK parity for IBM communication. If you don't give this command, IBM mode is OFF. Since IBM

VM/CMS KERMIT does not have timeout capability, SET IBM ON also turns on the timeout facility automatically, as if you had typed "SET TIMER ON".

LOCAL-ECHO ON (or OFF)
: When you CONNECT to a remote host, you must set LOCAL-ECHO ON if the host is half duplex, OFF if full duplex. OFF by default.

LOG
: Specify a log file on the current CP/M disk into which to record incoming characters during CONNECT. If the remote host can do XON/XOFF, then the log file will normally capture every character shown on the screen. When connected to the remote system, several single-character arguments to the connect escape character can be used to control logging -- Q (quit), R (resume), L (toggle). If you use R or L during connect without having previously specified a log file name, then `KERMIT.LOG` is used. An open log is closed when you escape back to the PC.

PARITY
: Sets parity for outgoing characters to one of the following: NONE, SPACE, MARK, EVEN, or ODD. On input, if parity is NONE, then the 8th bit is kept (as data), otherwise it is stripped and ignored. The parity setting applies to both terminal connection and file transfer. If you set parity to anything other than NONE, Kermit-86 will attempt to use "8th bit prefixing" to transfer binary files. If the other KERMIT is also capable of 8th bit prefixing, then binary files can be transferred successfully; if not, the 8th bit of each data byte will be lost (you will see a warning on your screen if this happens).

PORT
: Allows you to switch between different communication ports on the PC. This command is not available on all systems.

TIMER ON (or OFF)
: Enable or disable the timeout facility. The timer is off by default, because in the normal case KERMIT-86 is communicating with a mainframe KERMIT that has its own timer. Mainframe KERMIT timers tend to be more precise or adaptable to changing conditions. You should SET TIMER ON if you are communicating with another KERMIT that does not have a timer. You should SET TIMER OFF if you are communicating over a network with long delays.

WARNING ON (or OFF)
: Warn user of filename conflicts when receiving files from remote host, and attempt to generate a unique name by adding "&" characters to the given name. OFF by default.

SHOW
: Show the current settings of the SET parameters.

TAKE
: Take KERMIT-86 commands from the specified file. The file should not contain any TAKE commands; nested command files do not work.

LOCAL
: This is a prefix for local file management commands, to distinguish them from remote file management commands (which aren't implemented yet). The LOCAL prefix is optional; if left off, the commands will be performed locally.

SPACE
: Show how much space is used and remaining on the current disk.

DIRECTORY
: Provide a directory listing for the current disk, showing the name and size of each file. A filespec may be given to select only a certain file or wildcard file group.

DELETE
: Delete the specified files from the current disk.

TYPE
: A wildcard filespec is accepted and files displayed alphabetically. The display is paged in Unix fashion with "`--more--`" displayed on the last line. Typein options at that point can be obtained by hitting a '`?`'.

16.2. Installation:

CP/M-86 KERMIT is broken up into several source modules:

`C86CMD.A86`	Command parser
`C86FIL.A86`	File handler
`C86Xxx.A86`	System Dependent I/O
`C86KER.A86`	Main Program
`C86PRO.A86`	Protocol Module

```
C86TRM.A86          Terminal Emulation
C86UTL.A86          Utilities
```

The main program module, `C86KER.A86`, contains INCLUDE directives for the other files. The C86Xxx module is stored with "xx" replaced by codes denoting the machine for which the program is being built -- RB for Rainbow, AP for NEC APC, etc. The program may be built on the CP/M-86 system by obtaining all the source files listed above, storing them on the current disk with the names indicated, renaming the appropriate `C86Xxx.A86` file to be `C86XXX.A86`, and then doing:

```
ASM86 C86KER $PZ   (takes about 6 minutes on the Rainbow)
GENCMD C86KER      (takes less than a minute)
```

and, if desired,

```
REN KERMIT.CMD=C86KER.CMD
```

16.3. DEC Rainbow 100 Support

Kermit-86 runs on the DEC Rainbow 100 or 100+ under CP/M-86/80, version 1 or 2, on the 8088 side. It uses the built-in firmware to emulate a VT102 ANSI terminal during CONNECT, and runs well at speeds up to 9600 baud.

You should be able to download the program using the old KERMIT on the Z80 side (Rainbow Kermit, VT180 Kermit, or generic CP/M-80 Kermit will do the job, but only under DEC CP/M-86/80 version 1.0), or an earlier version of Kermit-86.

If you don't have an earlier version of KERMIT, then follow the directions for installing KERMIT-80 (yes, KERMIT-80) in the KERMIT-80 section of the *Kermit User Guide*, but send the Kermit-86 hex file instead. This works because the Rainbow can run CP/M-80 programs like DDT.

Another way to get Kermit onto your Rainbow for the first time would be from a DEC VT-180 diskette. A VT-180 can use its own Kermit to load Rainbow Kermit onto its disk, which can then be read directly by a Rainbow. Also, note that VT-180 Kermit-80 can actually run on the Rainbow on the Z80 side under DEC CP/M-86/80 version 1 (but not version 2 or higher), at speeds of 1800 baud or lower.

16.4. NEC Advanced Personal Computer Support

(Contributed by Ron Blanford, University of Washington)

Currently only the standard serial port is supported, and not the H14 auxiliary port. The SET PORT command is not implemented.

While in Kermit's terminal emulation mode, local commands are initiated by a two-character sequence consisting of the "escape character" followed by one other character identifying the command. (Make the second character a '?' to see a list of the valid commands.) As distributed, the standard Kermit-86 uses the control-backslash character as the escape character in terminal mode. The trouble is that the CP/M-86 BIOS in the APC ignores a keyboard entry of Control-\ (i.e. holding down the CTRL key while striking the '\' key), making it difficult (impossible) to use this method to get out of terminal mode.

One solution is to perform a "SET ESCAPE ^" command before entering terminal mode to change the escape character to a caret (or any other character the APC keyboard will generate). This command could be placed in your `KERMIT.INI` file for automatic execution every time Kermit is started.

The simpler solution is to realize that the character code for a Control-\ is a hexadecimal 1C, and that this is the code generated by the INS key on the numeric keypad. Once you can remember that every reference to Control-\ should be interpreted as a reference to the INS key, this is actually easier to use than the two-key Control-\ sequence.

In the standard CP/M-86 BIOS, the unshifted DEL key generates a Control-X character (hexadecimal 18). This is the CP/M command to erase the current input line, and is very useful for local processing. Most mainframes do not use the Control-X character at all, so it becomes much less useful during terminal emulation. The DEL character (hexadecimal 7F), on the other hand, is often used by mainframes and can only be generated on the APC by holding down the SHIFT key while striking the DEL key (this capability is not mentioned anywhere in the documentation).

Because the Control-X character is so seldom used while the DEL character is commonly used, the initialization procedure in Kermit-86 modifies the CP/M-86 BIOS so that the DEL key generates the DEL character whether shifted or not. Control-X can still be generated if necessary by holding down the CTRL key while striking the 'X' key. The CP/M-86 BIOS is returned to its original state when Kermit terminates.

The APC uses escape sequences which have been standardized by the American National Standards Institute (ANSI) to control cursor movement, screen erasing, and character attribute manipulation. Perhaps the best-known other terminal which follows ANSI guidelines is the DEC VT100. The APC only recognizes a few of the more important ANSI commands, and not the complete set which the VT100 supports.

The ANSI/VT100 features that the NEC APC supports are:

- direct cursor addressing (by row and column)
- relative cursor addressing (up, down, left, right)
- line erasing (cursor to end, beginning to cursor, entire line)
- screen erasing (cursor to end, beginning to cursor, entire screen)
- character attributes (underline, reverse video, blink, but not bold)

In addition, the first four grey function keys (unshifted) generate the escape sequences associated with PF1 through PF4 on the VT100 keyboard. The arrow keys and numeric keypad DO NOT generate the corresponding VT100 sequences.

These functions are enough to support simple command line editing on most systems, and allow mailers or paged file display programs to clear the screen before each display. Underlining and reverse video are also useful in some applications. This is not enough to support the more sophisticated screen control required by screen editors such as EMACS or KED. In addition, due to a bug in the implementation of the CP/M-86 BIOS, the sequence ordinarily used to home the cursor (esc [H) does not work correctly; a patch for CP/M to correct this problem is distributed with APC Kermit-86.

Appendix I
The ASCII Character Set

ASCII Code (ANSI X3.4-1968)

There are 128 characters in the ASCII (American national Standard Code for Information Interchange) "alphabet". The characters are listed in order of ASCII value; the columns are labeled as follows:

Bit	Even parity bit for ASCII character.
ASCII Dec	Decimal (base 10) representation.
ASCII Oct	Octal (base 8) representation.
ASCII Hex	Hexadecimal (base 16) representation.
EBCDIC Hex	EBCDIC hexadecimal equivalent for Kermit translate tables.
Char	Name or graphical representation of character.
Remark	Description of character.

The first group consists of nonprintable ’control’ characters:

```
     .....ASCII.... EBCDIC
Bit  Dec   Oct  Hex  Hex    Char  Remarks
 0   000   000   00   00    NUL   ^@, Null, Idle
 1   001   001   01   01    SOH   ^A, Start of heading
 1   002   002   02   02    STX   ^B, Start of text
 0   003   003   03   03    ETX   ^C, End of text
 1   004   004   04   37    EOT   ^D, End of transmission
 0   005   005   05   2D    ENQ   ^E, Enquiry
 0   006   006   06   2E    ACK   ^F, Acknowledge
 1   007   007   07   2F    BEL   ^G, Bell, beep, or fleep
 1   008   010   08   16    BS    ^H, Backspace
 0   009   011   09   05    HT    ^I, Horizontal tab
 0   010   012   0A   25    LF    ^J, Line feed
 1   011   013   0B   0B    VT    ^K, Vertical tab
 0   012   014   0C   0C    FF    ^L, Form feed (top of page)
 1   013   015   0D   0D    CR    ^M, Carriage return
 1   014   016   0E   0E    SO    ^N, Shift out
 0   015   017   0F   0F    SI    ^O, Shift in
 1   016   020   10   10    DLE   ^P, Data link escape
 0   017   021   11   11    DC1   ^Q, Device control 1, XON
 0   018   022   12   12    DC2   ^R, Device control 2
 1   019   023   13   13    DC3   ^S, Device control 3, XOFF
 0   020   024   14   3C    DC4   ^T, Device control 4
 1   021   025   15   3D    NAK   ^U, Negative acknowledge
 1   022   026   16   32    SYN   ^V, Synchronous idle
 0   023   027   17   26    ETB   ^W, End of transmission block
 0   024   030   18   18    CAN   ^X, Cancel
 1   025   031   19   19    EM    ^Y, End of medium
 1   026   032   1A   3F    SUB   ^Z, Substitute
 0   027   033   1B   27    ESC   ^[, Escape, prefix, altmode
 1   028   034   1C   1C    FS    ^\, File separator
 0   029   035   1D   1D    GS    ^], Group separator
 0   030   036   1E   1E    RS    ^^, Record separator
 1   031   037   1F   1F    US    ^_, Unit separator
```

The last four are usually associated with the control version of backslash, right square bracket, uparrow (or circumflex), and underscore, respectively, but some terminals do not transmit these control characters.

The following characters are printable:

First, some punctuation characters.

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
1	032	040	20	40	SP	Space, blank
0	033	041	21	5A	!	Exclamation mark
0	034	042	22	7F	"	Doublequote
1	035	043	23	7B	#	Number sign, pound sign
0	036	044	24	5B	$	Dollar sign
1	037	045	25	6C	%	Percent sign
1	038	046	26	50	&	Ampersand
0	039	047	27	7D	'	Apostrophe, accent acute
0	040	050	28	4D	(	Left parenthesis
1	041	051	29	5D	)	Right parenthesis
1	042	052	2A	5C	*	Asterisk, star
0	043	053	2B	4E	+	Plus sign
1	044	054	2C	6B	,	Comma
0	045	055	2D	60	-	Dash, hyphen, minus sign
0	046	056	2E	4B	.	Period, dot
1	047	057	2F	61	/	Slash

Numeric characters:

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
0	048	060	30	F0	0	Zero
1	049	061	31	F1	1	One
1	050	062	32	F2	2	Two
0	051	063	33	F3	3	Three
1	052	064	34	F4	4	Four
0	053	065	35	F5	5	Five
0	054	066	36	F6	6	Six
1	055	067	37	F7	7	Seven
1	056	070	38	F8	8	Eight
0	057	071	39	F9	9	Nine

More punctuation characters:

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
0	058	072	3A	7A	:	Colon
1	059	073	3B	5E	;	Semicolon
0	060	074	3C	4C	<	Left angle bracket
1	061	075	3D	7E	=	Equal sign
1	062	076	3E	6E	>	Right angle bracket
0	063	077	3F	6F	?	Question mark
1	064	100	40	7C	@	"At" sign

Upper-case alphabetic characters (letters):

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
0	065	101	41	C1	A	
0	066	102	42	C2	B	
1	067	103	43	C3	C	
0	068	104	44	C4	D	
1	069	105	45	C5	E	
1	070	106	46	C6	F	
0	071	107	47	C7	G	
0	072	110	48	C8	H	
1	073	111	49	C9	I	
1	074	112	4A	D1	J	
0	075	113	4B	D2	K	
1	076	114	4C	D3	L	
0	077	115	4D	D4	M	
0	078	116	4E	D5	N	
1	079	117	4F	D6	O	
0	080	120	50	D7	P	
1	081	121	51	D8	Q	
1	082	122	52	D9	R	
0	083	123	53	E2	S	
1	084	124	54	E3	T	
0	085	125	55	E4	U	
0	086	126	56	E5	V	
1	087	127	57	E6	W	
1	088	130	58	E7	X	
0	089	131	59	E8	Y	
0	090	132	5A	E9	Z	

More punctuation characters:

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
1	091	133	5B	AD	[	Left square bracket
0	092	134	5C	E0	\	Backslash
1	093	135	5D	BD	]	Right square bracket
1	094	136	5E	5F	^	Circumflex, up arrow
0	095	137	5F	6D	_	Underscore, left arrow
0	096	140	60	79	`	Accent grave

Lower-case alphabetic characters (letters):

Bit	ASCII Dec	ASCII Oct	ASCII Hex	EBCDIC Hex	Char	Remarks
1	097	141	61	81	a	
1	098	142	62	82	b	
0	099	143	63	83	c	
1	100	144	64	84	d	
0	101	145	65	85	e	
0	102	146	66	86	f	
1	103	147	67	87	g	
1	104	150	68	88	h	
0	105	151	69	89	i	
0	106	152	6A	91	j	
1	107	153	6B	92	k	
0	108	154	6C	93	l	
1	109	155	6D	94	m	
1	110	156	6E	95	n	
0	111	157	6F	96	o	
1	112	160	70	97	p	
0	113	161	71	98	q	
0	114	162	72	99	r	
1	115	163	73	A2	s	
0	116	164	74	A3	t	
1	117	165	75	A4	u	
1	118	166	76	A5	v	
0	119	167	77	A6	w	
0	120	170	78	A7	x	
1	121	171	79	A8	y	
1	122	172	7A	A9	z	

More punctuation characters:

Bit	ASCII Dec	ASCII Oct	ASCII Hex	EBCDIC Hex	Char	Remarks
0	123	173	7B	C0	{	Left brace (curly bracket)
1	124	174	7C	4F	|	Vertical bar
0	125	175	7D	D0	}	Right brace (curly bracket)
0	126	176	7E	A1	~	Tilde

Finally, one more nonprintable character:

```
 0    127   177  7F   07    DEL   Delete, rubout
```

Index

Table of Contents

List of Figures

List of Tables