IBM SYSTEM/370 CMS KERMIT USER'S GUIDE

Version 4.3

John Chandler

Harvard/Smithsonian Center for Astrophysics

September 30, 1993

Table of Contents

List of Tables

1. IBM 370 KERMIT

Program: John Chandler (Harvard/Smithsonian Center for Astrophysics); contributions from Vaçe Kundakçi and Daphne Tzoar (Columbia U), Bob Shields (U. Maryland), Greg Small (UC Berkeley), Clark Frazier (Harvard Bus. Sch.), Bob Bolch and Steve Blankinship (Triangle), Ron Rusnak (U. Chicago), Roger Fajman and Dale Wright (NIH), André Pirard (U. Liège), Pierre Goyette (McGill U.)
Language: IBM 370 Assembler
Documentation: John Chandler (CfA)
Version: 4.3
Date: 1993 September

Kermit-370 is a family of programs implementing the KERMIT file transfer protocol for IBM 370-series mainframes (System/370, 303x, 43xx, 308x, 3090, *etc.*) under a variety of operating systems. Kermit-370 operates over asynchronous ASCII communication lines attached to a 3705-style or protocol enveloping 3708 front end (‘‘`TTY`’’ or line-mode 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), to the IBM 3174 protocol converter at level B2.0 or higher (‘‘`AEA`’’ devices), or to front-ends with graphics pass-through mode, such as the Datastream/Leedata 8010 and PCI 1076 (‘‘`GRAPHICS`’’ devices). As of this writing, the pending implementation of full(er) 7171 compatibility in the program product SIM3278 has not been completed. The non-line-mode devices are often called full-screen devices. For more details on front ends, see the section SET CONTROLLER.

The source is coded in IBM 370 assembly language and is compatible with the F, VS, and H assemblers. The code 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 describes 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, these systems are essentially half-duplex; the communication line must "turn around" before any data can be sent. The ‘‘`TTY`’’ devices are strictly half-duplex, and even the ‘‘`SERIES1`’’, ‘‘`GRAPHICS`’’, and ‘‘`AEA`’’ 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 how it must conform to the requirements of the IBM mainframe.

A second distinction is that 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`’’, ‘‘`GRAPHICS`’’, or ‘‘`AEA`’’ device are still in ASCII and therefore bypass the two outer layers. In any case, Kermit-370 translates text files finally into EBCDIC (the inner layer) before storing on disk. When Kermit-370 sends a file, the opposite translations occur. 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. 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 7-bit ASCII-to-EBCDIC translations can be found in the Appendix or the IBM System/370 Reference Card. See the section "Translation Tables" for more details.

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. (See the SET FILE TYPE subcommand.) 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, TSO, MUSIC, ROSCOE, and CICS variants are operational. Variants for DOS-4 and MTS have at least reached the "drawing board," but no others have even been started as of this writing. 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). There are supplemental files in the Kermit distribution with explanations of how to go about porting Kermit-370 and how to add support for new terminal controller types. For details, refer to the installation guide for the variant of your choice.

1.1. Translation Tables

Traditionally, IBM mainframe Kermits have translated 7-bit ASCII characters to 8-bit EBCDIC characters and ignored the "parity" bit in the process. Similarly, the 8-bit EBCDIC characters have been mapped onto 7-bit ASCII, thereby producing many ambiguities in translating the ASCII files back to EBCDIC. These ambiguities fall into two categories: EBCDIC characters not representable in ASCII have been rendered as ASCII nulls, and alternate EBCDIC representations of characters such as the ASCII backslash have been mapped together, but at least no two 7-bit ASCII characters are translated into the same EBCDIC character. The ambiguities were tolerable in environments where the traditionally non-printable characters never occurred in text files, but text processing has increasingly tended to include such characters for mathematical formulas or for languages other than English. Ultimately, the translation tables must become completely invertible, lest information be lost in the transfer. There has long been an option to replace parts of the translation tables via commands from the user (or imbedded in the INIT files), but such replacements were always supported locally and were, therefore, basically non-standard.

The concept of standard translations is currently in a state of flux because of the proliferation of 8-bit code pages and the countervailing efforts at standardization among groups such as the ISO and Kermit developers. In particular, Kermit-370 now supports a set of EBCDIC and "extended ASCII" code pages with built-in translation tables and automatic identification of the "ASCII" transfer character set via Attribute packets. This facility supports files stored using numerous IBM Country Extended Code Pages and permits transfers using character sets ASCII, ARABIC, CYRILLIC, GREEK, HEBREW, JAPAN-EUC, KATAKANA, LATIN1, LATIN2, LATIN3, and THAI. See Table 1-1 for a display of the allowed combinations of character sets. See also file ISOK7.TXT in the Kermit distribution for a somewhat outdated description of the protocol extensions. Kermit-370 currently supports text files in the following languages: Afrikaans, Albanian, Arabic, Bulgarian, Byelorussian, Catalan, Czech, Croatian, Danish, Dutch, English, Esperanto, Faeroese, Finnish, French, Gaelic, Galician, German, Greek, Hebrew, Hungarian, Icelandic, Italian, Japanese (Katakana and Kanji), Lao, Latin, Macedonian, Maltese, Norwegian, Polish, Portuguese, Quechua, Romanian, Russian, Serbian, Slovak, Slovene, Spanish, Swahili, Swedish, Thai, Turkish, Ukrainian, and Volapük. Visual representations of the characters sets may be found in the ISO register (for transfer) and in various IBM documents, such as S544-3156 "About Type" (for files).

Kermit itself normally operates in English, but there are versions with the interactive messages translated into other languages. The currently available languages are Czech, Dutch, Finnish, French, German, Italian, Polish, Portuguese, Russian, and Spanish. Special thanks to Petr Adelsberger, Mauricio Alvarenga, Lorenzo Beltrame, Janusz Bien, Jose Eduardo de Lucca, Richard Gatersleben, Kauko Haumalainen, Jaroslaw Kurowski, Roberto Magana, Alberto Rio, Christian Robert, Gisbert Selke, Karel Smuk, Rob van der Wal, Konstantin Vinogradov, and Joachim Wlodarz for preparing these translations. See the installation guide for details on the alternate-language versions. There are no translations yet of the help files, nor of this document.

	Transfer character set										
Local	ASCII	ARAB.	CYR.	GREEK	HEB.	JAPAN	KATAK.	L1	L2	L3	THAI
CP037	*							**			
CP273								**			
CP275								**			
CP277								**			
CP278								**			
CP280								**			
CP281						*		**			
CP282								**			
CP284								**			
CP285								**			
CP290						*	**				
CP297								**			
CP420		**									
CP424					**						
CP500	*					*		**			
CP838											**
CP870									**		
CP871								**			
CP875				**							
CP880	*		**								
CP905										**	
CZECH									*		
DKOI	*		***								
EBCDIC	*	*						***			
H-EBCD						*	*				
KANJI						*					

Table 1-1: Allowed character set combinations in Kermit-370

All allowed combinations are marked with asterisks, and the preferred combination in each row or column is the one with the most asterisks. Whenever a character set is specified, either directly or through an Attribute packet, the other category of character set is checked to see if it makes up an allowed combination. If not, it is forced to the preferred character set.

As nearly as possible, the tables in Kermit-370 are invertible, but all of the character sets reserve many (typically 65) code points for control characters and leave them officially undefined and unprintable. This applies both to IBM code pages and ISO standard 8-bit character sets. Although 33 of the controls have widely accepted mappings, the others do not, and Kermit-370 currently uses those given in an appendix of IBM's VS/Fortran Reference Manual. Needless to say, such translations are arbitrary and may be invalidated by future decisions at IBM or ISO. Still, most of the translations are likely to be stable in the long run. Table 1-2 shows the current translation from LATIN1 to EBCDIC, which is likely to be the most often used. Other translations, including the reverse ones, may be displayed using the TDUMP subcommand of Kermit-370.

Besides converting files for transmission, Kermit-370 must also compensate for the EBCDIC/ASCII translation performed by some front ends and must, therefore, be able to apply the exact reverse translations on both input and output. This is the "middle layer" of translation described earlier for ‘‘TTY’’ devices. Consider the fate of a Left Square Bracket character in an inbound packet on a ‘‘TTY’’ line. It begins as ASCII code x'5B' (91 decimal), but the front end translates it to EBCDIC before presenting it to Kermit-370. In this example, suppose it becomes EBCDIC code x'4A' (74 decimal) instead of the standard x'AD' (173 decimal). Then, Kermit must be primed with

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
1-	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2-	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3-	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4-	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5-	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
6-	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7-	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
8-	20	21	22	23	24	15	06	17	28	29	2A	2B	2C	09	0A	1B
9-	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	FF
A-	41	AA	4A	B1	9F	B2	6A	B5	BB	B4	9A	8A	B0	CA	AF	BC
B-	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
C-	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
D-	AC	69	ED	EE	EB	EF	EC	BF	80	FD	FE	FB	FC	BA	AE	59
E-	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
F-	8C	49	CD	CE	CB	CF	CC	E1	70	DD	DE	DB	DC	8D	8E	DF

Table 1-2: EBCDIC (hexadecimal) code points for LATIN1

This table shows the values of the EBCDIC equivalents for the code points in the LATIN1 character set. The values are arranged in LATIN1 collating sequence, and the rows and columns are labeled with the first and second digits, respectively, of the LATIN1 code points. For example, LATIN1 code 41 (hex) is upper-case "A", and the intersection of row "4-" and column "-1" has the value C1 (hex), which is the EBCDIC code for "A". Rows "0-", "1-", "8-", and "9-" are officially undefined in ISO 8859-1 and so, in principle, could be changed at some future time, especially "8-" and "9-". Note that this table uses a format close to that of the TDUMP subcommand and of the Kermit code itself, but character-set tables are often displayed with the rows and columns interchanged.

a `SET TETOA 74 91` so that, when reconstructing the original ASCII packet, the character becomes x'5B' again. Otherwise, the packet checksum will appear invalid. Kermit comes with a pair of default tables, but it may be necessary to customize them, as this example shows. The following procedure will reveal any changes needed. The procedure has two stages: the first is for any line-mode front end, and the second only for front ends capable of full 8-bit data transfer. Note that, although the discussion refers to hexadecimal values, the Kermit-370 SET subcommands require decimal numeric arguments. Also, note that this procedure assumes you will run Kermit with TTABLE set on; hence, the references to Kermit-370 tables are to TATOE and TETOA. If you choose to define Kermit's file translation to match that of the front end, you should leave TTABLE set off and remove the T's from the SET subcommands described below. This will also turn off the automatic switching of translation tables according to the Attribute packets received from the other Kermit. Under CMS, you must issue a `CMS SET INPUT` and a `CMS SET OUTPUT` before starting this procedure.

1. Create a file containing all the non-control EBCDIC characters (hex codes 40-FF) and display the file on any available ASCII terminal hooked up to the line-mode front end in question.

2. If any printable ASCII character is missing from the display, Kermit cannot work through this front end (unless you modify the tables in the front end itself).

3. If any ASCII character appears twice, there is no cause for alarm.

4. If any ASCII character does not appear where it should, according to Table 1-3, a SET TATOE must be added to the system INIT file. For example, if EBCDIC code 5F (Not Sign, according to the Appendix) appears as an ASCII Tilde (7E), but EBCDIC A1 (Tilde) does not, a SET TATOE 126 95 is required. Warning: characters considered unprintable by the front end are likely to be filtered out entirely when you display the file; do not expect the display to line up just like Table 1-3.

5. Create a file on the mainframe using an ASCII terminal for input, and enter all 95 printable ASCII

characters in collating sequence. You can presumably save time by skipping the 52 upper- and lower-case letters and the 10 digits.

6. Display the file from the previous step in hexadecimal or other binary form.
7. If any duplicates appear among the 95 characters (or 33, if you have taken the short cut), Kermit cannot work through this front end (unless you modify the tables in the front end).
8. Compare the hexadecimal codes with rows 2-7 of Table 1-2. If a discrepancy appears, a SET TETOA must be added to the system INIT file. For example, if ASCII Left Bracket (5B) appears as EBCDIC 4A, a SET TETOA 74 91 is needed. At this point, the first stage is complete.
9. If 8-bit line-mode file transfer is desired, you must now verify the extended character set. Display the file of EBCDIC codes again, this time using a terminal with extended character set display, or capture the session with a micro Kermit and display the resulting file in hexadecimal. If any code in the range A0-FE does not appear, the front end will not allow 8-bit Kermit data transfers. Generate SET TATOE entries for the entire range, as needed, just as in the example given for 7-bit codes. If the front end's translation tables are documented, it may be easier to work from the manual, but you must be sure that you have tables for all of the translations that occur on the data path.
10. Create a file of the 95 extended ASCII codes A0-FE, if possible, by using a terminal capable of transmitting those codes or transmitting a pre-made file "raw". If necessary, use the appropriate table(s) in the manual(s) instead.
11. Again, if any duplicates appear, this front end is incapable of 8-bit file transfers. Otherwise, generate SET TETOA entries as before.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4-												.	<	(	+	|
5-	&										!	$	*	)	;	^
6-	-	/										,	%	_	>	?
7-										`	:	#	@	'	=	"
8-		a	b	c	d	e	f	g	h	i						
9-		j	k	l	m	n	o	p	q	r						
A-		~	s	t	u	v	w	x	y	z				[		
B-														]		
C-	{	A	B	C	D	E	F	G	H	I						
D-	}	J	K	L	M	N	O	P	Q	R						
E-	\		S	T	U	V	W	X	Y	Z						
F-	0	1	2	3	4	5	6	7	8	9						

Table 1-3: Character graphics for EBCDIC codes

This table shows the representations of the EBCDIC codes arranged in EBCDIC collating sequence, row by row. Codes that do not correspond to ISO 646 characters (7-bit ASCII), have been left blank. One special case is the EBCDIC code 5F, which represents a Not Sign in IBM's CP037, but which is traditionally mapped to the ASCII Circumflex, as shown here. WARNING: this manual may have been printed on a device that maps characters differently. Consult the Appendix to verify.

1.2. File Attributes

Kermit-370 attempts to send and receive file attributes along with the files themselves. Before Kermit receives a file, it compares the Length attribute (if available) with the amount of disk space available (if known) so that the file can be rejected if it will not fit. In addition, the Type, Access, Encoding, Disposition, and Format attributes, if any, are interpreted in order to match the received file to the original as nearly as possible. All other attributes (including Date) are simply ignored. For Type, Kermit recognizes text and binary specifications; for Access, it recognizes append, normal, and supersede; for Encoding, it recognizes ASCII, extended, and EBCDIC (which it treats as binary); for Disposition, it recognizes mail, print, and job; and for Format, it recognizes text, D-binary, V-binary, binary, and LRECL. Any other values are rejected.

On sending a file, if the other Kermit is willing to accept Attribute packets, Kermit-370 sends the Type, Encoding, Format, Date, and Length attributes (unless some or all of them have been disabled -- see the SET ATTRIBUTE subcommand).

Unfortunately, the set of file attributes defined in the Kermit protocol is not well matched to the needs of IBM 370 file systems, so that much of the machinery for creating and interpreting of Attribute packets is useless. For example, the "Format" attribute, which should logically specify the record format of the received file, gives only the format of the file in transmission and cannot distinguish between fixed-length and varying-length records for text files. This limitation is partly due to the fact that other Kermits do not generally support the same attributes. In short, even if the protocol made it possible, some of the important attributes would never be specified anyway. Nonetheless, efforts are being made to extend the protocol to be more comprehensive.

1.3. 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 or commas.

Upon 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 transfer character-set latin1
set file collision rename
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.

Besides knowing the mechanics of entering Kermit subcommands and interpreting responses, the user must understand which subcommands are necessary. The default values for Kermit-370 options have, in some cases, been constrained by a desire for continuity, so that some of the default behavior is distinctly "suboptimal." An example of this is the traditional 80-byte default RECEIVE packet size, which generally gives much slower transmission than a size of 1000 or more. Such options are flagged in this chapter with a notation that they perhaps ought to be set in the INIT files. Note, however, that the interplay of Kermit protocol sometimes provides a performance upgrade with no special action required from the Kermit-370 user. An example of this is the SEND packet-size, which is always under the control of the other Kermit. Kermit-370 always sends packets as long as the other Kermit allows, unless the frequency of transmission errors indicates that shorter packets would be more efficient (see the subcommand SET SPEED).

Kermit-370 also attempts to produce optimal results by adapting to the file attributes sent by the other Kermit along with the files, such as TYPE and LRECL. Such attributes, when sent by the other Kermit, temporarily override the current settings in Kermit-370 during the reception of the associated file. Potentially even more importantly, Kermit-370 automatically recognizes the character set used for the file transfer and chooses, if necessary, a compatible character set for file storage. Table 1-1 shows the currently supported list of character sets. As with any automatic operation, Kermit's honoring of the transmitted attributes may need to be suppressed in part or altogether, and the SET ATTRIBUTE subcommand provides this capability.

Another attribute that Kermit-370 supports is file disposition, which allows files to be received and immediately retransmitted as electronic mail, sent to a printer, or submitted as a batch job. These three options are highly site-specific and are implemented as a set of three host commands with similar calling sequences. In each case, the command is invoked in two different ways in the process of processing the file. It is called without any *filespec* or other options as soon as the corresponding attribute is recognized by Kermit-370. If the command is not implemented or for some reason returns a non-zero completion code (such a code should be negative, if at all possible), Kermit-370 rejects the file using the normal attribute exchange mechanism. Otherwise, the file is received as usual, and the command is invoked again at the end-of-file. The second time, the command is given the name of the received file and the options received from the sending Kermit (such as the list of recipients for electronic mail). The command is then responsible for disposing of the file (and deleting it, if desired). If, for any reason, the file cannot be sent as required, the command should return a negative completion code (or non-zero, at any rate), and Kermit-370 will send back an error message to the sending Kermit. The command itself should refrain from issuing any messages of its own or permitting commands that it invokes to issue messages. See the beginning of the chapter on the system-specific variant of Kermit-370 for the exact command syntax. Some sample implementations of such commands may be available in the Kermit distribution -- refer to the system-specific installation guide for a list of samples.

1.4. Kermit-370 Subcommands

The following is a 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.
GIVE*	creates a TAKE file snapshot of a table.
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.
STOP	easy escape from protocol mode.
TAKE*	subcommands from file.
TDUMP*	dumps the contents of a table.
TYPE*	a file.
VERSION*	of Kermit-370.
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. To help avoid confusion, this chapter will often use the term "foreign" to apply to the Kermit at the other end from Kermit-370. All the above subcommands may be preceded by the LOCAL prefix, but only certain ones are valid with REMOTE, including some not shown here. See the description of the SERVER subcommand for details. Any text replies Kermit-370 gets from the foreign 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 describes subcommands with special meaning or use in Kermit-370, except the highly system-dependent ones. For the latter, refer to the appropriate chapter. Subcommands are listed in alphabetical order.

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 sends the text as a raw transmission according to the current CONTROLLER setting. Thus, XECHO will, if necessary, break the text into pieces no larger than the current SEND PACKET-SIZE and will use transparent mode if CONTROLLER is SERIES1, GRAPHICS or AEA. 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 ''`^`''. XECHO also decodes 8th-bit quoting, just as in Kermit protocol, and this feature can be disabled by setting 8th-bit quoting off. For example, if the 8th-bit quote character is ''&'', entering the subcommand ''`XECHO &A`'' will transmit a code x'C1', but the same subcommand with quoting off will transmit two bytes: x'2641'.

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. 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 GIVE Subcommand

Syntax: `GIVE` *table-name filespec*

This compares the named translation or selection table with its current 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 translation tables, and CONTROL-CHAR is the selection table. The details of the *filespec* are system-dependent, but those details will, in general, be the same as for the TAKE subcommand (*q.v.*). In the case of tables ATOE and ETOA, the current defaults are the values from the most recent character-set definition, if any, but the defaults for TATOE and TETOA always remain at the initial values. See the SET FILE CHARACTER-SET and SET TRANSFER CHARACTER-SET subcommands for more details.

The HINTS Subcommand

Syntax: `HINTS`

This subcommand produces a screenful of suggestions for Kermit operation, including warnings about any current settings that may be undesirable.

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.

When Kermit-370 is in (non-local) server mode, you must avoid sending it any HOST commands that trigger full-screen terminal I/O, since the server-client interface does not provide any full-screen terminal emulation, nor is Kermit-370 usually able to intercept such I/O in any case.

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. Note, however, that the specified text must not begin with a second KERMIT prefix.

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. When files are received, their names are recorded in a transaction log in memory and may be viewed later via the TDUMP NAMES subcommand. The log contains the names along with the file sizes and any relevant error messages. Another form of log may be provided through an accounting exit routine, which is called at the end of every RECEIVE session. Such an accounting log may contain elapsed time and numbers of bytes sent, received, and transferred to/from disk. If an error occurs during the file transfer, as much of the file as was received is saved on disk. If, however, the sending of a file is cancelled by the user of the foreign system, Kermit-370 will discard whatever had arrived, unless FILE COLLISION is APPEND or INCOMPLETE is KEEP.

Kermit-370 has a context-dependent maximum record length, and received records longer than that will be folded or truncated to the proper length; when this happens, Kermit may or may not stop, depending on the FILE subparameter LONGLINE. If truncation does occur, Kermit will later note the fact as an error (unless something more serious happens in the meantime). In addition, when a fixed record length is specified, received records are padded to the correct length. 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, *i.e.*, folded. For more details on the RECEIVE subcommand syntax and operation, see the chapter on the desired system-specific variant of Kermit-370 under both RECEIVE and SET FILE. See also (in this chapter) the SET FILE LONGLINE subcommand for details on record truncation and folding.

The SEND Subcommand

Syntax: `SEND [`*filespec*`[<`*options*`>] [`*foreign-filespec*`]][, ...]`

The SEND subcommand tells Kermit-370 to send a file or file group to the other (foreign) Kermit. If no such file exists, Kermit-370 simply displays an error message and issues another prompt. Like RECEIVE, SEND keeps a transaction log of all files transferred in a group and calls the accounting exit routine at the end of the session. If this subcommand is issued without any arguments, Kermit-370 will prompt the user for both native and foreign *filespec*s (and will insist on getting the former, but will do without the latter). Either with or without prompting, SEND allows specifying a list of up to 13 files (or file groups) separated by commas. (By ending the command line or subsequent response with a comma, the user ensures further prompting.) In this syntax, the *options* enclosed in angle brackets may be regarded as part of the native *filespec*. There must be no intervening blanks. At present, the only supported option is a range of line numbers within the file to be sent. The range takes the form [*n*][-[*m*]], where *n* is the number of the first line to send (counting from 1), and *m* is the last. Omitting either number implies the corresponding limit is the physical beginning or end of the file. This same option syntax may be used in some commands issued to the foreign Kermit, namely, those involving *filespec*s in 370 form. In particular, GET and REMOTE TYPE can use this syntax. Note that a trailing ''>'' is used by some other Kermits, such as MS-Kermit, to indicate redirection of the output from REMOTE commands to disk. In order to transmit the ''>'' and also prevent the redirection to disk, it would be necessary to add an explicit redirection to the terminal, as in

```
REM TYPE filespec<n-m> > CON
```

Although file transfers cannot be cancelled from the 370 side, Kermit-370 is capable of responding to "cancel file" or "cancel batch" signals from the local Kermit; these are typically entered by typing CTRL-X or CTRL-Z, respectively.

When Kermit-370 sends files using long packets (longer than 94), the throughput is especially sensitive to the level of noise on the line because retries are so time-consuming. Therefore, Kermit-370 imposes an extra, heuristic size limit on packets when retries have been found necessary. When that is the case, after every 15 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. The algorithm is explained in *Kermit News* V. 3 #1. For more details on the SEND subcommand syntax and operation, see the chapter on the desired system-specific variant of Kermit-370.

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.
DELETE	a file or files.
DIRECTORY	display file attributes.
HELP	display a command summary, such as this.
HOST	execute a system command.
KERMIT	execute a Kermit-370 subcommand.
PRINT	send a file to be printed by the server.
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 client 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 first form is also distinguished by the fact that the line range may be specified in the same manner as in the SEND subcommand. The syntax of the others is system-dependent.

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:

ATOE	Modify the Kermit-370 ASCII-to-EBCDIC table.
ATTRIBUTE	Determine A-packet generation.
BLOCK-CHECK	Level of error checking for file transfer.
CONTROLLER	Indicate type of terminal connection.
CONTROL-CHAR	Set prefixing state.
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...
CHARACTER-SET	... for 370 storage.
COLLISION	... treatment for duplicate names.
LONGLINE	... treatment of too-long records.
OVERWRITE	... treatment of attributes.
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	
PROMPT	For Kermit-370 subcommands.
RETRY	Maximum retry count...
INIT	... for initial packet exchange.
PACKET	... per packet for ongoing transfer.
SERVER-TIMEOUT	Spacing between server NAK's.
SPEED	Line speed for packet-size calculations.
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.
TRANSFER	Options for transmission...
CHARACTER-SET	... of text files.

LOCKING-SHIFT	... protocol extension.
TTABLE	Determine which tables undo the terminal translation.
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.
PARITY	Indicate if 7-bit or 8-bit data.
QUOTE	Use to quote control characters in packets.
START-OF-PACKET	Packet beginning marker.
TIMEOUT	Time limit for response.

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. Helpful hint: if you have files that make use of extended (8-bit) ASCII codes and wish to upload them via Kermit-370, be sure to define unique EBCDIC equivalents of all the needed 8-bit ASCII codes or else treat such files as binary data. Any time you use the SET ATOE or SET ETOA subcommands, that has a side effect equivalent to SET ATTRIBUTE ENCODING OFF (*q.v.*). If the extended ASCII character set is one of those supported for file transfer in Kermit-370, you need only issue a SET TRANSFER CHARACTER-SET (*q.v.*).

Note: the meaning of the tables depends on the TTABLE setting -- if TTABLE is OFF, the TATOE and TETOA tables are not used.

SET ATTRIBUTE

Syntax: `SET ATTRIBUTE [`*attribute*`] ON` *or* `OFF`

The individual *attribute*s are LENGTH, TYPE, DATE, CREATOR, ACCOUNT, AREA, PASSWORD, BLOCKSIZE, ACCESS, ENCODING, DISPOSITION, PROTECT, ORIGIN, FORMAT, SYS-INFO, and BYTE-LENGTH. Kermit-370 distinguishes between the two forms of this subcommand by counting "words". In order to see the list of supported attributes, you must enter ‘‘`SET ATTR ? ?`’’; if you enter just ‘‘`SET ATTR ?`’’, Kermit will list just the alternatives ON and OFF.

ON The specified attribute is to be processed, or attribute packets are generated for all outgoing files, provided the other Kermit indicates the ability to accept them. (Default).

OFF The specified attribute is to be ignored and not generated, or attribute packets are never generated.

SET BLOCK-CHECK

Syntax: `SET BLOCK-CHECK` *type*

This determines the type of block check used during file transfer, provided the other Kermit agrees. Valid options for *type* are: 1-byte (for a one-character checksum), 2-byte (for a two-character checksum), 3-byte (for a three-character CRC), and Blank-free-2 (for a shifted two-character checksum that avoids using blanks. 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, AEA, FULLSCREEN, VTAMTTY, or NONE. Kermit-370 automatically determines whether you are connected via a Series/1 (or similar) emulation controller or a TTY line. In some circumstances, such as when the connection is through a non-graphics-capable 3174 port, Kermit will set CONTROLLER to NONE, which has the effect of disabling file transfers. This subcommand is provided, though, to allow the automatic choice to be superseded, and because Kermit may not be able to distinguish between Series/1-type and other 3270-emulation controllers. In particular, there is no way to distinguish between FULLSCREEN and GRAPHICS from within Kermit. When CONTROLLER is set to SERIES1, GRAPHICS, or AEA, Kermit disables the 3270 protocol conversion function by putting the terminal controller into "transparent mode", which allows Kermit packets to pass through intact. Note: an incorrect CONTROLLER setting may lock up or wipe out your session when you try to transfer files.

Kermit operation is possible through an IBM 3708 front end, but only in a rather specific configuration. See the installation guide for your variant of Kermit-370 for the details of that and other hardware-related restrictions and configurations.

SET CONTROL-CHAR

Syntax: `SET CONTROL-CHAR` *mode* [*number*]

The *mode* may be PREFIXED (normal Kermit protocol for control characters) or UNPREFIXED. In the latter case, the specified control character (given as a decimal number in the range 0-31 or 128-159) is transmitted "as is". If the *number* is omitted, then all control characters are set accordingly. Kermit-370 automatically overrides the user's settings for certain characters used for Kermit protocol: the start-of-packet, the end-of-packet, the handshake (if set), and XOFF (not really used, but too dangerous to send). Sending control characters without prefixes can speed up transfers of binary files.

The status of prefixing is initially "PREFIXED" for all control characters. It can be displayed by the TDUMP CONTROL subcommand, or saved in the form of a TAKE file by the GIVE CONTROL subcommand (*q.v.*).

SET DEBUG

Syntax: `SET DEBUG OFF` *or* `ON [RAW] [I/O] [SAVE] [LONG] [TIME]`

Note: any combination, in any order, of RAW, I/O, SAVE, and TIME may follow or replace ON. Each of the three implies ON.

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, GRAPHICS, or AEA.

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 or VTAMTTY terminals, and ASCII for SERIES1, GRAPHICS, and AEA terminals. This option is generally not recommended; I/O is preferable.

I/O The same as ON, but the log includes additional transmission status information, such as the AID returned by a full-screen device. See below for a summary of the log formats.

SAVE The same as ON, but the log file is closed after each entry is added, so that, if the session is abnormally terminated, the log file will be complete and readable.

LONG The same as I/O, but the additional information is not truncated to 36 bytes.

TIME Used only with I/O or LONG. The lines of hexadecimal dump are tagged with the time of day from the CPU clock, truncated to the nearest second.

OFF Stop logging packets and close the the log file. (Default.)

Often, problems with Kermit file transfers or server-mode operations can be diagnosed by setting DEBUG on in one or both Kermits, regardless of where the problems actually lie. For Kermit-370, the maximum amount of information can generally be obtained by setting DEBUG to I/O or LONG, but the format of the log depends somewhat on which variant of Kermit-370 is involved. Before examining the log, you should set DEBUG OFF either explicitly or by exiting from Kermit-370.

There is an optional feature for logging dumps of storage blocks at selected points in the execution. By default, this feature is disabled (at assembly time) by having the variable symbol &KTRACE set to NO and also by virtue of the fact that no calls to the dump routine are present in the distribution code. Further, the dumps are suppressed unless both DEBUG and TEST (*q.v.*) are set on. To select when and what blocks to dump, it is necessary to insert calls to KHDMP at appropriate points in the source before assembling. Each such call generates a dump each time it is executed, provided that DEBUG and TEST are set, and the contents of all registers are preserved. The call specifies the starting address, the length, and a short title for the block. Only eight characters of the title will be used. Some examples:

`KHDMP ATOE+128,128,'ATOEhigh'`
: Dump the 2nd half of the ATOE table.

`KHDMP KHDSAV,20,'R14 - R2'`
: Dump registers 14-2.

`KHDMP 32(,13),40,'**R3-R12'`
: Dump registers 3-12.

`KHDMP (3),(0),'**QBLOCK'`
: Dump block addressed by R3 with length specified in R0.

There is another debugging facility that is enabled by the variable symbol &KTRACE along with the dump option, namely, an execution trace. There is a circular buffer of trace elements in Kermit's working storage, and a new element is written each time a Kermit subroutine is called or returns. The elements contain the subroutine name plus (on entry) a sequence number and the contents of registers 0 and 1 or (on exit) the character ‘‘`>`’’ and the contents of registers 15 and 1. There is also a mechanism for tracing extra events by inserting KTRACE calls into the source. For example,

`KTRACE 0(5),REGS=5`
: Trace eight bytes pointed to by R5 and R5 itself.

`KTRACE FOOBAR`
: Trace eight bytes at label FOOBAR.

`KTRACE 'Found it',REGS=(1,7)`
: Trace "Found it", R1, and R7.

The trace table is simply updated in storage, eating its own tail. It can be found in a memory dump by locating the "eye-catcher" that says ‘‘`KTRACE:`’’, which precedes the start, current, and end pointers for the table. The table is also accessible interactively via the TDUMP subcommand (*q.v.*).

Each line in the debug log begins with a one-letter tag and a colon and contains information according to the tag. The following tags are defined.

S: The text of a packet sent. Normally, it will be encoded in EBCDIC for convenience, but if DEBUG is set to RAW, the packet will appear exactly as passed to or from the system, i.e., in EBCDIC for TTY or VTAMTTY lines and in ASCII for full-screen lines.

R: The text of a packet received. The same encoding applies.

A: The AID and buffer address returned by a full-screen device along with a read operation (three characters in all). The values should all be printable EBCDIC. This obsolete tag was used by TSO and MUSIC Kermits and appeared only when DEBUG was set to I/O.

*: Data dumped by the optional KHDMP routine.

The following tags appear only when DEBUG is set to I/O or LONG. All values are in hexadecimal. The meanings differ slightly according to the operating system. For CMS, the I/O parameter list is a channel command; for TSO or ROSCOE, the SVC 93 (TPUT/TGET) parameters; for CICS, an intermediate string similar to channel commands. Similarly, the status data consist of the stored CSW plus an attention interrupt indicator for CMS, but the return code from the I/O operation for TSO, ROSCOE, CICS, and MUSIC.

a: Channel and device status after an unexpected attention interrupt.

b: I/O parameter list for recovering from a CP break-in on screen.

c: I/O parameter list for resuming normal screen operation.

d: Data transferred on the previously indicated I/O operation.

e: Status data after an I/O command has completed with an error.

g: I/O parameter list for reading from the screen buffer.

i: Status data after an I/O command has completed normally.

m: I/O parameter list for displaying text on the screen.

o: I/O parameter list for initializing the screen for transfers.

r: I/O parameter list for reading from the terminal.

w: I/O parameter list for a transparent write.

?: I/O parameter list for some other operation, such as clearing the screen.

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 subcommands like RECEIVE, SERVER, REMOTE, and the like.

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 CHARACTER-SET

Syntax: `SET FILE CHARACTER-SET` *name*

Specifies the name of the character set used in files stored on disk. This setting may be superseded by an Attribute packet of an incoming file. Currently, the available names are CP037, CP273, CP275, CP277, CP278, CP280, CP281, CP282, CP284, CP285, CP290, CP297, CP420, CP424, CP500, CP838, CP870, CP871, CP875, CP880, CP905, CP1047, CZECH, DKOI, EBCDIC, H-EBCDIK-DASH, and KANJI (or FUJITSU-KANJI, HITACHI-KANJI, or IBM-KANJI). The names beginning with CP refer to IBM code pages, while DKOI is the Cyrillic standard GOST 19768-87 used in the USSR, CZECH is a character set sometimes used in Czechoslovakia, and EBCDIC (the default) is the traditional *de facto* standard EBCDIC character set. A character set other than the default may be required by local conventions and, if so, should be specified in the system or user INIT files. See Table 1-1 for the allowed combinations of transfer and file character sets. Explicitly setting this option has a side effect equivalent to issuing SET ATTRIBUTE ENCODING ON (*q.v.*).

The name KANJI is actually just an alias for the local preferred proprietary Kanji code (Fujitsu, Hitachi, or IBM).

The various Kanji character sets have two-byte codes (DBCS), but are used with one-byte code pages (SBCS) as well. It is normally necessary to issue two SET FILE CHARACTER-SET SUBCOMMANDS in order to set up for a DBCS: first, selecting a compatible SBCS, and, second, selecting the DBCS itself. If the current SBCS is incompatible with the selected DBCS, a default will be chosen. This default, along with the choice of the particular proprietary character set associated with the alias KANJI, is chosen by the installer.

SET FILE COLLISION

Syntax: `SET FILE COLLISION` *action*

Specifies the action to take when an incoming file has the same name as an existing one. Two of the options involve choosing an alternative, unique name similar to the one in conflict, but the details of choosing are system-specific. Typically, the method involves adding digits to the existing name.

APPEND The new file is appended to the old one. This option has the the same effect as the old subcommand SET APPEND ON.

BACKUP The existing file is renamed, and the new file is given the desired name as if no conflict had occurred.

DISCARD The incoming file is rejected by returning a "cancel file" indication on any Data packets.

OVERWRITE The existing file is overwritten with the incoming file. This is the default for the CMS, TSO, ROSCOE, and MUSIC variants. Changing this default is a good candidate for INIT files. When COLLISION is set to OVERWRITE, the attributes of the new file are determined by the current setting of FILE OVERWRITE (*q.v.*).

RENAME The incoming file is renamed so as not to destroy (overwrite) the pre-existing one, and the new name is returned to the sending Kermit for information purposes. This has the same effect as the old subcommand SET WARNING ON. This is the default for the CICS variant.

SET FILE LONGLINE

Syntax: `SET FILE LONGLINE FOLD` *or* `TRUNCATE` *or* `HALT`

This specifies the action to take when a received line is longer than the current maximum record length. That length is determined by the context in a system-specific way. Refer to the description of the RECEIVE subcommand in the appropriate chapter for details.

FOLD Specifies that long lines are to be split into two or more records as needed, all but the last being of the maximum length. No null records are created when the received line is an exact multiple of the record length. BINARY files are always considered, by definition, as a single line and, therefore, are *always* folded, regardless of the setting of this parameter. V-BINARY and D-BINARY files, on the other hand, are reconstructed by folding at points determined by the context within the received file, and cannot be folded further to fit the current maximum length. Any such records that are too long will be truncated.

TRUNCATE Specifies that long lines are to be truncated at the maximum length. (Default.) Kermit-370 takes note of the number of such truncations performed on a file and reports it in the STATUS message and also treats the fact of truncation as an error when the file transfer is complete. The file will have been transferred, but obviously not quite intact. However, this option can be useful for some tasks, such as stripping sequence numbers from card images. The process is similar to that provided by the SET MARGIN RIGHT subcommand for sending files.

HALT Specifies that a file transfer is to halt immediately if a received line is too long. Kermit-370 then issues an error packet and stops the transfer.

SET FILE OVERWRITE

Syntax: `SET FILE OVERWRITE DEFAULT` *or* `PRESERVE`

DEFAULT Specifies that the current file attribute settings are to be used for the new file. The result is roughly the same as if the old file, if any, were completely erased before the new file is received. (Default.)

PRESERVE Specifies that the attributes of the file being overwritten are to be retained and used for the new file. The result is roughly the same as if the old files contents were deleted, and the new file were appended to the empty stub.

SET FILE TYPE

Syntax: `SET FILE TYPE` *type*

Specifies the type of data comprising files to be sent or received. This setting may be temporarily superseded by the Attribute packets for a file being received.

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.) Note: trailing blanks are removed from outgoing, varying-length records only if they consist of a single blank each, or if there is a right margin specified (*q.v.*). Further note: a given file is intrinsically categorized as fixed-length or varying-length (or undefined-length) by the file system and cannot be changed simply by setting the FILE RECFM parameter in Kermit. Conversion between formats is a system-specific function; see the system documentation or consult your local support staff for details on conversion techniques.

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 FILE COLLISION is APPEND, 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 variants of Kermit-370 do not support any alternate lines. No variant 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 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 variant of Kermit-370 followed by a ">" sign, *e.g.*, `Kermit-CMS>`. If the *string* is omitted, normal system prompting will occur.

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 on 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 SERVER-TIMEOUT

Syntax: `SET SERVER-TIMEOUT` *time*

This defines the *time* in seconds that Kermit-370 in server mode should wait for a command before sending a NAK packet. The default is 120. A value of 0 means that Kermit should wait indefinitely, not only in the server loop, but in all transfers, regardless of the timeout value specified by the other Kermit. Some variants are unable to time out in any case. Also, timeouts are not implemented for any of the full-screen terminal controllers.

SET SPEED

Syntax: `SET SPEED` *number*

This determines the communication line speed assumed by Kermit-370 in calculating the optimum packet size. If the value is zero, such calculations are suppressed. This option is purely informative and has no effect on actual line speed. (Default 1200.)

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 and must be in strictly increasing order.

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 type 1 checksum testing on received packets.

OFF Normal operation. (Default.)

SET TRANSFER CHARACTER-SET

Syntax: `SET TRANSFER CHARACTER-SET` *name*

Specifies the name of the character set used in sending or receiving files. This setting may be superseded by an Attribute packet of an incoming file. Currently, the available names are ASCII (the default), ARABIC, CYRILLIC, GREEK, HEBREW, JAPAN-EUC, KATAKANA, LATIN1, LATIN2, LATIN3, THAI, and TRANSPARENT. There are also special aliases L1, L2, and L3 for the LATINx names. All but JAPAN-EUC, TRANSPARENT, and ASCII represent 8-bit codes composed of a pair of 94- or 96-character sets from the ISO registry combined with normal definitions for the so-called C0 and C1 characters. JAPAN-EUC is a DBCS for encoding Kanji characters, plus Roman, Greek, and Cyrillic. ASCII is the traditional character set supported by Kermit, but one of the newer, 8-bit sets would be preferable for most users. This option is, therefore, a good candidate for inclusion in the system INIT file. See Table 1-1 for the allowed combinations of transfer and file character sets. Explicitly setting this option has a side effect equivalent to issuing SET ATTRIBUTE ENCODING ON (*q.v.*). There is one exception, namely, TRANSPARENT, which sets ENCODING OFF and replaces both translation tables with null operations, regardless of the current nominal file character set.

The biggest drawback of the built-in tables for the various character sets is that neither the ISO registry nor IBM defines any mapping between the C1 characters (hex 80-9F in ISO arrangements) and the characters of EBCDIC code pages. Thus, the mappings in Kermit-370 tables are somewhat arbitrary, and future pronouncements may suddenly invalidate some or all of those 32 mappings.

SET TRANSFER LOCKING-SHIFT

Syntax: `SET TRANSFER LOCKING-SHIFT ON` *or* `OFF` *or* `FORCED`

ON The Kermit locking-shift protocol is to be used in transfers to or from cooperating partners, provided that 8th-bit quoting is enabled.

OFF The Kermit locking-shift protocol is not to be used.

FORCED The Kermit locking-shift protocol is to be used, regardless of the cooperation of the other Kermit. The encoding uses only the locking shifts, to the exclusion of 8th-bit quoting.

SET TTABLE

Syntax: `SET TTABLE ON` *or* `OFF` *or* `KP`

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, GRAPHICS, and AEA connections.

OFF The ATOE and ETOA tables are used for all translations by Kermit-370. (Default.)

KP Same as ON, but also establishes values in the TATOE and TETOA tables based on IBM's corporate standard ASCII/EBCDIC translation (distinct from the internationally accepted *de facto* standard).

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. The default value is an ampersand.

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.

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. For all parameters besides QUOTE, the operational value is the RECEIVE. 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. Specifying a value greater than 94 is necessary and sufficient to enable the long-packet protocol for transfers to Kermit-370 (provided the other Kermit is willing). Kermit-370 will actually accept long packets in any case, but the protocol requires that the other Kermit not send them unless Kermit-370 asks. Raising this value from the default is a good candidate for inclusion in INIT files. In practice, the packet 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). It may also be used to specify the packet size for a "raw" download via the XTYPE subcommand. This parameter has no other function and is completely irrelevant to long packets. If the other Kermit asks for long packets, Kermit-370 will always comply.

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.

PARITY

Syntax: `SET` *mode* `PARITY MARK` *or* `NONE`

RECEIVE specifies the parity expected in the transparent-mode ASCII data received by the mainframe from a full-screen device. Such data will typically have either all Mark parity (seven data bits with the eighth bit set) or no parity (eight data bits). This is typically not the same as the parity used in communications between the protocol convertor and the terminal. 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.

SEND is also an operational value, specifying the parity to be used in constructing outgoing data packets on full-screen devices. NONE is the default and is generally preferable, in that it permits binary transfers without the need for eighth-bit prefixing, but MARK may be required in some configurations.

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 need to be changed to establish contact.

SEND may also need to be changed 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 may be needed to define the *time* in seconds Kermit-370 is to wait for a response from the other Kermit in the initial packet exchange, although the default value 0 (indefinite wait) is probably satisfactory, especially since Kermit-370 in many cases cannot time out anyway. Specifying a non-zero value will prevent the other Kermit from ever requesting infinite "patience" from Kermit-370.

The SHOW Subcommand

Syntax: `SHOW [`*option*`]`

The SHOW subcommand displays the values of all parameters that can be changed with the SET subcommand, except CONTROL-CHAR, 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*. When ‘‘`ALL`’’ is specified or implied, all parameters other than the attribute switches are displayed.

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 there are several exceptions. If an invalid subcommand is entered, the status becomes ‘‘`Kermit command error`’’, and the next subcommand entered will reset the status. Also, 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).

Other information is also included. When Kermit-370 has been forced to truncate one or more records in the last 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 line (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. Further, 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. If TEST is set on, Kermit also reports the maximum size attained by its storage stack since execution began.

The STOP Subcommand

Syntax: `STOP`

This is not a subcommand in the usual sense. Instead, it is a command string that can be entered on the communication line while Kermit-370 is in protocol mode and will cause protocol mode to cease immediately. This may be useful if the other Kermit has crashed. The word "stop" may be entered in either upper or lower case, but it must be the only character string in the "packet" in question. If you are using a full-screen terminal, and if other information appears on the screen, you must clear that other text from the screen (using CLEAR EOF) before pressing ENTER.

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, and the nesting may continue to a depth of ten. If a TAKE file includes the subcommand SERVER, however, the nesting count is saved and starts over again in server mode in case the client Kermit should transmit a REMOTE KERMIT TAKE command. 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 TDUMP Subcommand

Syntax: `TDUMP` *table-name* *or* `NAMES` *or* `TRACE`

This displays the contents of *table-name*. The same table can be modified using the SET subcommand. The ATOE, ETOA, TATOE, and TETOA translation tables and the CONTROL-CHAR prefixing selection table can presently be displayed and changed. The NAMES table is the transaction log for the last transfer, consisting of the *filespec* of each file sent or received, along with the size (in Kbytes) and any error messages. If the Kermit TRACE facility is enabled, the TRACE table may be displayed (and destroyed in the process). This table contains entries for subroutine calls and returns during program execution, but Kermit normally does not have the facility enabled. See SET DEBUG for more details on execution tracing.

The TYPE and XTYPE Subcommands

Syntax: `[X]TYPE` *filespec*

These subcommands display the named file. TYPE is effectively a synonym for (and allows the same options as) the host system command for displaying files at the terminal, but XTYPE performs a raw file transfer on the current communication line (which need not be the terminal) according to the current CONTROLLER setting. Thus, XTYPE uses transparent mode if CONTROLLER is SERIES1, GRAPHICS, or AEA. Also, it sends the data in bursts no larger than the current SEND PACKET-SIZE. Since XTYPE is basically a modified SEND, the options allowed on the *filespec* for SEND are also allowed for XTYPE.

The VERSION Subcommand

Syntax: `VERSION`

This subcommand displays the program version number and date.

1.5. Before Connecting to the Mainframe

Several options must be set in the micro Kermit before connecting to an IBM 370 system as a line-mode device. You should set LOCAL-ECHO 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 LOCAL-ECHO to 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 Kermits, 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 full-screen device also requires that certain options be set in the micro Kermit. You should set LOCAL-ECHO 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 variants of Kermit-370 (notably CMS) cannot provide timeouts, and you may need to set the TIMER to ON in the micro.

When you are connecting through a protocol convertor, it is useful to know the key sequence that causes the screen image to be repainted from the controller's memory. In many cases, it is CTRL-V, although CTRL-G and CTRL-C are also sometimes used. In general, this sequence should be typed whenever reconnecting to Kermit-370 after being in Kermit protocol mode (and sometimes after merely escaping to the local Kermit), since the local Kermit may have modified the screen.

1.6. Trouble-shooting Protocol Converters

Many, but not all, protocol converters have transparent modes that permit Kermit file transfers. The welter of competing and often incompatible communications devices would cause a major headache, except for three circumstances. First, Kermit-370 has routines for automatically detecting which kind of front end is controlling the current session; second, the Kermit installer is encouraged to tailor Kermit to force the correct choice of `CONTROLLER` whenever those routines don't work properly; and, third, Kermit offers a last-resort mode of operation that will work with almost any protocol converter. Because of the limitations in the catch-all mode (known as `FULLSCREEN` mode), it is still best to take advantage of the transparency, if any, in the protocol converter, and the automatic detection routines still play an important role. It will be instructive to outline what those routines actually do and how they can go wrong.

Recognizing a Series/1

Although protocol converters are advertised as simulating the behavior of IBM 3270-type terminals, there generally are differences which could be used to distinguish each type of device from the others and from real 3270-type terminals. However, all that really matters to Kermit-370 is whether there is a transparent mode available such that file transfers can be carried out. To date, only three fundamentally different transparent modes have been reported to Columbia, and it seems likely that no others have been (or, perhaps, ever will be) implemented. All are supported by Kermit.

Kermit-370 recognizes these front ends automatically by making two simple tests. The first takes advantage of one of the advanced features first implemented in the Yale ASCII system and subsequently copied in many of the devices that adopted the same transparent mode. This feature is a special 3270 data-stream order which requests a status report from the protocol converter. Kermit sends this order and then reads the "3270 screen". If Kermit sees a valid status report, it sets `CONTROLLER` to `SERIES1` and stops testing.

Two catches

Obviously, the Yale status order is not implemented in most other kinds of hardware. Thus, the order would be rejected by a non-Yale-type controller, and that could have undesirable side effects on the hardware. However, hardware is generally designed to be robust -- the real drawback lies in the side effects on certain communications software (notably VTAM/TSO), which may respond badly while trying to protect the robust hardware from illegal orders. If it proves impossible to make the external software behave properly, the only recourse is to modify Kermit-370 to skip the first test altogether and possibly to force the `CONTROLLER` setting; this modification is described in the relevant "Beware" file in a note dated 89/2/27.

Catch Two is that the status order is not implemented in all of the devices that support Yale-ASCII-style transparent mode. This means that some devices "fall through the cracks" in this procedure. A site where such devices are used may find it expedient to modify Kermit (following the same "Beware" pattern) to force the procedure to set `CONTROLLER` to `SERIES1` (assuming there are no other protocol converters also in use that support one of the other transparent modes). A list of such devices can be found in a footnote in the Kermit distribution file `ik0aaa.hlp`.

Recognizing a 3174

The second diagnostic test uses a hardware command (Read Partition Query) that is defined by IBM, but is not implemented on all 3270-type equipment. There is, thus, the same danger as in the first test, but the danger appears to be slight. Indeed, both CMS and TSO allow a user program to know in advance whether a Query is permitted. The Query response consists of one or more structured fields, and the 3174 AEA ASCII Graphics system (the only device with the `AEA` style of transparency) is easily identified by the appearance and content of a particular type of field. In fact, it is possible to tell from the Query data whether the particular 3174 line is allowed to use the ASCII Graphics transparency. Therefore, this test has three possible outcomes: Kermit may detect a transparency-enabled 3174 line (and set `CONTROLLER` to `AEA`); it may detect an incapable 3174 line (and set `CONTROLLER` to `NONE`); or it may detect "none of the above" (and set `CONTROLLER` to `GRAPHICS`). Thus, aside from the exceptions already noted, `GRAPHICS` simply means that the front end either supports SAS-style transparency or none at all.

Fallback positions

What should you do when the automatic detection fails? Obviously, the first thing is Be Prepared. Often, the misbehavior of VTAM can be halted by pressing ENTER or PA1, so you should be sure to know how to generate a PA1 when trying out Kermit on an unfamiliar type of protocol converter. Also, you should know what kind of transparency to expect for the front end and verify that Kermit-370 has, in fact, set `CONTROLLER` appropriately. This means checking the list of devices in the Kermit distribution file `ik0aaa.hlp`. If your configuration is listed as *un*supported, you may be wasting your time, but the list is not necessarily up-to-date. If your configuration is not listed at all, you have the opportunity to be a pioneer and report your findings back to Columbia for inclusion in future editions of the list. There are a few rules of thumb for quickly deducing the controller type by reading the manuals for the device; the manuals may not be specific enough, but this is clearly the easiest way of determining whether Kermit can support a given device and which controller type is applicable. The rules are as follows (in order of simplicity and likelihood):

1. GRAPHICS or SERIES1 may be implied when the device has a transparent or graphics mode described as compatible with that of a supported device listed in `ik0aaa.hlp`.
2. SERIES1 is implied when the device runs the "Yale ASCII Communication System" or something with a similar name.
3. GRAPHICS is implied if the manual mentions the SAS Institute in the context of ASCII graphics.
4. GRAPHICS is implied when output transparent data may be preceded by a WCC (Write Control Character) and 70 (hex).
5. SERIES1 is implied when transparent data must be preceded by a WCC and either 115D7F110005 (write-read) or 115D7F110000 (write-only).
6. GRAPHICS or SERIES1 may be implied when some of the manufacturer's other products are listed in `ik0aaa.hlp`, and all are shown as being of one type.
7. If none of the above rules apply, but the manuals describe a transparent mode in detail, the device may be a totally new type. The distribution file `ik0con.hlp` has hints on implementing Kermit support for the new type.
8. If nothing else works, you can probably use FULLSCREEN mode, as long as the micro Kermit supports it.

If Kermit tries to transfer a file with the wrong `CONTROLLER` value, there is a distressing possibility for the session to lock or, at least, appear to lock. When and if this happens, be sure to connect back to the mainframe, type ‘‘`STOP`’’, and press ENTER several times (perhaps as many as 15 times) before taking any drastic steps like breaking the connection. ‘‘`STOP`’’ is a special escape mechanism for getting out of Kermit protocol mode quickly. Kermit-370 recognizes such a request in most situations where terminal I/O is not entirely frozen. Sometimes, apparent lock-ups are due to something as simple as incorrect parity settings in the micro Kermit, so always check the basic communication settings and, if necessary, experiment before trying a different `CONTROLLER` type. Also, to avoid unnecessary confusion, check for the existence of a Kermit initialization file (possibly created by the installer) which could be re-setting `CONTROLLER` after the automatic procedure has finished. Such a re-setting is a poor idea in an initialization file, even a personal one, unless there is absolutely only one kind of communications equipment on your system.

If file transfers do not work at first, it is best to do the following before trying again:

1. Reduce the packet size to no more than 80 at both ends.

2. Enable 8th-bit quoting at both ends.

3. Set `SEND PARITY MARK` in Kermit-370.

If those changes do not make transfers work, the next remedial action depends on the symptoms of failure.

- No packets exchanged and session locked up after reconnecting: change packet characters in both directions.

- No packets exchanged, but no lockup: change packet characters or parity.

- Always multiple retries of third or fourth packet: reduce receiving packet size.

- Multiple retries after random number of packets: check hardware and cables.

- Multiple retries after file-dependent number of packets: check for equipment that intercepts one or more printable characters or reduce the packet size.

When all else fails, you should be ready to reset `CONTROLLER` by hand and try again. The change most likely to be necessary is from `GRAPHICS` to `SERIES1`. However, it is conceivable that the installer has modified Kermit-370 at your location to force the `CONTROLLER` setting from the start, in which case, you might need to go the other way. Normally, Kermit's diagnostic procedure at start-up takes one or two seconds (because of programmed delays), so you should be suspicious if the Kermit prompt appears immediately after you start the program. The only initial `CONTROLLER` setting that you should *not* change by hand is `NONE`, which means that Kermit-370 has recognized a 3174 AEA line that is not configured for file transfer (or else the Kermit installer has a warped sense of humor).

If no amount of experimenting gets a transparent mode to work, it is time to recheck the list of supported devices and the age of your equipment. If yours is very old, it may require new microcode or some other software or hardware upgrade. In any case, if your results (whether positive or negative) are not already shown in `ik0aaa.hlp`, you should report them to Columbia so that others may profit by your experience.

1.7. 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 1-4.

The error codes in Table 1-4 bear no relationship to the severity of the associated error conditions, aside from the assignment of code 0. The underlying rationale is that the only current generic system for the treatment of completion codes is to take a non-zero code as an indication of error. Indeed, Kermit returns a completion code of 0 when "error" condition 1 holds.

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
21	PTY	8th-bit quote not set
22	FTS	File too short
23	SOH	Missing start-of-packet
24	OPT	Option error on filespec
25	DSP	Unable to dispose of file

Table 1-4: Error messages and codes for Kermit-370

1.8. What’s New

Below is a list of the changes in Version 4.3 of Kermit-370.

1. Compatibility with the (aging) F-level assembler.
2. Support for LATIN2, LATIN3, TRANSPARENT, CP870, CP905, and CP880, as well as the aliases L1, L2, and L3. New alias CP1047 for EBCDIC.
3. Support for IBM 3174 ASCII Graphics mode.
4. Improved controller detection, including local customization options.
5. Support for new unprefixed transmission of selected control characters.
6. Support for REMOTE PRINT, REMOTE MAIL, and REMOTE SUBMIT.
7. Improved error message for bad packet-size, new alias PACKET-LENGTH for PACKET-SIZE.
8. Correct observance of FILE COLLISION for all files in a group.
9. Ignoring spurious flow-control "packets" from (for example) MS-Kermit.
10. Support for new locking-shift Kermit protocol.
11. Support for Japanese Kanji file transfer and support for the Thai and Arabic character sets.
12. New versions of Kermit with the interactive messages in languages other than English.

13. New FULLSCREEN controller type.
14. New HINTS subcommand.
15. More graceful recovery from terminal I/O errors and exceptions.
16. Support for SNA LU1 3770-type devices and 8-bit, no-parity devices.
17. Optional conversion of EBCDIC printer carriage control into ASCII control characters.
18. Support for STOP command on "dumb" 3270 terminals and PCI protocol converters.
19. New efficiency display in STATUS report, based on SPEED setting.
20. New, uniform messages upon entering protocol mode, in the form "KERMIT READY TO SEND..." (or RECEIVE or SERVE).
21. 8-bit XECHO output.
22. Control prefixing for C1 controls.
23. New VERSION subcommand.
24. Improved debugging facilities.

1.9. What's Missing

Work on Kermit-370 will continue. Features that need to be improved or added include:

- Implement file archiving.
- Implement file transfer checkpointing.
- Add SET REPEAT subcommand.
- Improve Kermit-370 operation as a local Kermit.
- Implement public server mode.
- Allow REMOTE KERMIT HELP, REMOTE KERMIT DIR, and REMOTE SET from a micro.
- Add new SET FILE LONGLINE DISCARD option to allow multi-file transfer to proceed past a truncation problem.
- System-specific upgrades; see the respective chapters for details.

Anyone interested in working on these or other improvements should first get in touch with the Center for Computing Activities at Columbia University to find out if someone else has already begun a similar project (and, if so, who).

1.10. Further Reading

Below is a list of references for some of the material in this chapter.

1. *About Type: IBM's Technical Reference for 240-Pel Digitized Type*, S544-3156-02 (1989). This manual contains visual tables of many EBCDIC code pages and a comprehensive list of character names and acronyms.
2. *IBM System/370 Reference Summary*, GX20-1850-3 (1976). This reference card contains EBCDIC and ASCII character codes.
3. *IBM VS Fortran Application Programming: Language Reference*, GC26-3986-1 (1982). Appendix E contains a table of EBCDIC and ASCII characters with an implied full 256-byte translation table.
4. *IBM 3174 Character Set Reference*, GA27-3831-02 (1990). Chapter 5 contains visual tables of many

EBCDIC code pages.

5. *Info-Kermit Digest* Vol. 11 #1 (1989). This issue contains draft specifications of some Kermit protocol extensions.
6. *ISO International Register of Coded Character Sets to be used with Escape Sequences* (1989). This (very large) document has complete and unambiguous descriptions of standard coded character sets. It can be obtained from the ECMA.
7. *Kermit, A File Transfer Protocol* by Frank da Cruz; Digital Press (1987). This book contains a thorough description of the Kermit protocol and services with copious examples.
8. Kermit distribution file ISOK7.TXT (1992). This preliminary draft describes the new transfer protocol, including the international character-set support.
9. *Kermit News* Vol. 3 #1, p.5, "Dynamic Packet Size Control" (1988). This article describes an algorithm for optimizing Kermit throughput in the face of line noise.
10. *Kermit News* #4, p.16, "International Character Sets" (1990). This article discusses the new transfer protocol.
11. *ASCII and EBCDIC Character Set and Code Issues in Systems Applications Architecture*, SHARE white paper by Edwin Hart (1989). This document, available as file SHARE REQUIRE from LISTSERV@JHUVM, lays out general considerations for character codes and translatability.
12. Kermit distribution file LSHIFT.TXT (1991). This file describes the new protocol extension for better compression of 8th-bit text on 7-bit channels.

2. IBM CMS KERMIT

Program:	John Chandler (Harvard/Smithsonian Center for Astrophysics); contributions from Vaçe Kundakçi and Daphne Tzoar (Columbia U), Bob Shields (U. Maryland), Greg Small (UC Berkeley), Clark Frazier (Harvard Bus. Sch.), Bob Bolch and Steve Blankinship (Triangle), Ron Rusnak (U. Chicago), André Pirard (U. Liège)
Language:	IBM/370 Assembler
Documentation:	John Chandler (CfA)
Version:	4.3.0 (93/9/30)
Date:	1993 September

Kermit-CMS Capabilities At A Glance:

Local operation:	Yes
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
Packet logging:	Yes
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

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`
Mail command:	`EXEC KERMAIL` *filespec* `(` *users*
Print command:	`EXEC KERMPRT` *filespec* `(` *options*
Submit command:	`EXEC KERMSUB` *filespec* `(` *options*
Maximum packet size:	1913 (SERIES1), 2030 (TTY)
Maximum disk LRECL:	65535

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 variant of Kermit-370 that runs under the CMS operating system. The primary documentation for Kermit-CMS is actually the chapter on Kermit-370 (entitled IBM 370 Kermit), 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.

2.1. The 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 CMS *filespec* takes the form

```
filename filetype filemode
```

(often abbreviated FN FT FM). The filename and filetype are one to eight characters each. The filename 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, CMS requires a type for each file. Therefore, Kermit-CMS supplies a default type of ‘‘`$`’’ for any received file if no type is provided by the remote system. The same default is used for a missing filename. At the same time, Kermit forces the FN and FT to conform to CMS rules in other respects. The FN and FT may contain, in any order, uppercase letters, digits, and the special characters ‘‘`$`’’ (dollar sign), ‘‘`#`’’ (number sign), ‘‘`@`’’ (at sign), ‘‘`+`’’ (plus), ‘‘`-`’’ (hyphen), ‘‘`:`’’ (colon), and ‘‘`_`’’ (underscore). Any other character, if found in the FN or FT, 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 in all but the filemode number. Even the filemode letter is not a fixed part of the *filespec* because the same mini-disk or Shared File System (SFS) directory could be accessed under a different mode letter. In some ways, the filemode letter is like a disk directory designator, even when it refers to a mini-disk, 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. However, extra information may be added by way of the SET FOREIGN subcommand.

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 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. An important point to note is that records being written to a RECFM V file are not limited in length by the LRECL, but only by the CMS maximum (65535 bytes).

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).

2.2. 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 CMS is unable to interrupt a read on its "console". This means that the CMS variant 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, LINEDIT is set OFF to ensure that all characters are taken literally, 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, 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.

If, at some installation, Kermit can be run only on ‘‘`TTY`’’ lines, users will often be forced to disconnect or log off ongoing sessions on fullscreen lines. In general, users may operate through IBM 3270-type terminals and then disconnect in order to reconnect to a line that supports Kermit. In cases like this, users should be warned that reconnecting to a session over a ‘‘`TTY`’’ line is different from logging on initially over such a line. In particular, the CMS SET parameters AUTOREAD and BLIP and the CP TERMINAL parameter LINESIZE may need to be reset after reconnecting. AUTOREAD should be ON for ‘‘`TTY`’’ lines and OFF for fullscreen lines; the other two parameters are a matter of taste. A similar warning applies to reconnecting in the opposite direction.

CMS is different from some other IBM mainframe systems in that it 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`.

Kermit under VM/XA and beyond

Recent evolution of IBM's VM operating system has been marked by sharp discontinuities caused by, among other things, the transition to Extended Architecture (XA) mode. As a result, there are now two CMS variants of Kermit-370, one for the traditional systems and one for VM/XA. The former is now XA-cognizant but not, in IBM's terminology, XA-tolerant. In other words, it will run only in 370 mode or under VM/SP. The latter variant is fully bimodal and will run in 370, XA, or ESA mode under a bimodal (5.5 or 7 or higher) CMS, but will not assemble or run under pre-5.5 releases of CMS. The differences between the two variants are essentially invisible to the user, however, aside from the announcement of the release number when Kermit starts up. Both variants carefully determine whether they are running under VM/XA and, if so, avoid setting the CP parameters ACNT and TIMER, which VM/XA SP 2 does not support. Also, the traditional variant halts gracefully if it finds itself running in XA mode.

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 4.3.0 (93/9/30)
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 type binary # server
```

Kermit will exit and return to CMS after completing the specified subcommand or subcommands. Several subcommands may be given on the command line as long as they are separated by the LINEND character, which is number sign in this case. Note that the LINEND delimiter is a function of CP, rather than Kermit, and applies only to commands entered from the terminal and only when LINEDIT is on or when talking to CP itself. 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. 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. However, some CMS system or user 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. However, some system configurations, especially those including VTAM, are incapable of resuming Kermit protocol transmission after interruption by direct console I/O, so such commands should generally be avoided.

2.3. Kermit-CMS Subcommands

Kermit-CMS supports all the subcommands described in 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 subcommand ‘‘`HOST`’’. 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.

This section concentrates on the subcommands that have special form or meaning for Kermit-CMS. These are ordered alphabetically. See the chapter on Kermit-370 for further details.

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.

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 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. If this subcommand is issued without any arguments at all, Kermit-CMS will prompt the user for both foreign and native *filespecs*.

The GIVE Subcommand

Syntax: `GIVE` *table-name* *filespec*

This subcommand compares the named translation 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 HELP Subcommand

Syntax: `HELP` *subcom*

This subcommand displays the relevant part of the Kermit help file when the latter is a partitioned data set. Kermit verifies that the argument is a valid Kermit subcommand (or a non-ambiguous abbreviation) and then displays the corresponding member of the PDS. For the SET subcommand, individual parameters may also be specified, as in

```
HELP SET BLOCK-CHECK
```

If no subcommand name is included, Kermit displays the member that gives an overview of Kermit operation. Kermit looks for a help file with a filename matching the Kermit command itself, but will settle for one with a filename of KERMIT if necessary. Thus, the response to the HELP subcommand may depend upon which Kermit module is invoked. The choice of whether to format the help file as a PDS is an option at installation time. If the installer has chosen to leave it as an ordinary sequential file, or if no help file has been installed recently, Kermit will fall back on the old behavior of the HELP subcommand: it will issue the CMS HELP command for Kermit and therefore display the entire file. Since the help file is rather long, this subcommand is not recommended for users with line-mode terminals, unless the help file is installed as a PDS. See the installation guide for more details about the help files.

The RECEIVE Subcommand

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

The RECEIVE subcommand tells Kermit to receive a file or file group from the other system. You must issue the corresponding 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.

For purposes of truncation and folding, the maximum record length for a received file is 65535 if RECFM is V and "LRECL" if RECFM is F.

If the incoming file has the same name as an existing file, the action taken depends on the FILE COLLISION setting. The possible settings and their meanings are given in the Kermit-370 chapter. Two of the settings (BACKUP and RENAME) require that Kermit-CMS change the incoming name so as not to obliterate a 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 SEND Subcommand

Syntax: `SEND [`*filespec*`[<`*options*`>] [`*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]
```

or

```
fn.ft.[fm]
```

but the filemode is optional only if the *foreign-filespec* is omitted. The "dotted" notation is interpreted by changing up to two dots into spaces, so only the normal CMS-style form is "real" as far as Kermit is concerned. For details on the *options*, see the description of SEND in the Kermit-370 chapter. Note that no blanks may intervene between the CMS *filespec* and the *options*, even though the *filespec* itself has imbedded blanks (either explicitly or implicitly through the "dotted" notation).

The *filespec* may contain the wildcard characters ‘‘`*`’’ or ‘‘`%`’’. If it does, 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. If both *filespecs* are omitted for this subcommand, Kermit will prompt separately for each, and the respective syntaxes are exactly as described above, except the filemode is optional even if a *foreign-filespec* is to be supplied. This prompting mode is especially useful when more than one file (or file group) is to be sent, since the command line is limited to 130 characters.

Trailing blanks in a text file with RECFM F are deemed superfluous and are stripped off when Kermit-CMS downloads the file. In order to treat such blanks as significant, you must convert the record format to V, for example, by using COPYFILE with the "RECFM V" option. Note: you must not use XEDIT for such a conversion, since it also strips trailing blanks from files with RECFM V.

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, folded, or truncated, as needed, to the current LRECL.

SET PROMPT

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

This sets the prompt string, just as in other variants of Kermit-370, except that the string is padded with the current HANDSHAKE character, if any, unless the string is empty or already ends with that character.

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 via the other Kermit), Kermit-CMS will search all accessed disks for the named file or files. The search follows the usual search order.

OFF If the filemode is not specified, only the "home" disk and its read-only extensions will be searched for matching files. (Default.)

The SPACE Subcommand

Syntax: `SPACE [`*letter*`]`

This subcommand displays the storage allocation on the specified CMS mini-disk or SFS directory. 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 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 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 Returning from Kermit-370" in the Kermit-370 chapter). An EXEC could therefore be set up to react appropriately to file transmission errors or other unpredictable events. A third difference is that, although an EXEC may issue a TAKE subcommand, the latter will not be executed until after the EXEC processor returns to Kermit.

2.4. How to build an executable Kermit-CMS

Before attempting to build Kermit-CMS, look in the Kermit distribution under 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. In fact, IKCKER INS contains an EXEC for installing Kermit nearly automatically.

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. KERMBOOT can be useful under CMS/SP Release 3 and below, but is not needed under Release 4 and cannot be used at all under Release 5 and above. 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 (the hard way):

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, IK0COM, IK0CMD, (optional: IK0KAN), 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 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 and VM/XA SP 2, these are DMSSP, CMSLIB, TSOMAC, and OSMACRO. Under VM/XA SP 2.1, they are DMSOM, DMSGPI, and OSMACRO.
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" as a nucleus extension for invocation at need. Note: the nucleus extension can be removed by the command ‘‘`NUCXDROP KERMIT`’’.

To create a runnable version the easy way, extract IKCINS EXEC from IKCKER INS and run it. When it is finished, you may perform any desired tests and then move the MODULE and HELP files to their permanent locations. It is useful for the Kermit module and help file to have the same filename and reside on the same mini-disk.

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.

2.5. What's New

Below is a list of the CMS-specific features in Version 4.3.0 of Kermit-CMS added since the previous major release, Version 4.2, in March of 1990. For the list of generic additions, see the chapter on Kermit-370.

1. New, automated installation procedure collected as an EXEC contained in the installation guide. This includes selection of XA support and SFS support as appropriate.
2. Help for individual Kermit subcommands from the HELP subcommand.
3. Improved recovery from I/O errors during file transfer.
4. Improved execution of CMS commands under Kermit with SYSCMD ON.
5. File names no longer considered to "collide" with files existing on read-only extensions.
6. Support for disk space check in advance of receiving a file into SFS and more accurate space check for ordinary mini-disks.
7. Improved SFS wildcard sending.
8. Small bug fixes.

2.6. 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.
- Implement file archiving.
- Add a SET REPEAT subcommand.
- Implement public server mode, allowing a disconnected virtual machine to provide Kermit services via CP DIAL.
- Add a CONNECT subcommand. This may be impossible.
- 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.
- Set file date/time on received files from the information sent by the other Kermit.

Anyone interested in working on these or other improvements should first get in touch with the Center for Computing Activities at Columbia University to find out if someone else has already begun a similar project (and, if so, who).

Index