Kermit-20 Version 5.3 Announcement

Thomas DeBellis, SLOGIN@TOMMYT.#DECnet (HECnet)

Summary

Version 5.3 is the first major release of Kermit-20 in two decades and represents approximately a year of development for the following functionality:

- Updates Kermit to version 7 of Tops-20.
- Adds DECnet NRT transport to Tops-10, Tops-20 and Ultrix hosts.
- Adds pseudo-terminal support.
- Support for batch stream execution, including extensive testing.
- Microsecond timing (limited by Tops-20)
- Efficiency Enhancements
- Symbolic (C) escape sequences and enhanced parsing
- New and Enhanced commands
- Revamped and enhanced macro facility

A large amount of restructuring has been done with an eye towards future development, instrumentation, debugging and overall robustness. The following is a brief table of changes.

Table of Contents

Restructuring

At 201 pages of source code, Kermit-20 exceeded the capability of the Tops-20 assembler (MACRO-20), resulting in MCRNEC errors, effectively preventing any possibility of further development. As a result, the source code was split up and the memory layout was changed and enhanced.

Despite the large amount of changes, this was in no way a complete rewrite of the original source as it nor largely exists in separate modules, typically having been enhanced. Apart from the new functionality, most changes happened because the increased demands being put on the code caused 'edge case' errors to be discovered. In other words, effectivity supporting megabaud network transport sometimes turned out to be qualitatively different than kilobaud transport.

Source Code

The following are the new source modules. Pages are Tops-20 and consist 512 of words, a word being thirty-six bits. Characters are packed 5 characters per word and are different from eight bit bytes. This totals to approximately 1.4 megabytes of source code which represents a 366% increase in size.

Item	Name	Pages	Characters	Summary
1	K20DSP	37	93,795	Display module; show commands
2	K20EDT	21	51,343	Edit history
3	K20HLP	57	145,533	Built-in help
4	K20IOC	57	145,758	INPUT/OUTPUT/TRANSMIT commands
5	K20MAC	31	77,731	Kermit Macros
6	K20MIT	76	194,250	Protocol handling
7	K20NET	57	144,186	DECnet, physical and pseudo-terminals
8	K20PAR	64	162,959	Parsing and some execution
9	K20PDC	11	27,080	Packet display and decoding
10	K20SRV	57	144,035	Server commands and support
11	K20SUB	53	133,960	Miscellaneous subroutines
12	K20TIM	41	103,131	Microsecond timing
13	K20UNV	18	44,553	INPUT/OUTPUT/TRANSMIT commands

Memory Layout

Kermit-20 was one of the first two programs to implement the KERMIT protocol, which was done under version 5 of Tops-20. This limited the address space to a single section of 512 pages. In Tops-20 version 7, extended memory support was completed, resulting in an increase to 32 sections of 512 pages.

This increased virtual memory was used to move certain items out of section zero, in order to increase cache performance, make buffer management easier, prevent symbol table overwrite and to implement guard pages to catch off by one errors. The memory was also write-protected to catch straw stores.

Using MACRO program sections (.PSECT's) and Program Data Vectors (PDV's), section one was created to contain the symbol table, all help text and most program static text, such as all keywords, switches and guide words. It is very difficult for Kermit, which, with a single small exception, runs exclusively in section zero, to smash these.

Batch Enhancements

Previous versions of Kermit-20 were not Batch compliant as they did not properly handle the lack of Control-C capability, use compliant error signaling (I.E., ESOUT%) and modified various terminals in a manner incompatible with Batch, even when the local terminal was not being used for transfer.

Regression and Functional Testing

Remediation allowed extensive testing scripts to be run. While the Tops-20 Batch control language is not sufficiently powerful to replicate the functionality found in C-Kermit scripting, it is more than strong enough to write functional and regression testing scripts. High level explanations of these control files may be found in KERMIT-20-TESTING-BATTERY.docx and are summarized as follows:

1. K20NUL: Basic Kermit-20 Server functional checkout
2. K20NUP: Basic Kermit-20 Server functional checking using PUT
3. K20PTY: Kermit-20 Server Transfer additional and post transfer checks
4. K20NRV: Kermit-20 Server functions via Tops-20 DECnet NRT to local host
5. K20NRT: Kermit-20 Server functions via Tops-20 DECnet NRT to remote host
6. K20NOL: Basic regression test against Kermit-20 4.2(174) [2-May-85]
7. K20POL: Regression test against Kermit-20 4.2(174) [2-May-85] using GET
8. K20PNW: Regression test against Kermit-20 4.2(174) [2-May-85] using PUT
9. K20NUR: Basic Kermit-20 Server functional checkout with parity
10. K20POR: Regression test against Kermit-20 4.2(174) [2-May-85] with parity
11. K20B8T: Kermit-20 Server 8-bit file transfer and post transfer checks
12. K20B8P: Kermit-20 Server 8-bit file transfer with parity and post transfer checks
13. K20B8A: Kermit-20 Server terminal 8-bit file transfer with parity and transfer checks
14. K20DPD: Kermit-20 Packet Decoding Example
15. K20TCP: Kermit-20 Transmit/Capture Testing via pseudo-terminal
16. K20TCN: Kermit-20 Transmit/Capture Testing via DECnet NRT
17. K10NRT: Kermit-10 Regression Tests via DECnet NRT
18. K10NRP: Kermit-10 Parity Regression Tests via DECnet NRT

Extensive regression testing was done against a version of Kermit-20 from 1985 as well as Kermit-20 for Tops-10 (see items 17 and 18).

C-Kermit was tested separately using Mac OSX; remarkably, while there are a very large number of C-Kermit scripts, none of them appear to be for basic checkout and regression testing.

Testing Limitations

The testing control and resulting log files are provided for reference purposes. As these are user specific, it should not be assumed that they will execute correctly without local modifications. Such self-configuration is not possible using the Tops-20 Batch control language.

There are no control files for C-Kermit as no Ultrix host was available for testing on HECnet at the time of writing.

It was not possible to test Kermit-20 5.3 on an actual KL10 based DECSYSTEM-20 as these are largely museum pieces which would have involved travel to use.

Kermit-20 5.3 does not run on KS based systems and thus could not be tested against any, as these are limited to Tops-20 4.2. It is possible to upgrade Tops-20 to version 5 on a KS10, but these sources are not available. Were this to be done, then Kermit-20 5.3 could probably be backported to support this.

Kermit-20 5.3 was not tested against any Systems Concepts hardware as none was available for testing on HECnet.

Kermit-20 5.3 was not testing against any XKL system as none was available and the version of Tops-20 which runs on XKL systems does not support DECnet.

Kermit-20 5.3 does not run on TENEX nor has any version of Kermit-20 ever run on TENEX.

Enhanced Debugging Logging

Previously, Kermit-20 logging was limited to the second and displayed the actual characters going over the wire. Unfortunately, at megabaud and higher speeds, a second is not sufficiently granular and a number of messages can happen in that time.

Further, unless one is intimately familiar with the Kermit protocol itself, puzzling out what the characters actually mean can be problematic. Logging was therefore enhanced to provide time of day to the millisecond and a breakout of what is being done, viz:

```
S,31-Mar-2023 19:30:58.411, type: G, seq: 0, len: 4, Blk: 0, Generic, Print Working Directory
R,31-Mar-2023 19:30:58.413, type: Y, seq: 0, len: 4, Blk: 47, Ack(GA), STAR:<KERMIT.K20MIT>
S,31-Mar-2023 19:30:58.422, type: G, seq: 0, len: 4, Blk: 47, Generic, Server Status Query
R,31-Mar-2023 19:30:58.425, type: X, seq: 0, len: 4, Blk: 29, Text: (null)
S,31-Mar-2023 19:30:58.425, type: Y, seq: 0, len: 3, Blk: 29, Acknowledged packet type "X"
R,31-Mar-2023 19:30:58.427, type: D, seq: 1, len: 3, Blk: 33, Data: #M#J Maximum number of
   characters in packet:  94 received: 94 se
```

Packet decoding can be invoked by SET DEBUG PACKETS /DECODE. This is particularly useful when trying to understand negotiations, such as:

```
R,31-Mar-2023 19:30:58.753, type: I, seq: 0, len: 3, Blk: 32, Initialization
    Params: 10, MaxL: 94, TimO: 18, Npad: 0, PadC: ^@, EOL: ^M, Qctl: #, Qbin: Yes, ChkT: 6-
   bit, Rept: ~, Long: Available
S,31-Mar-2023 19:30:58.753, type: Y, seq: 0, len: 13, Blk: 32, Initialization Acknowledgement
    Params: 10, MaxL: 94, TimO: 20, Npad: 0, PadC: ^@, EOL: ^M, Qctl: #, Qbin: Yes, ChkT: 6-
   bit, Rept: ~, Long: Available
R,31-Mar-2023 19:30:58.756, type: R, seq: 0, len: 13, Blk: 5, Receive: NUL:
S,31-Mar-2023 19:30:58.756, type: S, seq: 0, len: 13, Blk: 5, Send Initiation
```

```
    Params: 10, MaxL: 94, TimO: 18, Npad: 0, PadC: ^@, EOL: ^M, Qctl: #, Qbin: Yes, ChkT: 6-
   bit, Rept: ~, Long: Available
R,31-Mar-2023 19:30:58.758, type: Y, seq: 0, len: 13, Blk: 50, Send Initiation Acknowledgement
    Params: 10, MaxL: 94, TimO: 20, Npad: 0, PadC: ^@, EOL: ^M, Qctl: #, Qbin: Yes, ChkT: 6-
   bit, Rept: ~, Long: Available
S,31-Mar-2023 19:30:58.758, type: F, seq: 0, len: 7, Blk: 50, File: NUL:
R,31-Mar-2023 19:30:58.759, type: Y, seq: 1, len: 7, Blk: 31, Acknowledged packet type "F"
S,31-Mar-2023 19:30:58.759, type: Z, seq: 1, len: 3, Blk: 31, End of File
R,31-Mar-2023 19:30:58.761, type: Y, seq: 2, len: 3, Blk: 32, Acknowledged packet type "Z"
S,31-Mar-2023 19:30:58.761, type: B, seq: 2, len: 3, Blk: 32, End of Transmission
R,31-Mar-2023 19:30:58.762, type: Y, seq: 3, len: 3, Blk: 33, Acknowledged packet type "B"
```

The above are the complete negotiations for Kermit-20 in server mode being asked to fetch the file NUL:

Time of Day Limitations

A careful inspection of the above reveals that not even millisecond resolution is sufficient to distinguish certain messages. Unfortunately, this is the maximum that Tops-20 can provide for time of day. While Tops-20 can provide a 100 kHz interface for timing intervals down to 10 microseconds (via the HPTIM% .HPELP function), this is decoupled from the time of day (I.E., GTAD%) and artificially wrapped every 76:21:17.90694.

The latter wrap allows the 1 mHz KL10 clock to simulate both the 100 kHz DK10 clock used on TENEX KA10 hardware and the TENEX interface. It effectively prevents high precision timing after the system has been up over 3 Days, 4 Hours, 21 Minutes, 17 Seconds, 906 Milliseconds and 940 Microseconds.

Fixing the problem would involve modifying Tops-20 to expose the complete raw clock (via the RDTIME instruction), the last time and day set and the uptime when it was set. While it would be possible to use XPEEK% and USRIO% to get the first two items, Tops-20 does not record the 3rd, so it is impossible for Kermit to completely resolve the problem by itself.

Kermit-20 is able to do the following via an understanding of some Tops-20 monitor internals. The time of day is kept in a 36 bit full word with the left half word being the number of integral days since November 17th, 1858 and the right half word being a fixed point fraction of the day, the denominator thus being 2^{18} or 262,144.

It is thus possible to time events from before the American Civil War until August 7th, 2576 (or some 717 years, 8 months and 21 days) to a resolution of 329.5898438 milliseconds. Since the time display of ODTIM% shows only fractions, Kermit-20 attempts to provide millisecond timing by keeping all times in milliseconds and calculating time of day off of the system boot date and time.

This has the advantage of providing times which do not go backwards, but microsecond time of day is still unavailable. All other times in Kermit at in HPTIM% units and are unrelated to time of day.

SERVER Enhancements

Server mode has been enhanced to provide some additional information.

PWD (Generic 'A')

Prints the current working directory, viz:

```
Kermit-20>Remote PWD
STAR:<KERMIT.K20MIT>
```

This allows better reverse compatibility with other versions of Kermit which can request this, such as C-Kermit.

STATUS (Generic 'Q')

Generic 'Q' is not adequately documented in the Kermit protocol, but the behavior is modeled after Kermit-10. It is possible that future versions of C-Kermit will implement the query. The format of the result is exactly what would be seen in Kermit-20's `LOCAL STATISTICS` command and is used to check for inconsistencies during regression tests.

This format is described below in the STATISTICS (*Enhanced*) command.

The NUL: device

Previous versions of Kermit-20 would only do transfers to file structures. Due to the amount of testing being done and to test extreme edge cases, the restriction was relaxed to allow transfers to the `NUL:` device, which has the effect of simply throwing away the data.

This also has the advantage of being able to time the raw protocol without storage latency getting in the way. Such extreme testing caught an edge case of zero length source file producing a target file with a byte count of zero yet a page count of one.

Most NUL: transfers are simulated, such as `DELETE` (which does nothing) and `DIRECTORY`(which always shows the same thing), viz:

```
Kermit-20>Remote Type "NUL:"
.
NUL:


[OK]
```

```
Kermit-20>Remote Delete "NUL:"



NUL: [OK]
 1 file
[OK]
Kermit-20>Remote Directory "NUL:"
.

NUL:
NUL:                    0          0(7)        Now
 1 file
[OK]
```

It will be noted that remote file names may now be specified in double quotation marks. This allows an easier specification of files with embedded spaces as well as facilitating escape recognition for end of line confirmation.

Since connecting to NUL: makes no sense under Tops-20, this is not simulated.

Efficiency Enhancements

Kermit-20 was largely coded for protocol correctness and robust error recovery. However, in certain instances, multiple instructions and JSYI were used where this could have been avoided. A number of these cases have been re-coded in order to reduce execution time requirements, typically trading space for time.

For example, EXTEND MOVST is used extensively for INPUT recognition and parity generation, resulting in a single instruction at the cost of a table, typically being less than a hundred words. In addition to generating parity, Kermit-20 will now detect parity errors, but this is only used for regression testing as the protocol itself provides these services and network connections are (claimed to be) error free.

C Escape Sequences

Previously, Kermit-20 would only allow special characters to be specified using an octal format. As these can be difficult to remember, the following ANSI C escape sequences are now recognized by Kermit-20:

Escape Sequence	ASCII Value Octal	Description
\a	7	Alert (Beep, Bell)
\b	10	Backspace
\t	11	Horizontal Tab
\n	12	Line Feed (Unix Newline)
\v	13	Vertical Tab

\f	14	Form feed (Page Break)
\r	5	Carriage Return (bare, no line feed)
\"	42	Double quotation mark
\'	47	Apostrophe or single quotation mark
\?	77	Question mark (used to avoid trigraphs)
\\	134	Backslash
\e	33	Escape character
\q	42	Double quote (for use inside quoted strings)
\c	3	ETX (Control-C, useful for DEC OS's)
\d	4	EOT (Control-D, Unix End of File)
\z	32	SUB (Control-Z, Tops-10/20 End of File)
\@	100	Tops-20 default EXEC command prompt
\.	56	Tops-10 monitor prompt

At-sign is probably best managed inside a quoted string. As an example, searching for "foo" followed by a Tops-20 newline and EXEC prompt might be written as "foo\r\n@".

Note the following:

1. All octal values are range checked to US ASCII seven bits and an error is thrown if this limit is exceeded. An IBM system is expected to be front-ended with the appropriate converter to handle explicit EBCDIC values.
2. Digits 8 and 9 will fail an octal conversion. All other characters will merely silently stop the conversion.
3. In all cases, Kermit will not process more than eleven octal digits after a backslash.
4. Newline on Tops-20, Tops-10 and other DEC operating systems is unlike Windows, Unix or Multics, where it is the single character "\n". The proper sequence to search for is "\r\n", in that order.
5. Unlike C or its derivatives, escape characters are *not* case sensitive, meaning (capital) \N will produce the same value as lowercase \n (octal 12). This is in keeping with all parsing under Tops-20 and allows for a user to provide emphasis.
6. The last seven items (\e, \q, \c, \d, \z, \@ and \.) are non-standard, the last five (\q, \c, \d, \z, \@ and \.) being unique to Kermit-20.
7. Any character that is not in the above table will cause an error.
8. Because of the string instructions being used, it is far more efficient to search for a single character. For example, on Tops-20, when searching for the result of an EXEC command, INPUT "\@" will execute with substantially less processor overhead then INPUT "\r\n\@".

Enhanced Escape Character Parsing

Where it is necessary to specify an ASCII control character (for example, when setting the ESCAPE or HANDSHAKE characters), Kermit-20 provides the user with some flexibility.

The basic problem with specifying a control character is that these are frequently used by operating systems and applications as a source of interrupts to signal specific conditions or requests. Therefore, the character may not be directly available interactively and difficult to insert into a KERMIT.INI file.

Kermit allows a control character to be specified in one of two ways, the first being to simply type the octal numeric value. For example, 015 might be typed to indicate a carriage return. However, unless you happen to have an eidetic or otherwise photographic memory, recalling these values can be challenging in the absence of a handy table (which can be its own challenge).

The easier way to do this with Kermit is to type the carat or "^" character (also known as the up arrow character) followed by another character. This is intended as a mnemonic aid.

For example, when typing a ^B (ASCII 002) normally, the "B" key is 'chorded' with the Control key. This means that the Control key is pressed and held down and while being held thusly, the "B" key is pressed and released and then the Control key is released. The resulting character (ASCII 003) is then generated and it is commonly displayed as "^B".

In order to specify this ^B (or Control-B) to Kermit, the user would simply type the carat character followed by the letter "B". Kermit is caseless in this regard, so either "^b" or "^B" will work. As with all native Tops-20 programs, escape and question mark may be used, viz:

```
Kermit-20>SET ESCAPE ^ ? ASCII control character one of the
following:

 A    B    C    D    E    F    G    H    I    J    K    L    M
 N    O    P    Q    R    S    T    U    V    W    X    Y    Z
  or to specify the NUL character, type "@"
  or to specify the ESCAPE character, type "["
  or to specify the file separator character, type "\"
  or to specify the group separator character, type "]"
  or to specify the record separator character, type "^"
  or to specify the unit separator character, type "_"
  or to specify the rubout or delete character, type "/"
```

The carat and printable character to control character mapping is as follows:

Control Character	Octal Value	Control Character	Octal Value	Control Characte	Octal Value
^@	0	^K	13	^V	26
^A	1	^L	14	^W	27
^B	2	^M	15	^X	30
^C	3	^N	16	^Y	31
^D	4	^O	17	^Z	32
^E	5	^P	20	^[	33
^F	6	^Q	21	^\	34
^G	7	^R	22	^]	35

^H	10	^S	23	^^	36
^I	11	^T	24	^_	37
^J	12	^U	25	^/	177

Note that the use of up-arrow forward slash is non-standard as the more typical way of representing rubout would be up-arrow question mark ("^?"). This is not possible with Kermit-20 as question mark gives the menu shown above. Forward slash is used as it is on the same key as question mark is, unshifted on many keyboards.

Enhanced Quote Characters

The Kermit protocol mandates that the quote character be in the ASCII printable range, meaning you could see it were you to print it. More specifically, the character can not be in the control range (0 to 37 octal), nor a space (octal 40) nor rubout (177).

Kermit-20 allows the quote character to be directly specified, provided it is one of the following:

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

Other grammatical characters can not be specified directly because they are either usurped by Tops-20 itself or Kermit already uses them for other purposes (such as macro definition). These may be specified either as a mnemonic keyword or as octal. The following table lists the keywords, the characters and the octal equivalents:

Keyword	Character	Octal
at-sign	@	100
comma	,	54
dash	-	55
exclamation-mark	!	41
question-mark	?	77
semicolon	;	73

Thus, as far as Kermit-20 is concerned, the following two SET commands do exactly the same thing, setting the start-of-packet character to a comma.

```
Kermit-20>set send quote comma
Kermit-20>set send quote 54
```

New Commands, Differences and Enhancements

CAPTURE (*New*)

```
CAPTURE [ [ /TIMEOUT n ] [ /EOF ^Z | ^D | $ | ^G ] ]
     filespec [ prompt | "prompt" ]
```

Intended for use in remote mode only. This *invisible* command is the remote counterpart to the local TRANSMIT command since it 'captures' the text file that TRANSMIT sends. It is used only for testing purposes and the help text for the TRANSMIT command should be used as a reference before attempting such a capture.

The prompt must match what TRANSMIT was told to recognize as CAPTURE will print this after each and every line of text received. A line of text is expected to be terminated by carriage return (\r) and CAPTURE will write this to the indicated file and a line feed (\n) appended as TRANSMIT assumes carriage returns and strips line feeds.

The prompt string may be enclosed in quotes and may include special characters by preceding their octal ASCII values with a backslash (e.g. \15 for carriage return) or C escape sequence (e.g., \r being converted into octal 15).

Only a single file may be captured, no wildcards are allowed in the filespec. The file must be created on a file structure to which you either have or have gained access. No other device is permitted with the exception of NUL:, which is only used for debugging and testing purposes. It is *strongly* recommended that this file *not* be particularly large (see TRANSMIT).

The file will be captured using the current settings for duplex, parity, and flow control. These parity settings must be set beforehand and *must match* as a parity failure will cause the capture to be aborted.

Unless a SET INPUT TIMEOUT default has been set or a /TIMEOUT switch has been given there may be no timeouts on input text. Thus, like INPUT, CAPTURE command will wait forever for each line of text to appear. A control-C typed two or three times will close the file specified by CAPTURE command and return you to the Kermit-20> prompt.

The optional /EOF switch may be used to indicate that a particular character should be sent when the End-of-File (EOF) condition is detected. The following EOF characters are supported:

Character	Description
^Z	Used by DEC operating systems, such as Tops-10 and Tops-20, IBM/Microsoft OS's, such as DOS, OS/2 and Windows.
^D	Used by Unix type systems, such as Linux, Mac OSX and Ubuntu.
$	Dollar sign is used as a talisman for the escape character (Octal 33), which is used by Tops-10 and others as a line terminator.
^G	Used by certain DEC CUSPs, such as PIP and audible debugging.

The file is always received as an eight bit stream, even if it was seven bit ASCII originally and is always stored as seven bit ASCII, no matter what the source was.

Great caution should be exercised when using this command as it can crash communications systems, hang network connections and otherwise wedge other terminal based data paths. The safest approach may be to split the file up into several smaller pieces and do the TRANSMIT of each separated by PAUSE's at off hours in a batch job.

Recall that the whole point of the Kermit protocol was specifically to avoid this situation with DECSYSTEM-20 PDP-11 front end. Although these systems are only museum pieces now, their network implementations can still hang if pushed too hard.

If the wedge is bad enough, somebody may have to reboot the remote computer, which risks making you unpopular with the staff or owner if it happens often enough.

If you really want to send the entire file without looking for any prompts, specify a prompt of "\0" (ASCII zero or NUL). Again, this means that you are send the file at the fastest possible speed that the line will support with no acknowledgements; be aware that this will almost surely hang any computer which does not have a dedicated, real time input facility, particularly if this is done at Ethernet speeds.

Otherwise, this should only be used for testing and debugging purposes on loopback connections using pseudo-terminals as these will self-throttle.

CLEAR (*Enhanced*)

`CLEAR [ /ALWAYS | /DISPLAY | /ESTIMATE | /FLUSH | PEEK [/REPEAT [`*n*`]] ]`

Clear the monitor's input and output buffers of the currently selected line or network connection and remove any internally buffered data. For physical lines, attempt to clear any XOFF condition if XON/XOFF negotiation is being done.

A combined count of characters is returned if any were removed from either the monitor or internally.

This is highly useful in local mode, when the connection to the remote system appears to be stuck. Often, a stuck connection is the result of a rather large number of NAK's having piled up because:

1. The user hasn't issued a remote Kermit command in quite some time.
2. The remote server time-out parameter, which sets the NAK interval, having been set too low.

CLEAR can also be used for recovering from a TRANSMIT or INPUT error.

/ALWAYS

Whenever Kermit detects that it has cleared characters, it displays a summary like this:

```
Iterations: 2, Cleared: 19, Elapsed: 1.00012 (TOD: 3)
```

Using ALWAYS forces this summary to be displayed whether or not anything was actually cleared. It is not the default as this causes more output. The switch is provided largely for debugging and check-out purposes.

/DISPLAY

Show the data that was actually removed from the line or network connection and internal buffers, viz:

```
Kermit-20>CLEAR /DISPLAY
Buffered: 43, 'ENTI2, PANDA TOPS-20 Monitor 7.1(21746)-5
```

```
'
^A# N3^M
^A# N3^M
^A# N3^M
^A# N3^M
^A# N3^M
^A# N3^M
^A# N3^M
^A# N3^M
^A# N3^M

Characters flushed: 97
```

When connected to a virtual device, such as a DECnet NRT or a pseudo-terminal, a count of characters removed is kept. For DECnet, this number can get to be quite large and is the result of a Tops-20 monitor anomaly which Kermit-20 compensates for.

Despite the anomaly, be aware that a large number of NAK's can build up, perhaps because the connection has sat idle overnight, depending on the remote system's user and monitor buffers.

Although /FLUSH and /DISPLAY will exert best efforts and not return until the local Tops-20 hosts declares the line cleared, once the logjam is resolved locally, the remote host may still have more NAK's to deliver.

Therefore, do not assume that a single CLEAR will do the trick, multiple clears may need to be issued in order to fully drain the connection.

Like /FLUSH, /DISPLAY will toss any data in Kermit's internal buffer which is remaining for the INPUT command to process. This need only be done once; using /REPEAT to get rid of buffered internal data produces no detectable effect.

/ESTIMATE

Display what the monitor thinks is in the line or network buffer. Be aware that, unless you have a local or "loopback" connection to a Kermit on a Tops-20 pseudo-terminal (PTY), this number is often wrong because characters can be stacked up at the remote site, waiting for transmission.

In short, there is no way Tops-20 can know what a remote host might have in store for you. Therefore, the only truly accurate way to know how much data is stuck is to actually try remove it with either the /FLUSH or /DISPLAY switches.

To see what might be waiting for INPUT or TRANSMIT to process, do a /PEEK.

/FLUSH

Remote all data from the line or network connection. /FLUSH is the default; giving a CLEAR command with no switch implies /FLUSH.

In addition to getting rid of whatever is in the monitor buffers, /FLUSH will also remove what Kermit may have buffered internally for the INPUT or TRANSMIT command to read.

/FLUSH will report a count of characters it was able to remove, which will include the combined total of the monitor and Kermit internal buffers. Doing a SHOW LINE will show these totals separately.

/REPEAT [*n*]

The /REPEAT switch may be used to issue multiple CLEARs. By default, /REPEAT will try five clears, each separated by an interval of 1 second. A count parameter may be specified to change the repetition and an optional comma may be given, followed by the number of seconds to delay. For example:

```
CLEAR /DISPLAY /REPEAT:6,1.25
```

Will cause the CLEAR /DISPLAY logic to be repeated six times with a delay interval of one and a quarter seconds. Thus, the total time to execute the command will be seven and a half seconds.

Leaving a CLEAR /DISPLAY running for a long time is one way to determine whether you have a noisy line or whether any unexpected output might be hanging things up.

It is possible to repeat an /ESTIMATE, but the value is unlikely to change nor be of much value. The functionality exists for testing and debugging. Similarly, using /REPEAT to get rid of buffered internal data produces no detectable effect as the buffer will not contain any data until such time as another INPUT or TRANSMIT is issued.

A repeat loop may be aborted with a ^C. If the process does not have Control-C capability, then a Control-G may be used instead, and the user will be advised of this.

The status of a repeat loop may be requested with a Control-Y (^Y). It will print the number of iterations done so far, the total characters cleared and the elapsed time spent doing this, which includes the accumulated delays. For example:

```
Iterations: 17, Cleared: 120, Elapsed: 12:04.63581 (TOD: 2198)
```

/PEEK

Use of the TRANSMIT or OUTPUT and INPUT commands can cause Kermit to keep a certain amount of host output in an internal buffer. This buffer is exclusive to these commands and will not be seen by any data transfer commands (such as SEND and RECEIVE) nor will it be seen if you reconnect.

/PEEK will show the size of this internal buffer and what is in it, but does not remove anything to allow for further use of TRANSMIT, INPUT and OUTPUT. This is useful for debugging scripts (I.E., 'TAKE' files), macros and batch jobs.

Therefore, /ESTIMATE will not report this data, but /FLUSH will get rid of it and /DISPLAY will show it as part of removing it.

CWD (*Enhanced*)

```
CWD directory [ password | "password" ]
```

CWD means CHANGE WORKING DIRECTORY. It is the same as LOCAL CWD, which is executed on the local Tops-20 system. This operation is the same as the DEC-20 CONNECT command and, similarly, a password may be requested for proper completion. CWD is a Unix term; the operation here being more properly known as a "CONNECT".

Be aware that connecting to a Tops-20 directory has *very* different functionality than other operating systems and can have far-reaching consequences.

In particular, CWD can change access to other files as well as change the default directory in which file operations will be performed. This means that file operations which previously failed may now succeed and vice versa. CWD is commonly used to gain this access.

Also, a CWD changes the directory for the entire job and not just for the Kermit-20 fork. Concurrently executing forks will not be aware of this. Open files will not be effected, but any new file operation that does not explicitly specify or default the structure and directory will use the new 'working' (I.E., connected) directory.

Note that if the directory in question is on a regulated structure, then that structure must be mounted first with the DEC-20 MOUNT STRUCTURE command as Kermit-20 contains no facility to do this and is prevented from bypassing mount counts in most situations.

Attempting to specify a directory on a structure which has not been mounted will be rejected as a parse error. This allows the user to type a ^H to get the command back, ^C to the EXEC, mount the structure and then CONTINUE the Kermit-20 fork to confirm the CWD. The parse error will abort batch jobs.

Finally, it can readily be seen that makes no sense to connect to a device which does not support file structures and Kermit-20 will reject this. Trying to connect to NUL: does nothing and leaves you still connected to your currently connected directory.

Simply issuing CWD with no arguments will connect you to your login directory. If you are connected to a directory on one structure and wish to connect to a directory of the same name on another structure, you can merely type the structure name by itself and Kermit-20 will default the directory. If this fails for some reason, you will get a parse error and can retry the command.

The second form of CWD allows both the directory and the password to be specified on the same line. This is primarily intended for .CMD files, batch jobs and never for interactive use. This is because the password is echo'ed can thus be seen by others!

In this case, omit the password and Kermit-20 will only prompt for a password if it determines that it can't connect without one. When Kermit-20 prompts for a password, it will not be echo'ed.

In an attempt to decrease the needless use of passwords, Kermit-20 will reject a quoted password as a parse error in addition to not prompting for one. This will allow the user to type a ^H to get the command back and then type a ^W, rubout (or delete) and hit enter to retry the command. The parse error will abort batch jobs.

When issued as a REMOTE CWD, the directory specification may be put in double quotes.

CAUTION: Kermit-20 is not able overwrite the password text on half-duplex lines because it has no control over the line feed that is typed after the enter key is hit. It is suggested that the screen be promptly cleared or the paper removed.

DELETE (*Enhanced*)

DELETE *filespec*

DELETE is the same as LOCAL DELETE. It works like the DEC-20 DELETE command, except that if you SET EXPUNGE ON, then all files that you delete with Kermit-20, whether with interactive DELETE commands, or via REMOTE DELETE commands when Kermit-20 is running as a server, will also be automatically expunged.

Where files have multiple generations, setting Kermit-20 to run without automatic file expunging has been enhanced to result in *far* faster performance. If a delete results in multiple generations being removed, then the count of generations per file will be displayed. Otherwise, a single generation will not be remarked on. At the end, the total number files deleted will be displayed.

Be aware that Kermit-20 does not have an UNDELETE command. If EXPUNGE is off, then you must use the DEC-20 UNDELETE command to recover the file.

If the directory is EXPUNGED by another user (perhaps by a LOGOUT), then the deleted files will no longer exist and can not be undeleted. Similarly, any file that is deleted with EXPUNGE set ON can not be undeleted.

A file may only be deleted from a directory based device, such as a file structure. All other devices will be rejected with the exception of NUL:. The NUL: device may be deleted as many times as you like, but nothing useful happens. Such functionality is intended purely for debugging purposes, only.

When issued as a REMOTE DELETE, file recognition is not available, but the file specification may be enclosed in double quotes.

DIRECTORY (*Enhanced*)

DIRECTORY *filespec*

DIRECTORY is the same as LOCAL directory. It produces a listing of the specified files at your terminal. The default is to list all generations of all file types of all file names in the currently connected directory.

The DIRECTORY command can not be used to list files that have either been set invisible or are deleted. Use the DEC-20 DIRECTORY command to do this. Additionally, neither command can be used to list files that have been protected against listing or are in a directory to which you have no list access.

Kermit-20 can not list files on a device that has no directories and will reject such requests. Therefore, it makes no sense to try to list your or anyone else's terminal.

NUL: may be listed, but nothing particularly useful is happens, a single zero length file with a zero byte length named "NUL:" being displayed, the creation date always being "Now". Such functionality is intended purely for debugging purposes, only.

When issued as a REMOTE DIRECTORY, file recognition is not available, but the file specification may be enclosed in double quotes.

ECHO

```
ECHO [ "Quoted Text" | text-string ]
```

Echoes the given text string at your terminal. Useful in TAKE command files and with login scripts for reporting progress or telling you what to do. A quoted string may also be given, which may then be explicitly visually confirmed. Typing nothing (I.E., an ECHO all by itself) will echo a blank line.

ECHO will expand C escape sequences. For example, issuing this command:

```
ECHO "\r\n\qHello\q\r\n"
```

will result in:

```
"Hello"
```

being displayed on the terminal. This is useful for learning, prototyping and refamiliarization with the functionality.

ECHO will respect parity, but this isn't really useful for anything but debugging and internal checkout. In many cases, abject gibberish will be displayed, depending on the character font of the terminal or rendering device.

INPUT (*Enhanced*)

```
INPUT [/SILENT] [interval] [ "quoted text string" | text-string ]
```

INPUT is useful with OUTPUT and PAUSE for sending connect and login sequences to a remote host, e.g. over a dialout or network connection.

On the currently selected communication line, network or virtual connection, look for the given string for the specified time interval.

If no interval is specified, then wait for the default interval, which is 5 seconds unless changed by SET INPUT DEFAULT-TIMEOUT. An interval of 0 means no timeout (wait forever). A negative time will result in a parse error.

Typing only a carriage return and no parameters at all will cause the default interval to be used and the default search string to be used. If no search default has been given with a SET INPUT SEARCH-DEFAULT, then a default sequence of carriage return, line feed will be used.

Fractional seconds may be given for n, such as 1.25. These will be converted to the nearest millisecond value. Such short waits may be useful for communicating with extremely fast remote machines over very high speed lines.

They are also used for debugging and stress testing. Unless you are actively debugging or the machine is very lightly loaded, it is unwise to perform stress tests.

Characters coming in from the connection will be scanned for the search string, and when a match is found, the command will terminate successfully. If the string is not found within the given interval, the command will terminate unsuccessfully. While the INPUT command is active, all incoming characters will appear on your screen.

The search string may contain any printable characters. Control or other special characters may be included by preceding their octal ASCII value with a backslash, for instance foo\15 is "foo" followed by a carriage return, \100 is an atsign (@).

Another way of writing "foo\15" would be "foo\r" because certain common C language escape sequences, which are useful for their mnemonic value, are recognized.

Alphabetic case is ignored ("a" = "A") unless you have SET INPUT CASE OBSERVE. If no search string is given, then the INPUT command will keep operating until it times out.

As INPUT scans the character stream for a match, it displays the text on the terminal under the assumption that these are system responses that the user will want to see. This is essential for understanding what might have gone wrong on a failure.

However, this might not be desirable for commands which produce copious output, in a batch environment or for certain types of debugging. For this reason, the /SILENT switch may be used to discard this text and not print it. Caution should be used because any error messages will also be discarded when /SILENT is in effect.

As such, /SILENT processing can not be set as a default for the input command. It must be issued for each and every command.

If the INPUT command fails to find the requested string within the given interval, it will "fail"; if the INPUT command was issued from a command file (see TAKE), then the next command will be executed, unless you have SET INPUT TIMEOUT-ACTION to QUIT, in which case the command file will be terminated. An INPUT command can by interrupted by typing two ^C's or two ^G's if the process does not have Control-C capability.

A quoted string may prove useful for interactive prototyping or to allow Kermit to recognize an at-sign (@), the Tops-20 prompt.

OUTPUT (*Enhanced*)

```
OUTPUT [ "quoted text-string" | text-string ]
```

The given string is sent out the currently selected physical communication line, pseudo-terminal or network connection. A quoted string may prove useful for interactive prototyping or to allow Kermit to emit an at-sign (@) while retaining the ability to use indirect files.

Typing a simple carriage return will emit a bare ASCII carriage return (octal 15). Control characters may be included in \ooo format, that is, a backslash followed by the octal numeric representation of the ASCII value.

In addition, certain common C language escape sequences, which are useful for their mnemonic value are recognized. Thus, the \015 may be more easily written as \r.

PAUSE (*Enhanced*)

`PAUSE interval`

Pause for the given number of seconds. Useful with INPUT and OUTPUT.

There is no realistic maximum of seconds, other than what is imposed by the PDP-10's arithmetic and Tops-20's acceptance of the dismiss time. It is probably not useful to sleep for more than a few minutes.

The interval is assumed to be seconds, yet is parsed is a floating point number, which will be converted into milliseconds to allow for greater granularity of waits and debugging.

The smallest pause value is 0.001 or one millisecond. A negative value will be rejected.

TIME (*New*)

`TIME device-keyword | COPY | device: [/switches]`

Performs basic communications checks and timing tests on the specified device. These are not normally required as is not necessary to time a physical line because the baud rate can be gotten from the monitor.

TIME exists largely for network transport code validation and for the verification of certain internal self-checks. The speed results may be used as a basis for populating the device efficiency portion of the statistics command, but this will probably mean very little.

TIME is not necessary in any way to a regular user, having use only to a programmer. Because of this, the TIME command is invisible and does not show up as a keyword when a question mark list given to either the top-level or HELP command prompt.

A timed device keyword may be one of the following which also may be specified by the corresponding unpunctuated device name:

Synonym	Device	Punctuated	Remarks
data-sink	NUL	`NUL:`	Used as `.NULIO`
DECnet	DCN	`DCN:`	DECnet Client
	SRV	`SRV:`	DECnet Server
pipe	PIP	`PIP:`	Tops-20 pipe device
pseudo-terminal	PTY	`PTY0:`	Must specify device number

This allows for the device name to be specified with escape recognition without punctuation and for the unit number to not need to be specified.

COPY

`COPY *from-device to-device*`

Used to copy test results from one device to the other, which has the effect of causing the *to-device* to appear to have a different virtual speed than what was measured. This is used for algorithm checkout, edge case testing and providing abnormal inputs to trigger error recovery.

Transfer results checking is typically on a byte-by-byte comparison if parity is set to NONE. However, if parity checking is turned on, then the timing routines will use that parity to check results. This is for the purpose of validating the parity implementation--it is slower to check by parity.

For an idiomatic device type or explicitly specified device name, a switch may be one or more of the following: /BYTESIZE: /MODE: and /RECORD-LENGTH:.

CAUTION

1. Not all devices support all switch options
2. Not all devices support all switch values
3. On a pseudo-terminal, the post transfer check will ALWAYS fail for any byte size greater than 8, even if the device is opened in IMAGE mode. This the nature of the terminal driver.
4. Timing results can vary wildly, based on whatever else the system happens to be doing.
5. A test may time out on a heavily loaded system. It is probably anti-social to keep trying immediately.
6. The virtual speed is calculated by characters sent, the default being 8 bit characters. Since a character (or byte) on the PDP-10 can vary between 1 and 36 bits, the reported speed will change with different byte sizes, but the differences may be counter-intuitive, if not flat out meaningless.
7. An odd byte size may trigger other false negatives. "Odd" in this case does not mean that bit 35 is set, it means very uncommon or 'strange'; say a byte size of 13, for example.

/BYTESIZE

`/BYTESIZE: *decimal-number*`

Requests a decimal number between 1 and 36. Not all devices support all byte sizes. DECnet, for example, will only support byte sizes of 7, 8 and 36; eight being the most efficient.

The data used for the timing tests is formed internally from a sequence of thirty six bit words, each consisting of four 8 bit bytes with monotonically increasing values followed by a single four bit field with monotonically increasing value. The 8 bit fields will wrap after 377 octal while the 4 bit field wraps after 17 octal.

The default byte size used to calculate transfer speeds is 8 bit. Since speed is bytes sent divided by elapsed microseconds, a smaller byte size will result in a faster speed being calculated for a transfer that took the exact same wall time because the number of bytes has increased.

Thus, the displayed speed may first appear to be counter-intuitive, whereas it is actually likely to be devoid of any useful information.

It is possible to specify a pathological byte size which will cause the post transfer validation tests to fail and drastically change computed transfer times. Such results are either all false negatives or useful only for debugging.

For example, while DECnet will explicitly refuse an OPENF% that does not specify a byte size 7, 8 or 36, PIP: and PTY: will accept the parameter without complaints. However while transfers against PIP: will work, the exact same transfer against PTY: will be found to have errors. Strictly speaking, these are not errors but rather a result of the nature of the monitor's terminal driver (TTYSRV).

/MODE

`/MODE: (DUMP) | IMAGE | NORMAL | SMALL`

The keyword may be one of the following: IMAGE, NORMAL or SMALL. Be aware that not all devices support all modes. While Kermit will parse for all values, some of them will fail. In this case, a failure is defined behavior and not indicative of any true problem.

NORMAL means different things, depending on device context. For pseudo-terminals, it will cause the monitor to interpret certain characters as opposed to doing a straight passthrough. Otherwise, it means to use normal buffering and flushing.

SMALL mode is for the monitor to use smaller buffers and/or flush them more frequently, giving perhaps better interactive response. It is valid only for DECnet and pipes.

IMAGE mode is for the monitor to not process characters going through the device, which can result in less CPU usage. Only pseudo-terminals will accept IMAGE, which is set via the associated controlled terminal line.

DUMP is an invisible MODE keyword. No current Kermit transport device supports DUMP mode, however this selection can be specified and is used to test and debug device open failure recovery.

/RECORD-LENGTH

`/RECORD-LENGTH: `*`decimal number`*

This switch is valid only for the PIP: (pipe) device and cannot be specified for any other device.

Specify a decimal number between 1 and 511 for the size of records to write to the pipe device. A negative or zero record-length will be rejected.

Generally speaking, larger record sizes will result in faster transfer times.

NUL: device considerations:

Be aware that /MODE can not be specified for the NUL: device. This is because the NUL: test uses the special JFN, .NULIO. As no GTJFN% is ever done on NUL:, no OPENF% is done and thus there is nothing to pass the MODE value to.

Since writing to NUL: has the expected effect of throwing data away, certain post-transfer operations (such as parity checking) are impossible.

TRANSMIT (*Enhanced*)

`TRANSMIT [[/SILENT] [/TIMEOUT n] [ /EOF ^Z | ^D | $ | ^G ]] filespec [ prompt | "prompt" ]`

Intended for use in local mode only, but will function in remote mode for debugging purposes. Sends the specified text file a line at a time to the remote system, waiting for the specified prompt for each line. Kermit protocol is not used; the file is sent "raw".

The file is expected to be a sequence of variable length ASCII lines, each terminated by a carriage return (\r or octal 15), linefeed (\n or linefeed). The final linefeed will be clipped out of the text that is sent. Problems may happen for files which use bare carriage returns for the purposes of overprinting.

Only a single file may be sent, no wildcards are allowed in the *filespec*. The file must exist on a file structure to which you either have or have gained access. No other device is permitted with the exception of NUL:, which is only used for debugging and testing purposes.

It is strongly recommended that this file not be particularly large nor have very long lines (see below).

The prompt is any string, for instance, the prompt of a line editor in text insertion mode. The prompt string may be enclosed in quotes and may include special characters by preceding their octal ASCII values with a backslash (e.g. \15 for carriage return) or C escape sequence (e.g., \r being converted into octal 15).

If a prompt string is supplied, then alphabetic case will be ignored in searching for it unless you have SET INPUT CASE OBSERVE. If a prompt string is not supplied, then the following defaults will be tried, in order:

1. If a SEARCH-DEFAULT has been specified with SET INPUT, then this will be used.
2. If you have performed a SET HANDSHAKE command, then that handshake character will be used.
3. If neither 1 or 2 apply, then linefeed will be used.

The file will be sent using the current settings for duplex, parity, and flow control. Timeouts may be given via setting an INPUT TIMEOUT or via the use of the /TIMEOUT switch You may do a /TIMEOUT 0 to cause Kermit to wait forever for each prompt to appear. You may use the following control characters during TRANSMIT:

Character	Function
Carriage Return	Send the next line immediately, even if the prompt hasn't appeared. Note that in remote mode, an input string which includes \r will not work.
Control-P	Resend the previous line.
Control-C	Typed two or three times will cancel the TRANSMIT command and return you to the Kermit-20> prompt. You may then need to do a number of CLEAR/FLUSH's.

Control-G	Functions as Control-C if the process does not have Control-C capability

/EOF

The optional /EOF switch may be used to indicate that a particular character should be sent when the End-of-File (EOF) condition is detected. The following EOF characters are supported:

Character	Background
^Z	Used by DEC operating systems, such as Tops-10 and Tops-20, IBM/Microsoft OS's, such as DOS, OS/2 and Windows. Note that the later systems will require a line terminator after the ^Z.
^D	Used by Unix type systems, such as Linux, Mac OSX and Ubuntu.
$	Dollar sign is used as a talisman for the escape character, which is used by Tops-10 and others as a line terminator.
^G	Used by certain DEC CUSPs, such as PIP and audible debugging. Be aware that if Kermit is run with Control-C capability disabled, then it will use ^G instead, which will override its use here.

File Content Considerations

The file is always sent as an eight bit stream, even if it is seven bit ASCII. An eight bit file is expected to never have bit 0 set in any byte. If no parity is in effect, this bit is left alone and sent as is, which may cause confusion on the receiving system.

If this turns out to be a problem, then SET PARITY SPACE (or ZERO), will keep bit 0 cleared. If parity is in effect, then bit 0 will be ignored and the byte is processed as if it were 7 bit ASCII, thus eventually overwriting the bit.

/SILENT

As TRANSMIT scans the character stream for the prompt, it displays the text on the terminal under the assumption that these are system responses to commands (which were perhaps via OUTPUT) that the user will need to see. Reviewing the responses is essential for understanding what might have gone wrong in the case of a failure.

However, this behavior might not be desirable for commands which produce copious amounts of output, in a batch environment or for certain types of debugging. For this reason, the /SILENT switch may be used to discard this text and not print it.

Be aware that any and all error messages will also be discarded when /SILENT is in effect. As such, /SILENT processing can not be set as a default for the transmit command. It must be issued for each and every command.

/TIMEOUT

By default, TRANSMIT will scan (and hence wait) forever for the specified prompt.

This can be overridden by using the /TIMEOUT switch which takes floating point number for the number of seconds to wait before transmitting the next line, whether or not the prompt has

been seen. The minimum period is .001 seconds or a single millisecond. Negative numbers will be rejected.

Communications and System Considerations

Great caution should be exercised when using this command as it can crash communications systems, hang network connections and otherwise wedge other terminal based data paths. The safest approach may be to split the file up into several smaller pieces and do the TRANSMIT of each separated by PAUSE's at off hours in a batch job.

Recall that the whole point of the Kermit protocol was specifically to avoid this situation with DECSYSTEM-20 PDP-11 front end. Although these systems are only museum pieces now, their network implementations can still hang if pushed too hard.

If the wedge is bad enough, somebody may have to reboot the remote computer, which risks making you unpopular with the staff or owner if it happens often enough.

If you really want to send the entire file without looking for any prompts, specify a prompt of "\0" (ASCII zero or NUL). This will almost surely hang any computer which does not have a dedicated, real time input facility, particularly if this is done at Ethernet speeds. This is commonly done only for testing and debugging purposes on pseudo-terminals.

REMOTE ERROR (*Enhanced*)

```
REMOTE ERROR [ text | "text" ]
```

An invisible command used to send an "E" packet to the remote system. This is largely used for debugging, but if the connection becomes wedged and doing a number of CLEAR's has not worked, this may get the remote Kermit into a defined state.

Text may be specified and the remote Kermit should display this.

Be aware that sending an unsolicited "E" packet may cause certain Kermit implementations to stop and exit. Tops-10 Kermit is one such example.

PWD (*New*)

```
[ [REMOTE] | [LOCAL] ] PWD
```

Print the current connected directory. This is very different from the idea of a working directory which may simply be seen as a default directory specification as connecting to a directory in Tops-20 changes access rights (See CWD (*Enhanced*), above).

Be aware that not all Kermit's implement PWD (Generic packet type "A"). Kermit-20 does not prior to edit 188. Tops-10 Kermit does not, the idea of a path being only somewhat similar to a working directory.

One work-around may be to try REMOTE SPACE as this often lists what is being used as the current working directory.

SET PARITY (*Enhanced*)

```
SET PARITY [ [ even | mark | none | odd | space
     [ /check-parity | /generate-only |
```

```
        /packets-only | /terminal-and-packets ] ] ]
```

If parity is being used on the communications line, you must inform Kermit-20 so it can both send the desired parity on outgoing characters, and either check or strip it from incoming ones.

Be aware that parity may exist and be enforced by the foreign host before the Kermit protocol itself ever starts. Thus, Kermit can not negotiate parity because no communications will be possible before it is set—therefore, you **must** specify parity beforehand or you may have problems up to and including the connection being dropped.

The parity types are even, mark, none, odd and space. "none" means no parity processing is done, and the 8th bit of each character can be used for data when transmitting binary files. This will make transferring eight bit data more efficient.

The domain on which parity may be applied (or generated) while sending is either /packets-only or /terminal-and-packets. The DEC-20 does not normally use parity on communication lines and thus Kermit strips parity on all received data. This domain can be overridden by specifying /parity-checking. The Tops-20 terminal driver (TTYSRV) will only generate even parity.

If nothing is specified, then the defaults are used, which are none, /terminal-and-packets and /generate-only; these are congruent with previous behavior.

If odd, even, mark, or space are selected, binary files will be transferred using 8th-bit-prefixing, but only if the other side agrees. If this is not the case, they cannot be successfully transferred. If none is specified, 8th-bit-prefixing will not be requested. Be aware that 8th-bit-prefixing may result in slower transfer speeds for 8 bit data.

SET PARITY should be used for communicating with hosts that require character parity, or through devices or networks (like TELENET) that add parity to characters that pass through them. Both KERMITs should be set to the same parity.

The specified parity is used both for terminal connection (CONNECT/SET LINE) and file transfer (SEND, RECEIVE, GET) along with direct text issuance and recognition (INPUT, OUTPUT and TRANSMIT). Specifying /packets-only will cause parity to be restricted from terminal connections and text issuance and recognition.

Unless the foreign host insists on parity, specifying parity provides no real benefit as Kermit already has powerful error detection and correction that is built into the protocol. Furthermore, any detection of a parity error is considered a transfer terminating condition as would be the case were real parity being used.

Parity is unnecessary on a pseudo-terminal as the data transfer is not going over any communications medium and is entirely internal to Tops-20. Tops-20 will generate parity on a pseudo-terminal, but this is only used for testing purposes. Tops-20 itself does not check parity although a front-end line may do this.

Parity is also unnecessary for a Network Remote Terminal as DECnet is already providing error detection and recovery at both the packet and certain transport levels (via DDCMP, for example). Note that Tops-20 will not generate parity on either a DECnet Network Remote Terminal.

Parity can be still be specified in these cases, however, yet is used only for packet debugging and checkout. The default terminal handling on pseudo-terminals and NRT's ignores parity on input. In this scenario, /packets-only and /parity-checking should be specified.

The current parity setting, the domains and whether the local terminal will tolerate parity can be displayed with SHOW LINE, viz:

```
Kermit-20>show line

TTY for file transfer:  22
 (job's controlling terminal, KERMIT-20 is REMOTE)
  Handshake:        None
  Flow-Control:     XON-XOFF
  Parity:           None
  Duplex:           Full
  Break Simulation: Disabled
  Controlling Type: PTY [Parity]
  Input Buffers:    1
  Output Buffers:   1
```

If the local line supports parity and the monitor supports querying line parity, then Kermit will report whether the line supports parity. If the line supports parity, then Kermit will self-configure to respect that parity. This may be seen below, viz:

```
@terminal speed 9600
@terminal parity
@kermit
TOPS-20 Kermit version 5.3(230)-5

Kermit-20>show line

TTY for file transfer:  5 [Console]
 (job's controlling terminal, KERMIT-20 is REMOTE)
  Handshake:        None
  Flow-Control:     XON-XOFF
  Parity:           Even
  Duplex:           Full
  Speed:            9600
  Break Simulation: Enabled, 3 NULs at 50 baud
  Controlling Type: FE [Parity]
  Speed:       9600 Bd
  Input Buffers:    1
  Output Buffers:   2
```

SET PROMPT (Enhanced)

```
SET PROMPT string | "string"
```

Set the Kermit-20 prompt to whatever character string you like. This is especially useful when connected to another DEC-20 through an auto-dialer or via a network, using another Kermit-20 on the remote system.

A unique prompt for each Kermit-20 will clear up any confusion about which one you're talking to which is almost mandatory for effective debugging. This is also useful in a batch job doing configuration of the remote Kermit-20 via OUTPUT and INPUT commands prior to putting it in SERVER mode.

The string may optionally be enclosed in double quotes. This will allow a macro to change the prompt. One of the uses of this is to change the prompt to be the same name as the macro that sets it. This can serve as a memory aid to recall what settings are in effect.

The string may include C escape characters, although most of them will be of little use if not flat out counter-productive by potentially invoking command parsing.

As an aid, when Kermit-20 is running local and using a DECnet connection, the prompt will default to the node name of the local system, otherwise, it defaults to "Kermit-20>" Such defaulting will also happen for a pseudo-terminal connection, but it should be kept in mind that it is really the same system on both sides.

SET FLOW-CONTROL (*Enhanced*)

`SET FLOW-CONTROL` *`option`*

For communicating with full duplex systems. The DEC-20 system is capable of regulating the flow of characters on the line using XON/XOFF flow control. If characters are coming into the DEC-20 too fast, the DEC-20 front-end will send an XOFF signal, Control-S, to tell the system on the other side to stop sending characters.

After it has finished processing the characters in its input buffer, it will send a Control-Q to tell the other side to resume sending. The other system does the same thing when the DEC-20 is sending data characters.

Kermit-20 will use XON/XOFF flow control on a full-duplex connection by default, and it will not use it on a half-duplex connection. The options of the SET FLOW-CONTROL command are XON-XOFF and NONE. Invisible synonyms for XON-XOFF and NONE are ON and OFF, respectively. If you SET FLOW-CONTROL to anything other than NONE, HANDSHAKE is set to NONE.

FLOW-CONTROL is unnecessary on both pseudo-terminals and DECnet Remote Network Terminals, although it is simulated on both. This is largely useless for two reasons.

First, the XON/XOFF can not be honored by the simulation quickly enough to do any real good in contrast to the front end which will start and stop transmission in milliseconds, typically within a single character.

Second, other flow controls exist which Kermit does not need to manage. For a pseudo-terminal, Tops-20 will not allow an overrun, putting whomever is clogging the pipe to sleep until such time as the partner has drained and processed the stacked up communications data. Similar functionality is implemented by DECnet but is not visible to the NRT.

FLOW-CONTROL functionality is kept for debugging and checkout.

Note: If you notice that you are seeing ^S/^Q's being sent during communications (and you are not on a VT100 using smooth scroll), then the DEC-20's front end is probably being driven too hard. This is a dangerous thing to do as it risks crashing it, throwing off all the other users. You should remediate by using a smaller packet size and small pauses between packets.

SET INPUT DEFAULT-TIMEOUT (*Enhanced*)

```
SET INPUT DEFAULT-TIMEOUT [ n ]
```

n is the number of seconds for an INPUT command to time out after not receiving the requested input, if no interval is explicitly given in the INPUT command.

The default timeout value is 10 seconds. Fractional seconds may be given for n, such as 1.25. These will be converted to the nearest millisecond value. Such short waits may be useful for communicating with extremely fast remote machines over very high speed lines or for testing.

A negative value will be rejected. A value of zero means to never time out.

A default timeout may be override on a case by case basis by explicitly specifying the value to the INPUT command.

CONNECT (*Enhanced*)

```
CONNECT [ [number] | [host::] | ["pseudo-terminal"]
     ["/stay" | "/timeout" [n] ] ] |
     ["close"] | ["kill"]
```

Establish a terminal connection to the specified destination system or reconnect to the currently active connection. A destination may be one of the three following choices:

1. A physical terminal, parsed as a simple octal number.
2. A DECnet node.
3. The keyword, "pseudo-terminal", for a loopback connection to the local host.

/STAY

All destination choices may be following with the /STAY switch, which instructs Kermit to not create a terminal fork, but rather stay at the local system; this can aid processing the INPUT/OUTPUT commands by eliminating a source of asynchronous reads.

/STAY, INPUT and OUTPUT are the recommended way to use Kermit-20 from a batch stream. Using the communications fork in that context will not work without special modifications to BATCON. Even an arcane sequence of pause commands may not work.

/STAY with only INPUT and OUTPUT will also circumvent certain Tops-20 DECnet buffering bugs.

/TIMEOUT

/TIMEOUT may be used to override this and takes a numeric argument to specify the number of seconds to wait before failing the connection attempt. The maximum duration is 94 seconds. The SET DELAY parameter is taken to be the default duration before declaring a connection time out.

The minimum timeout is 0.001 seconds or one millisecond. Negative timeouts will be rejected.

While /TIMEOUT can be specified for any connection, it only has relevance for DECnet connections. You can abort the connection attempt by typing a ^C. Typing a ^Y will get you the current connection status as seen by Tops-20.

CLOSE

Unlike SET LINE and SET HOST, giving CONNECT with no arguments merely reconnects to the previous line, remote system or pseudo-terminal. The connection may be explicitly closed with CONNECT CLOSE. However, the terminal fork will be frozen and kept for later connections.

KILL

The transfer fork may be killed separately. Killing the transfer fork will not close the connection, but could cause repeated or lost data which should be irrelevant as no active transfer is in process. In this case, the CLEAR command may be tried to clear out any data.

Cautions and Limitations

Be aware of the following cautions and limitations:

1. The physical terminal must not be in use or otherwise assigned to another job. You can not Kermit to another person's logged in job.
2. You can not connect to a local network terminal; the line must be a physical line.
3. If you specify a DECnet node and the parse is successful, then that node is guaranteed to be known to Tops-20. However, it may or may not be online and connections may take several seconds to complete. The connection can fail for other reasons (see below).
4. The remote DECnet host must support the Tops-20 NRT protocol; the connection will be rejected if this is not the case. Currently, only a PDP-10 running either Tops-10 or Tops-20 or Ultrix has such support.
5. When logging in via a pseudo-terminal, if you use the same user id as the user id with which you are currently signed on, you will not be prompted for a password.
6. Therefore, do not immediately type a password as it may wind up being echo'ed and also show up in a log file.
7. A pseudo-terminal connection is largely used for testing and debugging. Using it to run another program, such as TELNET to connect to an Internet host or CTERM a non-36 bit DECnet host is not guaranteed to work.

Specifying nothing will reconnect to the remote connection. If there is no remote connection currently open, then an error will be given.

Switching between hosts

Get back to Kermit-20 by typing the escape character followed by the letter C. The escape character is Control-Backslash (^\) by default. When you type the escape character, several single-character commands are possible:

Character	Function
B	Transmit a BREAK signal.

C	Return to the local Kermit-20; check-pointing log.
P	Push to a new Exec. POP from the Exec to get back to the connection.
Q	Temporarily quit logging.
R	Resume logging.
S	Show status of the connection.
^\	(or whatever you have set the escape character to be) -- Typing the escape character twice sends one copy of it to the connected host.

The "B" command is always accepted but silently does nothing on a pseudo-terminal nor on a DECnet NRT; it is only relevant for a physical line.

You can use the SET ESCAPE command to define a different escape character.

Be aware the "C" command does *not* close an active connection. Use the "close" keyword to accomplish this. Holding the connection open means that a subsequent connect command with no argument will have the effect of continuing the remote session. This is contrasted with SET LINE where specifying no parameter *will* close the currently active session.

SET LINE (*Enhanced*)

```
SET LINE octal-number | DECnet node name | "pseudo-terminal"
```

Specify the octal TTY number or DECnet node to use for file transfer or CONNECT. If you issue this command, you will be running as a local Kermit, and you must log in to the remote system and run Kermit on that side in order to transfer a file. If you don't issue this command, Kermit-20 assumes it is running remotely, and does file transfer over its job's controlling terminal line.

You can also select the line directly in the CONNECT command.

"pseudo-terminal" may be given for a loopback connection to the local host. This is largely used for testing, debugging and checkout. However, it is possible to use the remote job to sign into other systems using different protocols, such as CTERM and TELNET.

Note the restriction that the remote DECnet host must be running Tops-20 terminal transport, currently Tops-20, Tops-10 and Ultrix.

Be aware that, when logging into the pseudo-terminal, if you use the same user id as the user id which you are currently signed on as, you will not be prompted for a password. Therefore, do not type it as it may wind up being echo'ed and also show up in a log file.

Issuing a bare set line with no arguments closes the current active connection, if there is one.

SET HOST (*New*)

```
SET HOST DECnet node name | "pseudo-terminal" [/stay | /timeout]
```

Specify the DECnet node to use for file transfer or CONNECT. If you issue this command, you will be running as a local Kermit, and you must log in to the remote system and run Kermit on that side in order to transfer a file. If you don't issue this command, Kermit-20 assumes it is running remotely, and does file transfer over its job's controlling terminal line.

You can also select the host directly in the CONNECT command. The /STAY and /TIMEOUT switches are the same as on the CONNECT command.

"pseudo-terminal" may be given for a loopback connection to the local host. This is largely used for testing, debugging and checkout.

Be aware that, when logging into the pseudo-terminal, if you use the same user id as the user id which you are currently signed on as, you will not be prompted for a password. Therefore, do not type it as it may wind up being echo'ed and also show up in a log file.

Specifying neither will close any remote connection.

Note the restriction that the remote DECnet host must support Tops-10/Tops-20 NRT protocol. This is usually PDP-10 running either Tops-10 or Tops-20, but the protocol is simple enough that any DECnet host can implement it, which Ultrix does.

SET DELAY (*Enhanced*)

`SET DELAY` *`floating-point-number`*

How many seconds to wait before sending the first packet. Use when remote and SEND'ing files back to your local Kermit. This gives you time to "escape" back and issue a RECEIVE command before packets start arriving.

The normal delay is 5 seconds.

The delay may be given in fractions of seconds and this will be converted to milliseconds. Negative delays are rejected. If no number is given, then the delay will default to 10 seconds.

Note that the delay is ignored both when Kermit is running in LOCAL or SERVER mode as there is no need to escape back and forth.

SET ESCAPE (*Enhanced*)

`SET ESCAPE [ octal-number | ^ + another character ]`

Indicate what control character you want to use to "escape" from remote connections. The number is the octal value of the ASCII control character, 0 through 37 and 177.

You may also type the carat ("^") character followed by a single character indicating the control character you wish to use.

When you type whatever has been chosen as the escape character, you must follow it by a single-character "argument" as described in Switching between hosts, above.

Entering the escape character twice in a row will cause the escape character itself to be sent. As is the case with other Kermit implementations, the word "CLOSE" does not mean disconnect. It means bring you back to the local host while holding the remote connection open. It is best to think of Close as meaning Escape.

If you do not specify anything, Kermit will default the escape character to octal 34 (Control-Backslash ^\). If you wish to do this when defining a macro, simply type a comma and go on to the next SET value you wish to change.

Kermit will reject a number of escape characters, among the following:

1. ^C is rejected in all cases.
2. ^G is rejected in a Batch job or if Kermit is running without ^C capability (SC%CTC), which is typically while debugging.
3. If you are running with Flow Control on, then neither ^S (a.k.a XOFF, DC3 or octal 23) or ^Q (a.k.a. XON, DC1 or octal 21) will be accepted.
4. If you are running with handshaking, then the handshake character will not be accepted.

Care should be taken for escape character selection. For example, a ^O may produce unexpected results. Some experimentation may be useful.

STATISTICS (*Enhanced*)

Gives statistics about the most recent file transfer. A sample is shown with the explanation following:

```
Kermit-20>Statistics

Maximum number of characters in packet:  9000 received: 120 sent
Communications duration: 14.40904 (TOD: 43), analysis:

       Sent:      514          Efficiency:     [100% Overhead]
       Received:  1463321      Efficiency:        1.2262
       Total:     1463835      Efficiency:        0.8158

Total characters per second:     99.2104 KC/s
Effective data rate:              1.1876 MBd
Pseudo-efficiency:                6.2324 per cent
ILDB: 1457309  SIN:  5888  SIN Max:  249  BIN:  6012
Interpacket pause in effect: 0 ms

Timeouts: 0
NAKs:     0
```

The first line of output indicates what size packets were negotiated for the transfer.

The second line of output gives the time spent doing the transfer in the form of hh:mm:ss.mssnn. hh indicates the number of hours spent and can go over 24 hours. mm and ss indicate minutes and seconds respectively and do not go over 59. mss indicates the number of milliseconds and can vary between 0 and 999. nn indicates 10's of microseconds and can vary between 0 and 99, meaning 10 and 990 microseconds.

The TOD figure is the number of elapsed Tops-20 time-of-day (TOD) ticks. A TOD tick is a fixed fraction whose unit quantity is the number of seconds in a day (86,400) divided by 2 to the 18th power (262,144) or .329589844 seconds. This is quite close to, but *not* a third of a second.

The efficiency is a multiplicative factor that is derived from counters that are kept on a per transfer basis, A transfer is defined to be all files that were either sent or received by the last

command. It is calculated as follows: the total number of characters in every file is divided by the number of characters that were necessary to communicate them.

If the factor is over 1, then Kermit compression is saving time. If it is under one, then the total characters used, including prefixing, is causing extra bandwidth to be consumed.

Since Kermit will only support one send or receive file group at a time, one part of the transfer will always show as %100 overhead, as these are the acknowledgements sent for packets received.

The numbers kept for STATISTICS and the numbers kept for SHOW LINE are not related and have nothing to do with one another.

Pseudo-efficiency is used to simulate the efficiency of the communications line, which was useful for hardware serial lines where the baud rate was known. As it is almost impossible to know the speed of all legs of a network transfer, this number is simulated against data that is gathered with the TIME command.

The `ILDB: 1457309 SIN: 5888 SIN Max: 249 BIN: 6012` line indicates how efficiently Tops-20 is being used and is of interest largely to developers. Briefly, the higher the ILDB number is, the better as monitor buffering (via SIN) has been used instead issuing single byte inputs (BIN).

Note: if the speed of the machine and the communications lines are extremely fast and the packet sizes are at maximum (9000 characters), then millisecond transfer times will be seen. In such environments, pathologically short file sizes will see microsecond transfer times. These are all far less than the TOD granularity described above which was the previous Kermit-20 time base.

The maximum time precision of most Tops-20 systems is derived from the frequency of the KL clock which operates at 1 mHz. This is converted to DK10 clock units, which runs at 100 kHz unless using an external clock source, in which case the maximum frequency is 400 kHz. Assuming 100 kHz, the finest granularity that can be measured on an unmodified Tops-20 is 10 microseconds.

SHOW LINE (*Enhanced*)

When running as LOCAL and using network or pseudo-terminal connection, Kermit keeps extra performance information. This information is separate from what is shown in STATISTICS for a Kermit protocol based transfer.

In this case, it is specific to the virtual terminal use by CONNECT, INPUT, OUTPUT and TRANSMIT. The following is an example for a pseudo-terminal:

```
TTY for file transfer:  22 [PTY7:]
 (pseudo-terminal loopback to VENTI2::, KERMIT-20 is LOCAL)
  Handshake:         None
  Flow-Control:      None
  Escape Character: ^\
  Parity:            None
```

```
Duplex:           Full
Break Simulation: Disabled
PTY Connection:   Online
Pseudo Speed:       19.0549 MBd
Virtual CFIBF%'s: 57
SOUTR%'s Issued:  23
SOUTR% Bytes:     424
Max SOUTR% Len:   150
Network BIN%'s:   64
Network SIN%'s:   52
Network SIN% Cnt: 3509
Network SIN% Max: 249
Controlling Type: PTY [Parity]
Pseudo Speed:       19.0549 MBd
Input Buffers:    1
Output Buffers:   1
```

The following counters are kept on behalf of the operation of a virtual terminal being driven by direct user activity on a local terminal.

Terminal BIN%'s

This is when a single character is read from the local terminal for transmission over the network link.

Virtual CFIBF%'s

The Tops-20 buffer clearing JSYi (CFIBF% and CFOBF%) do not work on a network connection that is not using a physical line. The data must be read and discarded. This is the count of the total characters 'cleared' in this way.

Buffer CFIBF%'s

Kermit-20 uses buffered network I/O to implement the INPUT and OUTPUT commands. Kermit holds any remaining characters after a successful search in case other INPUT commands are given. On a CLEAR command, this contains the total of how many of those characters were removed.

This data is different from a Virtual CFIBF% as it has already been read, the result of the CLEAR being merely that INPUT's buffer counter and pointer were reset.

Virtual BOUT%'s

This is when a single character is sent over the network line as a result of being read and no further terminal input being available. In this case, a SOUTR% must be used for a single byte to push the data over. This has more overhead than other cases.

SIN%'s Issued

If input is detected on the local terminal after it has woken up from a BIN%, a check is made for any type-ahead. If any exists, it is captured with a counted SIN%. This is the number of times that has happened.

SIN% Bytes Total

Total number of type ahead characters we were able to capture with a SIN.

Max SIN% Length

The maximum size of the type ahead that we've seen. On a fast or lightly loaded machine, this number may not be very high. The more data captured this way, the less any data may be in a front end, thus avoiding instability.

SOUTR%'s Issued

Total times we've used a SOUTR% to push captured type ahead. This does not include the SOUTR% used to push a single character.

SOUTR% Bytes

Total characters that have been written to the network as a result of data produced by typeahead activity.

Max SOUTR% Len

The maximum number of characters that have been able to be written using a SOUT% This shows the best case conditions of capturing type-ahead. The number will be artificially high in a batch job because the batch controller issues text a line at a time.

The following counters are kept on behalf of the network input being done while operating the remote connection.

Network BIN%'s

This is when a single character is read from the network.

Network SIN%'s

When a character is read from the network, a check is made for any additional data that might be available. Any such data is read with a single SIN%, which cuts down on JSYS overhead. This is the total number of times Kermit has been able to do that.

Network SIN% Cnt

This is the total number of characters that were captured by detecting additional data.

Network SIN% Max

This is the maximum number of characters that could be read at once.

DEFINE (Enhanced)

```
DEFINE [macroname] | ["macroname"]
```

```
[ /macro-switch ] | [set-parameters]
```

or top level switch,

```
DEFINE /COMPACT | /DUMP | /MAP | /RESET | /SAVE | /SUMMARY
```

A macro switch may be one of the following:

```
/DUPLICATE | /RENAME | /UNDEFINE
```

Define a "SET macro" to allow convenient association of one or more SET parameters with a mnemonic keyword of your choice.

The set-parameters are a list of one or more SET options, separated by commas. If you use Kermit-20 to communicate with several different kinds of systems, you may set up a macro for each, for instance:

```
DEFINE IBM PARITY MARK, DUPLEX HALF, HANDSHAKE XON, SEND PACKET-LENGTH 80
DEFINE UNIX PARITY NONE, DUPLEX FULL, HANDSHAKE NONE
DEFINE TELENET PARITY MARK
```

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

Putting the macro name in double quotes allows the use of escape recognition of the rest of the SET parameters. Similarly, question mark ("?") can then be used to get a list of the available set parameters. This can ease the process of creating macros.

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

```
DEFINE UNIX
```

Be aware of the following:

Macro definitions may not 'nest'. This means that they may not include other macro names as set arguments.

Using a quoted macro name when defining will allow you to define macros that consequently can not be parsed. One example would be the use of embedded spaces or tabs, such as "my macro".

If this happens, then the macro may be removed by specifying the quoted name with the /UNDEFINE switch. It is best to keep macro names being alphanumeric with a dash ("-") or underscore ("_") as a separator character.

/COMPACT

```
DEFINE /COMPACT
```

If Kermit gives an error that the macro buffer is full, you can use the /COMPACT switch to run the garbage collector. The delay for this action is not normally noticeable.

/COMPACT will give some statistics to indicate what has been reclaimed, if anything. For example, a macro table that started out with 11 macros, with four being undefined produces the following:

```
Kermit-20>DEFINE/COMPACT
Before: Macros: 7 used, 114 remaining.
Available storage: 10705 characters
After:  Macros: 7 used, 114 remaining.
Available storage: 11315 characters
Elapsed: 0.00097 (TOD: 0), CPU:  0.00094
```

The timing figures indicate that 970 microseconds were spent doing the /COMPACT, of which 940 microseconds were processor related. Elapsed time is also known as "wall" time.

The /COMPACT switch is unnecessary if no macros have ever been undefined or redefined. Issuing it in that case will do absolutely nothing other than expend processor time, while at the same time providing no benefit whatsoever.

Similarly, issuing two or more /COMPACT switches in a row will not provide any further benefit.

Issuing a /COMPACT after a /RESET does nothing other than inform you that there is nothing to compact.

It is possible to define the maximum number of macros while still having remaining macro memory if the macro names and bodies are relatively short. In this case, while /COMPACT may be able to release storage for macro names and bodies, you will still not be able to DEFINE or /DUPLICATE a new macro because the limit being exceeded is the number of entries in the macro keyword table.

/DUMP

`DEFINE /DUMP` *`output file specification`*

The macro table may be written directly to disk in a binary format with the /DUMP switch, which takes an output file argument. /DUMP writes exactly what is in the table, so it will not write the smallest file unless you /COMPACT the table, first.

The binary file may be read with the /MAP switch. A smaller binary file can both be written faster by /DUMP and processed faster by /MAP. The size of the file produced by /DUMP bears little relation to what is created with /SAVE.

If you have a large number of macros, then it may pay not to define them in the KERMIT.INI file but rather have your KERMIT.INI map the binary definitions which can then immediately be used with no further parsing. The character definitions themselves can always be extracted with /SAVE.

The format of the binary file is designed for speed, not size. It can be used directly by Kermit. A file produced by /SAVE may actually be smaller, but takes far longer to process.

You may only dump to file structures. Dumping the file to NUL: is also allowed, but does not do anything of obvious value and is only provided for debugging and checkout.

Although the binary file is not human readable, you can find out how many macros are in the file by looking at the byte count in a directory listing, which is the number of macros contained therein, not the number of bytes. This example shows a binary file with 7 macros in it.

```
 STAR:<KERMIT.K20MIT>
                          PGS Bytes(SZ)  Write                Writer

K.BIN.5;P777700             1 7(0)       22-Jul-2022 22:47:59 SLOGIN

Total of 2 pages in 1 file
```

The word size is 0 because it is meaningless, the contents of the file having varying byte sizes. This will also prevent both Kermit and the EXEC from typing the file, which will produce nothing but gibberish that runs the risk of locking up smart terminals and terminal simulators.

If you wish to know what is in the file, use FILDDT and look at k20mac.

/DUPLICATE

DEFINE *old-macro-name* /DUPLICATE *new-macro-name* | "*new-macro-name*"

Takes the text associated with the indicated macro and creates a new macro whose name is specified after the /DUPLICATE switch and sets that new macro's body to be a copy of the text of the originally specified macro. For example:

```
Kermit-20>SHOW IBM
  IBM = parity mark, duplex half, handshake xon
Kermit-20>DEFINE IBM/DUPLICATE MBI
Kermit-20>SHOW MBI
  MBI = parity mark, duplex half, handshake xon
```

You can only duplicate an existing macro. The new macro's name can not be that of an existing macro. The new macro name may be specified inside of double quotes to allow for question mark and escape recognition after the field.

/DUPLICATE will optimally use macro storage space and thus it is not necessary to do a /COMPACT, afterwards.

/MAP

DEFINE /MAP *input file specification*

The binary file created by /DUMP can be read with the /MAP switch. It is *substantially* faster to use the /MAP switch on startup than to define all your macros in a TAKE file.

Kermit does a few checks on the binary file, but these are not exhaustive. If a binary file becomes corrupted, it can crash Kermit.

If this is a concern, you should also /SAVE your macros for recovery. If you have not done this, you may be able to fix things with FILDDT, the first part of the binary file being a TBLUK% table.

You can /MAP from NUL:, but this only does the equivalent of a /RESET and is provided for debugging and checkout purposes.

/RENAME

DEFINE *old-macro-name* /RENAME *new-macro-name* | "*new-macro-name*"

Changes the name of an existing macro, the text of the macro being entirely unchanged. It is different from /DUPLICATE in that the existing macro name will no longer exist and its text is not duplicated having been assigned to the new macro name.

Thus, the old macro text may now only be used by the new macro name. For example:

```
Kermit-20>SHOW IBM
  IBM = parity mark, duplex half, handshake xon
Kermit-20>DEFINE IBM/RENAME MBI
Kermit-20>SHOW MBI
  MBI = parity mark, duplex half, handshake xon
```

At this point, you can not do a SET IBM because the macro name IBM no longer exists; you must do a SET MBI, instead.

You can only rename an existing macro. The new macro's name can not be that of an existing macro. The new macro name may be specified inside of double quotes to allow for question mark and escape recognition after the field.

/RESET

DEFINE /RESET

The /RESET switch may be used to remove all macro definitions, resulting in a pristine macro table that is completely empty. Thus, the maximum macros, names and bodies will be available, viz:

```
Kermit-20>DEFINE /RESET
Kermit-20>DEFINE /SUMMARY
Macros: 0 used, 121 remaining.
Available storage: 12150 characters
```

Doing a /COMPACT after a /RESET does nothing useful because there is nothing to /COMPACT.

/SAVE

DEFINE /SAVE *output file specification*

You can save the macro table into a human readable (ASCII) file using the /SAVE switch, which takes an output file argument. It writes the macro table as a series of DEFINE commands, which can subsequently be used in a KERMIT.INI file or changed with a text editor.

It is not necessary to do a /COMPACT before a /SAVE; the resulting file will always have the exact same size as the undefined macro text will never be written. This is in contrast to a /DUMP where doing a /COMPACT beforehand can matter significantly.

The size of the file produced by a /SAVE does not bear an immediately obvious relation to what is created with /DUMP and it is not useful to compare the two as they are designed for different purposes, the former for readability and the latter for speed.

You can only /SAVE to file structures with two exceptions:

1. You can save to your currently logged in terminal (TTY:) to see what would be written to disk. The information displayed is effectively no different than what you would see from a SHOW MACROS. This functionality is primarily used for debugging and checkout.
2. You can /SAVE to NUL:, but this does nothing particularly useful and is only used for debugging and checkout.

You can not /SAVE to any other device nor someone else's terminal nor any other line, even if you have assigned that line.

/SUMMARY

`DEFINE /SUMMARY`

The /SUMMARY switch provides a brief analysis of current macro table resource usage, as follows:

```
Kermit-20>DEFINE /SUMMARY
Macros: 11 used, 110 remaining.
Available storage: 10705 characters
```

/UNDEFINE

`DEFINE` *`existing-macro-name`* `/UNDEFINE`

Another way remove a macro is to follow the macro name with an explicit /UNDEFINE switch. Therefore, the following two commands do the same thing:

```
DEFINE UNIX
DEFINE UNIX /UNDEFINE
```

If you've managed to define a macro name that turns out to be quirky in some way or completely unusable, you can put it in quotes to undefine it, for example:

```
DEFINE "WRONG WAY" /UNDEFINE
```

Be aware that undefining a macro does not automatically reclaim the memory for reuse as this operation is relatively computationally expensive and, since most macros are read from the KERMIT.INI file, is not expected to happen frequently, if ever.

See the /COMPACT switch for information on handling out of memory situations.

GET (*Enhanced*)

GET *remote-filespec* | "*remote-filespec*" [*local-filespec*]

For use only when talking to a remote KERMIT server.

When running as a local KERMIT (I.E., when communicating with a remote KERMIT over an assigned TTY line, which you have specified with the SET LINE command or having done a SET HOST command), you may use the GET command to request the remote KERMIT server to send you the specified files.

The remote-filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. As files arrive, their names will be displayed on your screen, along with "." and "%" characters to indicate the packet traffic. You may type Control-A to get a brief status report, ^X to request that the current incoming file be cancelled, or ^Z to request that the entire incoming batch be cancelled.

If the remote KERMIT is not capable of server functions, then you will probably get an error message back from it like "Illegal packet type". In this case, you must connect to the other Kermit, give a SEND command, escape back, and give a RECEIVE command.

Syntax: The first form of GET command accepts an arbitrary string, terminated by a carriage return. The string specifies a file or files in the syntax of the remote system; this may be enclosed in double quotes.

The normal DEC-20 command characters for help (?), comment delimitation (! and ;), and file indirection (@) are enabled at the beginning of the field, you can only enter filenames starting with one of these characters by "quoting" it with a Control-V, which is discarded before sending the string to the remote system. Otherwise, using double quotes may result in a more consistent parse.

If you want to store the incoming file name with a different name than the remote host sends it with, just type GET alone on a line; Kermit-20 will prompt you separately for the source (remote) and destination (local) file specification. If more than one file arrives, only the first one will be stored under the given name; the rest will be stored under the names they are sent with.

The second form of the command can be used gain the same functionality on a single line by enclosing the remote file specification in double quotes. Escape recognition can then be used on the local file specification. The following example gets an ITS file "foo bar" from the baz device and stores it locally as foo-bar.txt:

```
Kermit-20>GET "BAZ;FOO BAR" FOO-BAR.TXT
```

The local file must be stored on a mounted file structure to which the user has access. It is possible to store to NUL:, but this does nothing but throw the data away and is only used for testing and debugging purposes.