Tape Drives

Published by LMI 1000 Massachusetts Avenue. Cambridge MA 02138 USA

Introduction

This package contains the following manuals:

- *Tape Drive Adapter*: Specifications, maintenance, and basic operations of the LMI Magnetic Tape Drive Adapter.
- *STR50 Magnetic Tape Unit*: Specifications, operations, testing, and maintenance of the 1/2" LMI magnetic tape drive.

Tape Drive Adapter

Published by LMI 1000 Massachusetts Avenue. Cambridge MA 02138 USA

The following document covers specifications, maintenance, and basic operations of the LMI Magnetic Tape Drive Adapter.

This manual published April, 1985.

Please help us to make LMI documentation work better for you! Send comments via your customer dialup mail line to Sarah Smith (username SWRS) or by U.S. Mail to:

Dr. Sarah Smith
Director, Documentation
LMI
1000 Massachusetts Avenue
Cambridge, MA 02138

Table of Contents

LMI Customer Service is ready to help with your maintenance and expansion needs.

Our toll-free Customer Assistance Line is open from 8:30 a.m. until 8:00 p.m. Eastern Time.

Customers outside Massachusetts, please call:
•1-800-872-LISP

Customers within Massachusetts, please call:
•1-800-325-6115

1. General Description

The LMI Lambda uses a Ciprico Multibus compatible 1/2" magnetic tape drive adapter. It is mounted on a larger carrier card that plugs directly into the Lambda card cage. The adaptor functions in 8- or 16-bit systems, single or multiprocesor, with 16-, 20-, or 24-bit addressing. It features:

- Control of up to 8 start/stop or streaming, PE or NRZI formatted drives.
- Programmable for 8- or 16-bit systems.
- Full 24-bit addressing.
- DMA operation.
- Single or multi-master environments.
- *Buffered, Direct*, or *Streaming* data transfer modes.
- Bus lock option during DMA transfers.
- Programmable Interrupt option.
- Optional on-board buffer up to 16K bytes.
- Automatic retry for all recoverable errors.
- 64-byte buffer to ease demands on the system bus.
- Powerful *Block Move* and *Exchange* commands for generalized data handling.
- Extensive self-diagnostic commands.
- May be used to execute user-written 8089 programs.
- Single 5-volt operation.

2. Specifications

Physical Dimensions

Height: 6.75 in.

Width: 12.0 in.

The adapter board is mounted on a larger carrier card of standard size which is then plugged into the Lambda card cage.

Power

Voltage: +5v $\pm 5\%$

Current	Typical	Maximum
2K	2.55	2.60
16K	3.00	3.10

Connectors

Two 50-pin flat cables to the tape drive

J1, J2: 3M, No. 3425 or equivalent

P1: Viking, No. 2VH43/1AV5 or equivalent

Interface Fully Intel Multibus compatible

Operating Temperature

0° to 55° C

General

Capacity: 8 drives

Drives Controlled: All drives complying with industry standard formatted interface.

Transfer Rate Tape speed:

to 500 KBps (16 bit system)

to 330 KBps (8 bit system) (400 ns ACK)

MTBF: 71,000 hours

3. Error Codes

The following error codes for unrecoverable errors detected by the tape drive adapter are returned in bits 0 to 4 of the Command Status Field.

00	No unrecoverable error.
01	Timed out waiting for expected *Data Busy* false.
02	Timed out waiting for expected *Data Busy* false, *Formatter Busy* false, and *Ready* true.
03	Timed out waiting for expected *Ready* false.
04	Timed out waiting for expected *Ready* true.
05	Timed out waiting for expected *Data Busy* true.
06	A memory time-out occurred during a system memory reference.
07	A blank tape was encountered where data was expected.
08	An error occurred in the micro-diagnostic.
09	An unexpected EOT was encountered during a forward operation, or *Load Point* during a reverse operation.
0A	A hard or soft error occurred which could not be eliminated by retry.
0B	A read overflow or write underflow occurred. This error indicates that the FIFO was empty when data was requested by the tape during a write, or full when the tape presented a byte during a read.
0C	Not used.
0D	A read parity error occurred on the byte interface between the drive and the tape drive adapter.
0E	An error was detected while calculating a checksum on the PROM.
0F	A tape time-out occurred, because the tape drive did not supply an expected read or write strobe. This normally occurs when attempting to read a larger record than was written.
10	Tape not ready.
11	A write was attempted on a tape without a write-enable ring.
12	Not used.
13	The diagnostic mode jumper was not installed while attempting to execute a Diagnostic command.
14	An attempt was made to link from a command which does not allow linking.
15	An unexpected filemark was encountered during a tape read.
16	An error in specifying a parameter was detected by the tape drive adapter.
17	Not used.

18	An unidentifiable hardware error occurred. Call LMI.
19	A streaming read or write operation was terminated by the operating system or disk.

STR50
Magnetic Tape Unit

Published by LMI 1000 Massachusetts Avenue. Cambridge MA 02138 USA

The following document covers specifications, maintenance, and basic operations of the LMI STR50 Magnetic Tape Unit.

This manual published April, 1985.

Please help us to make LMI documentation work better for you! Send comments via your customer dialup mail line to Sarah Smith (username SWRS) or by U.S. Mail to:

Dr. Sarah Smith
Director, Documentation
LMI
1000 Massachusetts Avenue
Cambridge, MA 02138

Table of Contents

LMI Customer Service is ready to hclp with your maintenance and expansion needs.

Our toll-free Customer Assistance Line is open from 8:30 a.m. until 8:00 p.m. Eastern Time.

Customers outside Massachusetts, please call:
•1-800-872-LISP

Customers within Massachusetts, please call:
•1-800-325-6115

1. General Description

The STR50 half-inch magnetic tape drive incorporated with the Lambda uses a dual-gap head, providing read-after-write capability. Read/write, control, and formatting electronics are all contained on a single printed-wiring board. The transport is designed to operate on 85 to 132 Vac or 195 to 263 Vac, single-phase, 48 to 61 Hz line power. It can accommodate reels up to 10.5 inches in diameter. Tape speed and density capabilities are 25 ips at 1600 bpi, or 100 ips at 1600 bpi in streaming mode.

1.1 Power Connection

A power cord is supplied only for the 85 to 132 Vac range indicated above.

> **CAUTION**: To prevent damage to the STR50 and ensure proper operation, be sure the outlet voltage is correct before applying power to the tape drive.

> **CAUTION** The weight of the STR50 can upset an inadequately mounted equipment rack. If the STR50 is to be extended on slides from the equipment rack, be sure that it is mounted securely.

2. Normal Operation

2.1 Controls and Indicators

The illuminated panel on the front of the STR50 (Figure 1) is used both as a control panel and to indicate the status of the machine. Table 1 lists the control/indicator type, its functions, and the conditions required for enabling these functions.

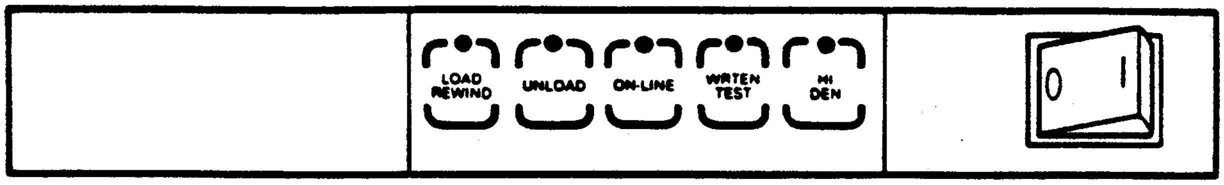


Figure 1. Control Panel

2.2 Loading Tape

CAUTION: Both the front-panel door and the top cover are locked during tape-loaded functions. Do not attempt to open either the top cover or the front-panel door during load operation or while tape is loaded in transport.

To load tape:

- Apply power to unit and verify that **UNLOAD** indicator is illuminated. Allow for a normal delay of about 2 seconds.
- Insure that tape is wound completely onto reel.
- Open plexiglass front-panel door by pressing down gently on top (center) of door.
- Insert tape into front of unit with the write-enable ring facing down.
- Close front-panel door.
- Press LOAD switch. The access doors are now locked. The **LOAD** indicator will blink,then remain illuminated when the load sequence is completed.
- Press ON-LINE switch to place tape on line.

2.3 Unloading Tape

NOTE: Transport must be in off-line mode. (The **ON-LINE** indicator light should be off.)

To unload tape:

- Press **UNLOAD** switch. While the tape is being unloaded, the **UNLOAD** indicator will blink and the access doors will remain locked. When the unload sequence is completed, the **UNLOAD** indicator will remain illuminated and the access doors will unlock.
- Open the front-panel door when the **UNLOAD** indicator stops blinking.
- Carefully remove the tape reel.
- Close the front-panel door.

Table 1 Controls and Indicators

CONTROL/ INDICATOR	TYPE	FUNCTIONS	CONDITIONS
POWER	ON/OFF Rocker switch and indicator	Switches the line power on and off.	Fuse installed. Line cord connected.
LOAD REWIND	Tactile switch and indicator	Loads tape to BOT marker. Rewinds tape to BOT marker. Illuminates to indicate BOT tab is positioned at photo-sensor. When pulsing, transport is executing a load or a rewind sequence.	Tape inserted in front panel door. Top cover and front panel door closed. Transport in off-line mode (ON-LINE indicator off).
UNLOAD	Tactile switch and indicator	Unloads tape from any point. UNLOAD indicator flashes during unload sequence, then remains lit.	Transports in off-line mode. (ON-LINE indicator off.)
ON-LINE	Tactile switch and indicator	Toggles transport on and off line. Indicator lights up when transport is in on-line mode.	Load sequence stops when BOT marker is sensed. Pressing on-line switch at this point places tape on line.
TEST	Tactile switch	Selects alternate operational mode for other switches.	For LMI diagnostic use only.

WRT EN (Write Enable)	Indicator	Illuminates to indicate write function may be performed.	Tape reel write-enable ring installed, mounted on supply hub, and tape loaded.
HI DEN (High Density)	Tactile switch and Indicator	Lambda operates in low-density mode (1600 bpi).	HI DEN light off.

3. Error Conditions

3.1 Error Indicators

The lighted control panel on the front of the STR50 is used to indicate operating failures or fault conditions. These errors can be:

Operator Errors

These error conditions normally occur during normal tape loading operations and are usually caused by improper loading procedures. Error signals will be displayed as a steady pattern of indicator lights on the front panel. See Table 2 for a listing of error codes.

Transport Errors

These codes indicate a serious deviation from the normal operating routine of the STR50 and may require correction by an experienced service technician. They are shown as a unique pattern of the front panel indicators with a quick double flash to alert the operator. See Chapter 4 for troubleshooting instructions.

3.2 Manual Load

When the autoload routine has failed to load a tape successfully, it may be necessary to load the tape manually. To do this:

- Extend the STR50 unit on its slides to clear the equipment rack.
- Lift the top cover and place the cover support in the slot provided.
- Place the reel of tape on the supply hub. Be sure that the reel is evenly seated on the hub.
- Depress and hold the manual unlock button. (Open the plexiglass door in the front panel. The button is on the bottom left-hand side of the tape reel opening.) Simultaneously rotate the supply hub clockwise until the supply reel is locked in place. A definite click should be felt.
- Thread the tape along the path shown in Figure 2. Carefully move the tachometer assembly away from the takeup hub, wrap tape clockwise around the takeup hub, and gently replace the tachometer assembly. Check that the tape is seated correctly on the guides and threaded properly over the head assembly.
- Close the top cover and place the transport in normal operating position.
- Depress and hold the HI DEN switch, then press the LOAD switch and release both. Tape should tighten and advance until the Beginning of Tape (BOT) tab reaches the

photosensor. The LOAD indicator will be illuminated, indicating that the STR50 is ready for use.

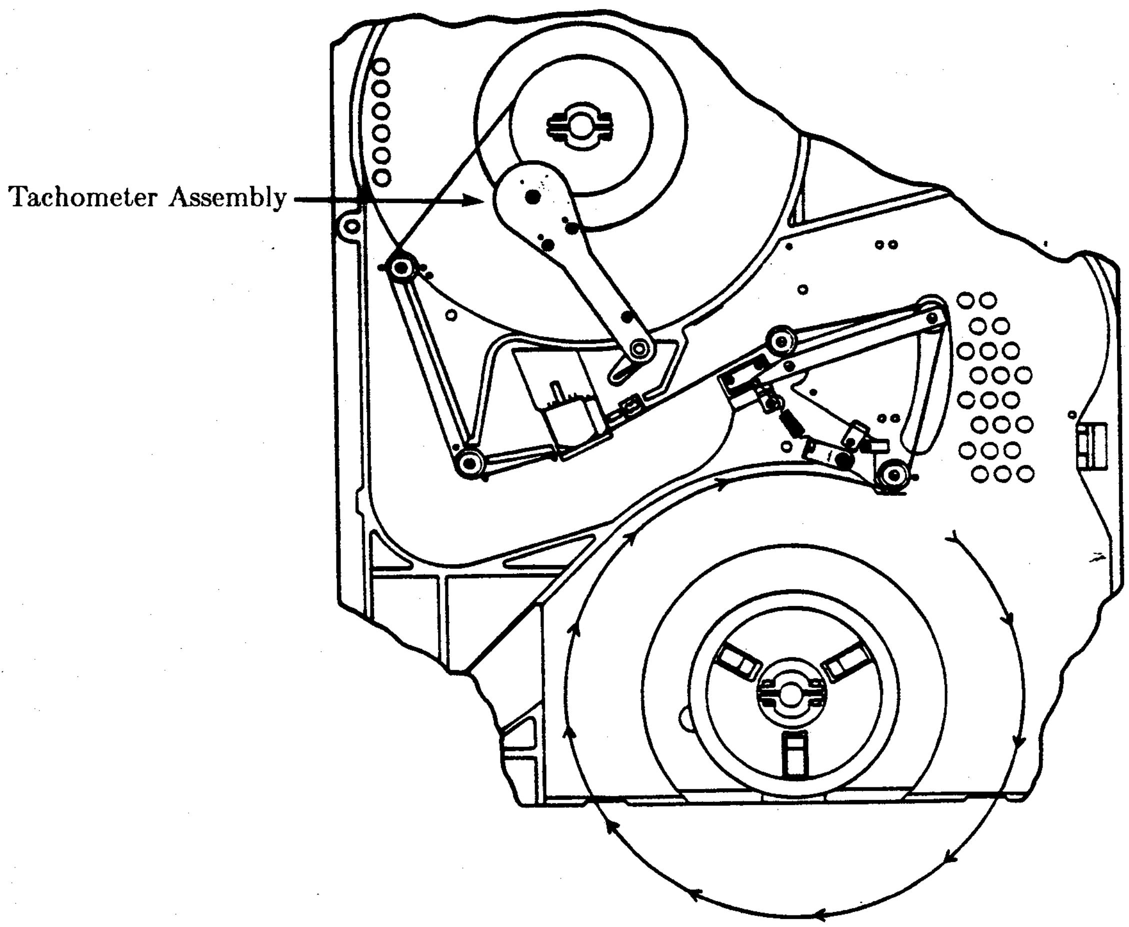


Figure 2. Tape Threading Path

Table 2 Operator Error Frcnt Panel Indications

INDICATION	CONDITIONS
All indicators flashing	STR50 unable to complete load sequence. Check tape leader for damage and try again. If still not successful, see Section 3.2.
All indicators flashing except LOAD.	BOT marker not detected within the first 35 feet of tape. The leader must be at least 6 feet in length.
All indicators flashing except UNLOAD	Tape reel was inserted upside down. The tape reel should be inserted with the slot for the write-enable ring facing down.
All indicators flashing except ON-LINE	Load or unload operation was attempted with the front-panel or top cover door open.
All indicators flashing except TEST	Load operation was attempted without a tape reel in place.

3.3 Manual Unload

If the STR50 does not complete the rewind/unload sequence successfully, the tape reel may be rewound manually. To do this:

- Lift the top-cover sides behind the front panel. Place the cover support in the slot provided.
- Rotate the supply reel counterclockwise to rewind the tape onto the supply reel.
- Press the manual unlock button. (Open the plexiglass door in the front-panel. The button is on the bottom left-hand side of the tape reel opening.) Simultaneously rotate the supply reel counterclockwise until it turns freely and can be removed from the transport.

4. Testing and Troubleshooting

4.1 Testing

The STR50 incorporates a power-up self-test and additional error detection during tape operation. During a power-up operation, all indicator lights on the front panel should be illuminated for about one second. If no problems are encountered, all lights except the **UNLOAD** indicator should go out. Continued illumination of any other switches may indicate a failure of the ROM or RAM test. If this happens, power cycle. If the error indications persist, call LMI.

4.2 Troubleshooting

The following tables list malfunction symptoms with the probable cause and simple remedial actions. If more serious problems are encountered, please call LMI for technical assistance.

Table 3	Power-up Malfunction Symptoms
Table 4	Operator Error Symptoms
Table 5	Transport Failure Symptoms
Table 6	System Failure Symptoms

Table 3 Power-up Malfunction Symptoms

SYMPTOM	CAUSE	REMEDIAL ACTION
Failure to complete power-up sequence. Transport unable to initiate any local or remote commands.	During a normal power-up operation, all the indicators on the front panel will light up for about 1 second. All except the UNLOAD light should then go out. No defect is indicated by this sequence.	
	Any invalid fault code (see tables 4-6) indicates failure. Fan operation during power-up also indicates failure.	Call LMI.

Table 4 Operator Error Symptoms

SYMPTOM	CAUSE	REMEDIAL ACTION
All indicators flashing	Tape leader damaged	Remove damaged tape. Replace BOT marker if necessary.
	Transport cannot complete load sequence	Call LMI.
All indicators flashing except LOAD	BOT marker not detected within the first 35 feet of tape.	Check tape for BOT marker.
All indicators flashing except UNLOAD	Tape reel inserted upside down.	Insert reel correctly.
	Tape-in-path sensor failed.	Call LMI.
All indicators flashing except ON-LINE	Load operation attempted with front panel door or top cover open.	Check door closure.
All indicators flashing except TEST	Load operation attempted without reel of tape inserted in unit.	Open top cover. Be sure reel is seated on supply hub. If not, seat reel properly and try again. During the load operation, check to see if the supply servo is rotating counterclockwise. If the reel is seated correctly and rotation is counterclockwise, call LMI for assistance.

Table 5 Transport Failure Symptoms

SYMPTOM	CAUSE	REMEDIAL ACTION
LOAD and UNLOAD indicators flashing	More than 3700 feet of tape beyond the BOT marker.	Usually caused by a long reel of tape. Try another reel; if problem persists, call LMI.
ON-LINE indicator flashing	Tension arm swing exceeded the normal operating range during auto load sequence.	Occurs only during load operation. Open top cover. If tape is not wrapped around correctly, or if tape is wrapped correctly but load sequence will not continue, call LMI.
LOAD and ON-LINE indicators flashing	Interface command received prior to completion of the previous command.	Usually caused by system failure. Call LMI.
UNLOAD and ON-LINE indicators flashing	Write command received with a write-protected reel of tape in place.	Reset error code and reload tape. Check to see if the WRT/EN light goes off. Call LMI.
LOAD, UNLOAD, and ON-LINE indicators flashing	Illegal or undefined command received.	Check cables and interface command lines. Call LMI.

TEST indicator flashing	Failure of supply hub locking mechanism.	Failure occurs only during load sequence. If reel appears to lock correctly, call LMI.
UNLOAD and TEST indicators flashing	Auto-zero function of the digital-to-analog converter failed during power-up.	Call LMI.
ON-LINE and TEST indicators flashing	Supply reel not seated on hub, or failure of file protect circuit.	Reseat tape reel and try again. If problem persists, call LMI.
LOAD, ON-LINE, and TEST indicators flashing	Supply reel did not remain locked during tape unload operation.	Call LMI.
TEST, UNLOAD, and ON-LINE indicators flashing.	Controller error: tape travel exceeded 18 feet past EOT marker.	Call LMI.
LOAD and HI DEN indicators flashing	Tape buffer tension arm exceeded its free travel limits.	Call LMI.
UNLOAD and HI DEN indicators flashing	Tape speed variations in excess of +10% occurred. Usually caused by bad tachometer assembly when drive is under system operation.	If failure occurs during power-up, check that the take-up hub moves momentarily counter-clockwise, then clockwise during powerup. Call LMI.

Table 6 System Failure Symptoms

SYMPTOM	CAUSE	REMEDIAL ACTION
Read or write errors during system operation.	System is unable to complete data transfer.	Try to read a known good tape to see if errors are caused by read or write logic. If errors still occur, call LMI. If the tape is read successfully, the problem is in the write formatter circuitry. Call LMI for assistance.
Tape reel cannot be removed from transport.	Tape not wound completely on supply reel or tape reel.	Be sure that tape is wound completely on supply reel after unload operation. If tape is wound completely on supply reel, verify that the tape reel is unlocked. Call LMI.
STR50 "runs away" with Data Busy false.	Transport formatter no longer controlling tape motion.	Call LMI.

Transport "runs away" with Data Busy true.	Transport formatter no longer controlling tape motion.	First, check read threshold and verify that it is in proper operating range. If transport was executing read operation when runaway occurred the read formatter may be the problem. Call LMI. If transport was executing a write operation, the write formatter may be the problem. Call LMI.
Doors will not lock or unlock. Operator unable to insert tape into transport.	Door lock malfunctioning.	Verify closure.
When drive is placed ON-LINE, tape unloads.	Transport will not operate in on-line mode.	Disconnect cables between transport and computer. If problem still exists, transport is at fault. Call LMI.
System detects invalid interface signals: IFBY IRDY IDBSY IFPT ILDP IEOT IONL IRWD ISPEED	System unable to verify correct transport status.	Interface signals may be at fault. Call LMI.

Transport ignores all commands sent by the controller, or transport executes a command other than the command issued by the controller.	System unable to initiate any remote command.	Check interface cable connection between drive and controller. Check command lines.
System is unable to select transport.	Invalid status indications from transport to controller.	Check interface cable connection to transport.

Disk Drives

Published by LMI 1000 Massachusetts Avenue. Cambridge MA 02138 USA

Introduction

This package contains the following manuals:

- *Multibus Controller*: Specifications and operations of the LMI Multibus disk controller board.
- *MSU169 Disk Drive*: Specifications and operations of the LMI MSU169 disk drive.
- *MSU474 Disk Drive*: Specifications and operations of the larger LMI MSU474 disk drive.

Multibus Controller

Published by LMI 1000 Massachusetts Avenue. Cambridge MA 02138 USA

The following document covers specifications, maintenance, and basic operations of the LMI Multibus Controller Board.

This manual published April, 1985.

Please help us to make LMI documentation work better for you! Send comments via your customer dialup mail line to Sarah Smith (username SWRS) or by U.S. Mail to:

Dr. Sarah Smith
Director, Documentation
LMI
1000 Massachusetts Avenue
Cambridge, MA 02138

Table of Contents

LMI Customer Service is ready to help with your maintenance and expansion needs.

Our toll-free Customer Assistance Line is open from 8:30 a.m. until 8:00 p.m. Eastern Time.

Customers outside Massachusetts, please call:
•1-800-872-LISP

Customers within Massachusetts, please call:
•1-800-325-6115

1. General Description

The LMI Lambda uses an Interphase Multibus controller board. The disk controller board is mounted on a larger carrier card that plugs directly into the Lambda card cage. The disk controller serves as a bus master during data transfers, using a variable burst length DMA technique. It can directly connect from one to four storage module drives by industry standard A and B cables. The controller board can control any type of drive with an SMD compatible interface.

Devices on the Multibus may command the controller to perform a disk function, such as READ, or WRITE a sector (or more) of data into or out of System Memory. All such functions have an extended list of parameters to define the exact function to be performed. This list is called the I/O Parameter Block (IOPB) and is found in common memory; that is, memory accessible to both the LMI Lambda CPU, and the controller board. In order to cause a disk function to be performed, the processor builds the IOPB in memory, writes a pointer to the IOPB into the Address Registers, and writes a *GO* to the Command Register. The function is automatically completed by the controller. Both an *Operation Done* interrupt and *Done Status* are provided.

The disk controller supports a wide variety of system configurations including:

- 8 and 16 bit (or mixed) systems
- Single or multiple CPU and other bus masters
- Serial and parallel bus priority
- Single user or multiuser-multitask environments
- Absolute or relative addressing modes
- Operation in *Buffered mode*, or *Direct mode.*

Multiple disk drive types with varying speeds and capacities can be controlled simultaneously.

2. Specifications

2.1 Physical Dimensions

Height:	6.75	in.	(to ejector)
	7.5	in.	(to cable connector)
Width:	12.0	in.	
Thickness:	.5	in.	
Weight:	14.0	oz.	

The disk controller board is mounted on a larger carrier card which is plugged into slot 16 of the Lambda card cage.

2.2 Power

```
 5 VDC ± 5%, 3.75 AMPS
-5 VDC ± 5%,  .6  AMPS
```

2.3 Connectors

Bus:	Card edge:	86 pins on .156 center
Board:	A cable:	60 pins
	B cable:	26 pins
Maximum cable length:		50 feet

2.4 Operating Temperature

Ambient temp: 0° to 55° C

3. General Operations

3.1 Status Code

Once a disk has been accepted, operation status is provided. The three possible status indicators are:

- 80H Operation successful, ready for next command
- 81H Operation in progress, busy
- 82H Error on last command

3.2 Error Codes

A status code of 82H after an operation will provide one of the following error codes:

NOTE: All error codes are hexadecimal.

10 DISK NOT READY

The disk's ready signal output is tested at the beginning of any command requiring a head movement (all commands except **Reset**). Error **10** is posted if the disk is not ready. Note that error **18** is issued if the drive is not powered up.

11 INVALID DISK ADDRESS

The unit selects bits in the I/O Parameter Block (IOPB) which are examined for the presence of a valid unit selection (0-3). Check on all commands except **Reset**.

12 SEEK ERROR

All commands except **Initialize** and **Reset** may cause a **Seek** operation to be initiated. The controller issues a **Seek** command to the disk drive, and, on completion, reads the header of the appropriate sector to verify the location of the head. If wrong, the SMD 2181 will execute a **Restore** and then **Reseek** the target track. The header will be read again and if the track is still wrong, the controller will post **Error 12 Seek Error**.

13 ECC CODE ERROR—DATA FIELD

The computed Error Correction Code (ECC) on the data did not agree with the ECC code appended to the data on the disk, and no error correction was attempted. NOTE: See also Error 23.

14 INVALID COMMAND CODE

The command code, byte 0 in the IOPB, was not valid.

15 NOT USED

16 INVALID SECTOR IN COMMAND

The target sector in the IOPB, byte 8 and 9 in the IOPB, was greater than the capacity of the drive as specified in byte 1 of that drive's Unit Initialization Block (UIB). This check is performed after the **Seek** has been done.

17 SPARE

18 BUS TIMEOUT

Bus acquisition was not made within 1 msec of a request, or XACK was not received within 1 msec of a MRDC/, MWTC/, or IOWTC/.

19 NOT USED

1A DISK WRITE PROTECTED

Posted when attempts are made to write to a disk that is write protected.

1B UNIT NOT SELECTED

A unit select was made and the unit failed to respond with unit selected. This error is returned when the drive unit number is misselected, the drive is not powered up, or the cable is not connected.

1C NO ADDRESS MARK—HEADER FIELD

This error is posted if no sync information is found in the header of the target sector.

Error correction will not be attempted on the header field.

1D NOT USED

1E DRIVE FAULTED

A Fault condition exists in a selected unit. The Fault should be cleared by a **Restore** command.

1F NOT USED

20 NOT USED

21 NOT USED

22 NOT USED

23 UNCORRECTABLE ERROR

Error correction was attempted on the data field and the error was found to be uncorrectable.

24 SPARE

25 SPARE

26 NO SECTOR PULSE

The sector pulse is missing from a selected unit.

27 DATA OVERRUN

A data field timeout error generally caused by missing TX or RX clock.

28 NO INDEX PULSE ON WRITE FORMAT

During a **Write Format** operation the disk controller looks for the index pulse from the disk. If not found within 65 msec., the error is posted. No retries.

29 SECTOR NOT FOUND

If at any point during a **Read** or **Write** type operation the target sector cannot be found, this error is posted.

2A ID FIELD ERROR—WRONG HEAD

The head number read from the disk in the header field was wrong.

2B INVALID SYNC IN DATA FIELD

The first byte read from the data field was not a valid sync character.

2C NOT USED

2D SEEK TIMEOUT ERROR

A seek was made and a normal complete response did not occur within 500 msec.

2E BUSY TIMEOUT

On a dual ported drive, BUSY has been active for more than 500 msec.

2F NOT ON CYLINDER

The drive must be ON-CYLINDER within three seconds after being selected.

30 RTZ TIMEOUT

A **Restore** command was executed and a normal complete did not occur within three seconds.

31 FORMAT OVERRUN ON DATA

40 UNIT NOT INITIALIZED

A command was attempted on a unit that has not been initialized.

42 GAP SPECIFICATION ERROR

4B SEEK ERROR

A seek error was reported by the disk drive.

4C MAPPED HEADER ERROR

No sector pulse found on track to be mapped.

51 BYTES/SECTOR SPECIFICATION ERROR

The bytes/sector in the UIB, bytes 2 and 3, exceed the capacity of the buffer.

52 INTERLEAVE SPECIFICATION FACTOR

The interleave factor in the UIB, byte 6, fails a sanity check.

53 INVALID HEAD ADDRESS

The target head in the IOPB, byte 5, was greater than the capacity of the drive as specified in byte 1 of that drive's UIB.

MSU169 Disk Drive

Published by LMI 1000 Massachusetts Avenue, Cambridge MA 02138 USA

The following document covers specifications, maintenance, and basic operations of the LMI MSU169 Disk Drive.

This manual published April, 1985.

Please help us to make LMI documentation work better for you! Send comments via your customer dialup mail line to Sarah Smith (username SWRS) or by U.S. Mail to:

Dr. Sarah Smith
Director, Documentation
LMI
1000 Massachusetts Avenue
Cambridge, MA 02138

Table of Contents

LMI Customer Service is ready to help with your maintenance and expansion needs.

Our toll-free Customer Assistance Line is open from 8:30 a.m. until 8:00 p.m. Eastern Time.

Customers outside Massachusetts, please call:
•1-800-872-LISP

Customers within Massachusetts, please call:
•1-800-325-6115

1. General Description

The LMI Lambda can be equipped with an MSU 169 Disk Drive with a maximum unformatted storage capacity of 168 megabytes. The 8-inch rigid disk drives are non-removable and are contained in a sealed module. Head positioning is performed by a rotary actuator using a closed loop servo.

The MSU169 uses a standard SMD interface, allowing the drives to be added to an existing disk configuration.

The MSU169 is mounted on slides in the base of the Lambda with room for a second MSU169 to be installed beside it.

2. Features

- Reliability
    - The MSU169 features Winchester type technology contact-start/stop (CSS) heads and media.
    - LSI circuits on each head amplify the small signal, thus reducing read errors by increasing the signal-to-noise ratio.
    - Heads, media, and positioning mechanism are sealed in a closed-loop air filtration system.
    - Electrical components located within the sealed disk area are minimized.
- Maintainability

  No scheduled maintenance is required. The use of a completely sealed disk enclosure, a belt-eliminating built-in DC spindle motor, and highly reliable printed circuit assemblies greatly reduces the need for maintenance.
- Compact size

  The unit can be mounted with two drives in a standard 19-inch rack. It weighs approximately 30 pounds.
- Low accoustical noise level and low vibration.

3. Specifications

3.1 Physical

Height:	5.0 in.
Width:	8.5 in.
Depth:	15.0 in.
Weight:	30 lbs.

3.2 Environmental

Temperature:

Operating	5^o to 40^o C.
Non-operating	-40^o to 60^o C.
Gradient	Less than $\pm$ 15^o C./hour

Humidity:

Operating	20% to 80% RH
Non-operating	5% to 95% RH

3.3 Power

DC Voltage	**Load Current (Basic)**	**Load Current (Dual Port)**
+5V $\pm$5%	3.5A	4.5A
-12V $\pm$5%	3.0A	4.0A
+24V $\pm$10%	3.6 Arms (Effective, typical)	
	7.2 Ao-p (Maximum)	
	4.6 Arms (POWER ON; Effective typical)	

The load currents of +5V DC and -12V DC will be stable during any operation being performed within the disk drive. However, the load current of +24V DC will be varied through a power up sequence or DC motor acceleration and/or seek operation.

4. Controls and Indicators

4.1 Powering Up/Down

The MSU169 is not equipped with a power ON/OFF switch. Powering up or down of the MSU169 is typically performed by powering up or down the system.

4.2 Operator Panel

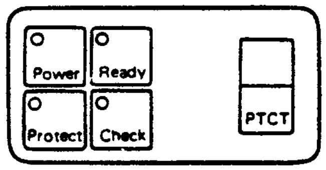


Figure 1. Operator Panel

The functions of the LED's and switches on the front panel are:

Power indicator: Red
LED lights when the power is turned on.

Ready indicator: Red
LED indicates the initial seek has been performed, or the termination of a *Seek* or *RTZ* operation.

Check indicator: Red
LED indicates a fault condition.

Protect indicator: Red
LED indicates that writing is inhibited.

Protect (PTCT) switch: White
Key inhibits the write operation.

Check clear switch: Gray (flat key)
Key resets a *Device Check* status.

MSU474 Disk Drive

Published by LMI 1000 Massachusetts Avenue. Cambridge MA 02138 USA

The following document covers specifications and basic operations of the LMI MSU474 Disk Drive.

This manual published April, 1985.

Please help us to make LMI documentation work better for you! Send comments via your customer dialup mail line to Sarah Smith (username SWRS) or by U.S. Mail to:

Dr. Sarah Smith
Director, Documentation
LMI
1000 Massachusetts Avenue
Cambridge, MA 02138

Table of Contents

LMI Customer Service is ready to help with your maintenance and expansion needs.

Our toll-free Customer Assistance Line is open from 8:30 a.m. until 8:00 p.m. Eastern Time.

Customers outside Massachusetts, please call:
•1-800-872-LISP

Customers within Massachusetts, please call:
•1-800-325-6115

1. General Description

The LMI Lambda is equipped with a MSU474 Disk Drive. The moving head drive has a storage capacity of up to 474 megabytes (unformatted). It uses Winchester type heads and platters, allowing high recording density, data transfer rate, and a high degree of reliability with rapid access time. The media is non-removable. The drive is appropriate for large-capacity, high-speed data storage in an on-line system.

The drive is designed to meet the following standards:

- UL478 Electronic Data Processing Unit and Systems
- CSA C22.2 No. 154-1975 Data Processing Equipment. (Under investigation)

2. Features

Large Capacity and High Performance

The MSU474 provides 474 megabytes of unformatted data on six disks, with high performance characteristics such as 1.859 megabytes per second data transfer, 18 milliseconds average access time, and 7.58 milliseconds average latency time.

Compact Size

The disk drive unit is configured with Disk Enclosure (DE), DC power supply and LSI curcuits. It is available for mounting in a standard 19-inch rack.

High Reliablilty

The disk enclosure includes a rotary actuator, a direct-drive spindle motor, the magnetic heads, the disks, and the carriage. A completely sealed self-contained airflow system is used within the DE to assure a clean environment for low-flying heads, thus ensuring very high reliability.

High Serviceability

The DE can easily be removed for replacement in the field. Serviceability is further improved by diagnostic information provided by the interface signals.

Low Maintenance

The MSU474 disk drive requires substantially reduced maintenance because of the completely sealed DE, a direct-drive DC Spindle motor, and highly reliable printed circuit boards.

Other Features

- A recording density of over 10^7 bits per square inch due to the advanced head and disk.
- Less than 0.62 KVA required despite large storage capacity.
- The dual channel feature, permitting two controllers to access the same disk drive so that a file can be shared by two different systems.
- Modified interface signals between the controller and the disk drive to agree with higher track capacity and high maintainability.

3. Specifications

The following tables list the specifications for the MSU474 Disk Drive.

Table 1 Specifications

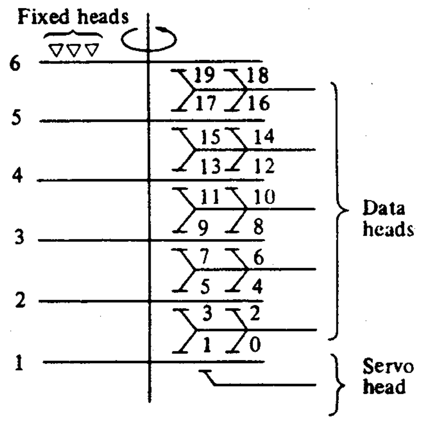
Capacity	/Drive (MB)	474.2 (Unformatted)
	/Track (KB)	28.160 (Unformatted)
Configuration of Disks and Heads		 Fixed heads 6 5 4 3 2 1 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 Data heads Servo head
Rotational Speed (RPM)		3,961
Latency (ms)		7.5
Disk	Diameter (Inch)	10.5
	Number	6
Heads	/Drive	20 + 1 (Servo)
	/Surface	2
Cylinders		842
Data Transfer Rate (MB/sec)		1.859
Positioning Time (ms)	Maximum	35
	Average	18
	Minimum	5
Track Density (TPI)		880
Bit Density (BPI)		12,800
Data Coding	on interface	NRZ
	on disk surface	MFM

Table 2 Environmental Requirements

Environment	Storage or transmit in packaged form		On Site Non-Operating	Operating Office Environment
Temperature*	Within 24 Hours	More than 24 Hours	23°F ∿ 140°F (-5°C ∿ 60°C) Max change 18°F/Hour (10°C/Hour)	50°F ∿ 104°F (10°C ∿ 40°C) Max change 18°F/Hour (10°C/Hour)
	-40°F ∿ 140°F (-40°C ∿ 60°C) Max change 36°F/Hour (20°C/Hour)	23°F ∿ 140°F (-5°C ∿ 60°C) Max change 18°F/Hour (10°C/Hour)		
Humidity	5% ∿ 95% RH Non-condensing		20% ∿ 80% RH Max change 10%/Hour Non-condensing	
Vibration	3G (When locked for shipment)		0.2G (10Hz ∿ 500Hz)	0.2G (5Hz ∿ 50Hz) 1G (50Hz ∿ 500Hz)
Shock	5G (Max. 30 ms)		3G (Max. 10ms)	2G (Max. 10ms)
Altitude	40,000 FT (12,000 m)		10,000 FT (3,000 m)	
Dust	0.168 mg/m^3 (Stearic Acid Standard)			
Air flow	——			2.5 m^3/min.
Acoustic	——			60 dBA

* $°C = \frac{5}{9}(°F-32)$

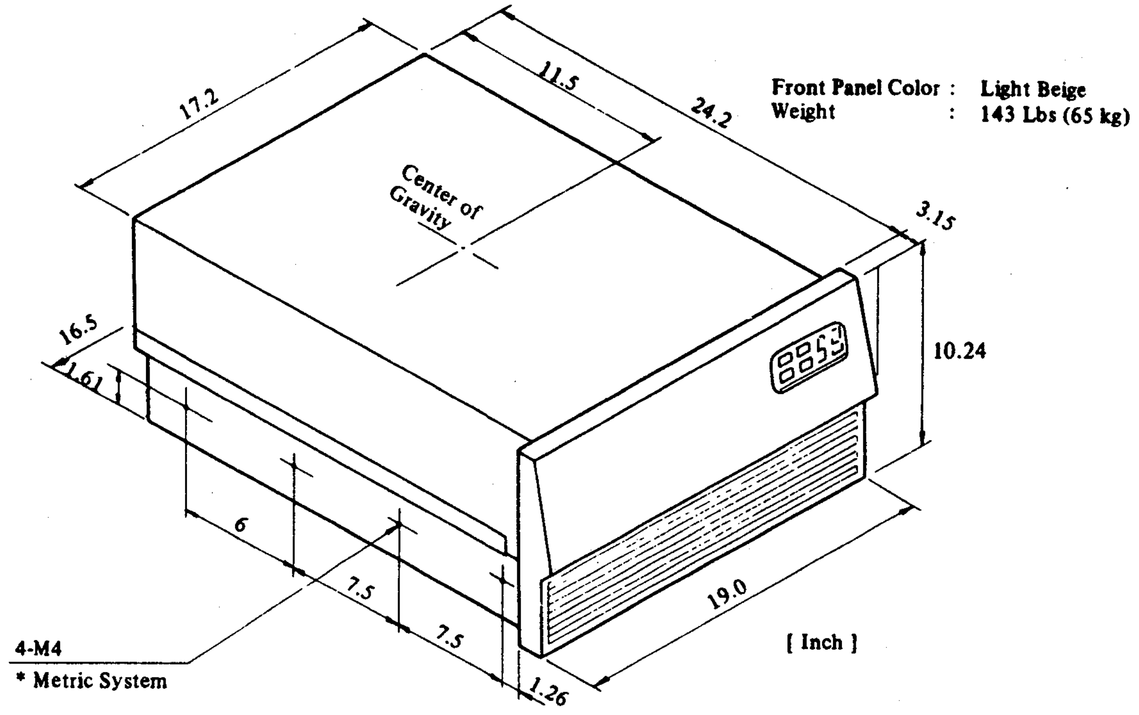
Table 3 Power Requirements

Voltage (Vac±10%)	Frequency (Hz±2Hz)	Current (Aac)		Power Consumption (KVA)	Heat Dissipation	
		Starting*	Running		(K•Cal/ Hour)	BTU/ Hour
100	50/60	5.3	5.7/5.4	0.57/0.54	460/420	1,800/1,600
120	60		4.6	0.55	430	1,700
220	50		2.8	0.62	510	2,000
240	50		2.6	0.62	510	2,000

*Worst case transient with a maximum of 40 amps for less than 1/2 cycle of input AC Power.

Table 4 Cables and Connectors

	Cable		Connector (Supplier)	
	Specification	Supplier	Drive Side	Cable Side
A-Cable (60-Pin)	Zo = 100±10Ω 28 AWG 7 Stands 100 FT. Max.	SPECTRA STRIP 455-248-60	FUJITSU FCN-704P060-AU/L	FUJITSU FCN-707J060-AU/B
			3M 3472-2303	3M 3334-6010
B-Cable (26-Pin)	Zo = 100±10Ω 28 AWG 7 Stands 50 FT. Max.	ANSLEY 174-26	FUJITSU FCN-705P026-AU/L	FUJITSU FCN-707J026-AU/B
			3M 3429-1303	3M 3399-6010



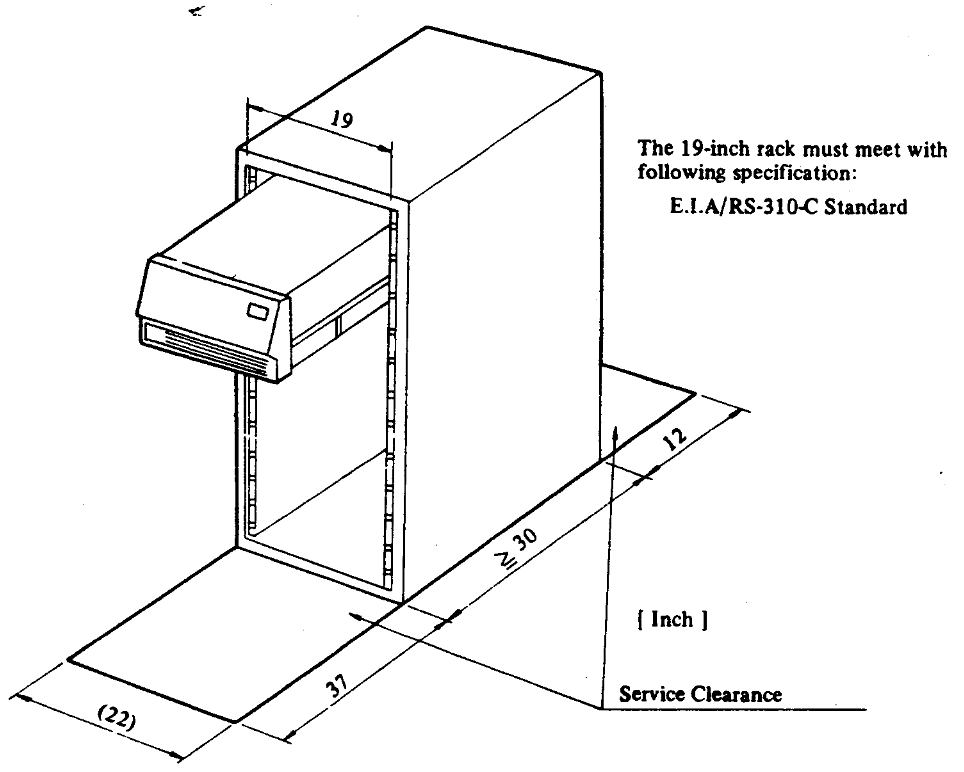


Figure 2. Physical Dimensions

4. General Operation

4.1 Switches and Controls

Switches and indicators on the operator panel are shown in the figure below.

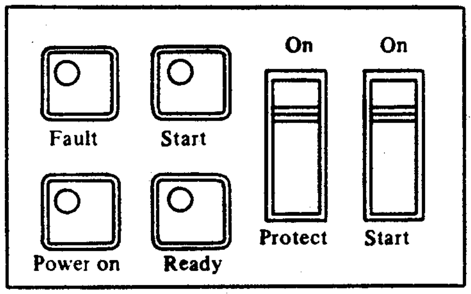


Figure 3. Operator Panel

Start/On Switch

Enables rotation of the spindle motor. The heads start an *Initial Seek* operation and stop on the cylinder zero approximatley 40 seconds later, lighting the *Ready* light. The spindle motor stops rotating approximately 15 seconds after the switch is set to the off position.

Protect/On Switch

Inhibits write operation. If a write command is issued from the controller while the PROTECT/ON switch is on, the *Fault* and *Control Check* conditions will be returned to the controller.

NOTE: The function of the PROTECT switch can occur with the selection of the drive. If the drive is selected before the switch is turned on, it must be deselected, then selected again to enable *File Select.*

Start (LED)

Light indicates that the spindle is rotating.

Ready (LED)

Light indicates that the spindle has reached the rated speed and no fault condition exists in the drive. It goes off when the heads are seeking the desired cylinder.

Fault (LED and Switch)

Indicates a *Fault* condition (i.e., R/W check status) or a *Seek Error.* Depressing the indicator switch clears this condition.

Power On (LED)

Light indicates that the DC power supply unit is on.

4.2 Power-up Procedure

To power up the MSU474:

- The main Lambda power supply should be ON; the *Power On* indicator light should be on.
- Press the START/ON switch on the disk front panel. The *Start* indicator should be on.
- Wait about 40 seconds for the disk to reach speed. The *ready* indicator should light, indicating that the disk is ready for use.

4.3 Power-down Procedure

To power down the MSU474:

- Leave LISP and/or UNIX gracefully. (See *The LMI Lambda Field Service Manual*, Section 6.18 and Section 6.19.)
- Switch the START/ON switch to the *off* position.

5. Error States

Table 5 lists the error states issued by the disk unit and control unit.

Table 5 Error Status

NOT READY	Not Ready status indicates disk drive is not ready
FAULT	Fault status indicates a fault condition has occurred in the unit
SEEK ERROR	Seek Error status indicates a seek error has occurred in seek operation
READ ERROR	Read Error status indicates a data error has occurred in read operation
AM MISSING	AM Missing status indicates that AM (Address Mark) has not found in read operation

Lambda Kermit User's Guide

Release 2.0: March 27, 1985

24-0100327-0001

Published by LMI 1000 Massachusetts Avenue. Cambridge MA 02138 USA

Information in this manual is current as of March 1985 for LMI Release 2.0.

Please help us to make LMI documentation work better for you! Send comments via your customer dialup mail line to Sarah Smith (username SWRS) or by U.S. Mail to:

Dr. Sarah Smith
Director, Documentation
LMI
1000 Massachusetts Avenue
Cambridge, MA 02138

Table of Contents

1. Introduction

The KERMIT file transfer protocol was developed at the Columbia University Center for Computing Activities, as were the first several KERMIT programs. Columbia has shared these programs freely with the worldwide computing community since 1981, and many individuals and institutions have contributed improvements or new implementations of the protocol. KERMIT implementations now support about sixty different systems.

Although KERMIT is free and available to anyone who requests it, it is not in "the public domain". The protocol, the manuals, the Columbia implementations, and many of the contributed implementations bear copyright notices dated 1981 or later, and include a legend like:

> Permission is granted to any individual or institution to copy or use this document and the programs described in it, except for explicitly commercial purposes.

This copyright notice is to protect KERMIT, Columbia University, and the various contributors from having their work usurped by others and sold as a product.

1.1 Restrictions on KERMIT

KERMIT may be passed along to others under the following conditions:

It is acceptable to charge a reproduction fee when supplying KERMIT to others. The reproduction fee may be designed to recover costs of media, packaging, printing, shipping, order processing, or any computer use required for reproduction. The fee should not reflect any program or documentation development effort, and it should be be independent of how many implementations of KERMIT appear on the medium or where they came from. It should not be viewed as a license fee. (Columbia charges $100.00 for a 2400 ft. reel of magnetic tape that includes all known versions of KERMIT, two printed manuals, various flyers, a box, and postage. There is an additional $100 order processing charge if an invoice must be sent.)

Commercial institutions may make unlimited internal use of KERMIT, and may incorporate KERMIT into their products with the following restrictions:

- A KERMIT program may not be sold as a product in and of itself. In addition to violating the prevailing spirit of sharing and cooperation, commercial sale of a product called "KERMIT" would violate the trademark held on that name by Henson Associates, Inc., creators of The Muppet Show.
- Existing KERMIT programs and documentation may be included with hardware or other software as part of a standard package, provided that the price of the hardware or software product is not raised significantly beyond costs of reproduction of the KERMIT component.

- KERMIT protocol may be included in a multi-protocol communication package as one of the communication options, or as a communication feature of some other kind of software package, in order to enhance the attractiveness of the package. KERMIT protocol file transfer and management should not be the primary purpose of the package. The price of the package should not be raised significantly because KERMIT was included, and the vendor's literature should make a statement to this effect.
- Credit for development of the KERMIT protocol should be given to the Columbia University Center for Computing Activities, and customers should be advised that KERMIT is available for many systems for only a nominal fee from Columbia and from various user group organizations, such as DECUS and SHARE.
- Columbia University holds the copyright on the KERMIT protocol, and may grant permission to any person or institution to develop a KERMIT program for any particular system. A commercial institution that intends to distribute KERMIT under the conditions listed above should be aware that other implementations of KERMIT for the same system may appear in the standard KERMIT distribution at any time. Columbia University encourages all developers of KERMIT software and documentation to contribute their work to Columbia for further distribution.
- Finally, Columbia University does not warrant in any way the KERMIT software nor the accuracy of any related documentation, and neither the authors of KERMIT programs or documentation nor Columbia University acknowledge any liability resulting from program or documentation errors.

Commercial organizations wishing to provide KERMIT to their customers should write a letter stating their plans and their agreement to comply with the guidelines listed above. The letter should be addressed to:

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

1.2 LMI Kermit

The LMI version of KERMIT allows communication and file transfer between the LAMBDA and other computers (including other LAMBDAs) via a standard RS-232 serial connection. KERMIT is normally used to transfer ASCII text files to and from various types of computers when more advanced technologies, such as those based on Ethernet, are not available.

This LMI version of KERMIT features a window interface, terminal emulation, a remote login server, and the ability to use the server facility provided by many timesharing implementations of the KERMIT protocol. The remote login server provides access to basic LAMBDA file manipulation commands and to a KERMIT server facility on many personal computers.

If you have trouble using this LMI version of KERMIT, don't hesitate to call the LMI Customer Assistance Line. The LMI CAL numbers are:

Outside Massachusetts:	1-800-872-LISP
Inside Massachusetts:	1-800-325-6115

1.3 More Information on KERMIT

This document is a brief explanation of the LMI version of KERMIT which runs on LAMBDAs. For more information on what KERMIT is and how it works, see the accompanying *Kermit User Guide* by the Columbia University Center for Computing Activities, in particular Chapter 2 on pages 7-16.

2. Connecting the Two Computers

The two machines must be connected by a cable or modem.

2.1 Cable Connections

Two machines at the same site may be connected by cable. To connect machines at the same site by cable, the industry standard RS-232 serial connection is best. To connect them, look on the back panel of the LAMBDA for Port A and Port B. Port A is male and is labelled "Data Terminal Equipment": Port B is female and labelled "Data Communication Equipment".

Connect serial Port B to the other computer via an RS-232 cable.

In many cases, it is necessary to make a special cable. Appendix A contains some guidelines for making one.

2.2 Modem Connections

Two machines in different sites may be connected by modems. Since most modems are DCE devices and the LAMBDA's serial port B is also DCE, you must switch lines 2 and 3 on the cable connecting the LAMBDA to the modem. . Lines 1 and 7 should go straight across.

NOTE: The LAMBDA's Remote Login Server facility does not support modem control lines of any kind. Therefore it will neither answer a call on a telephone line nor hang up afterwards. If you have a modem that can automatically answer and hang up phone calls, you can connect to a remote LAMBDA.

Some modems may be commanded to hang up with a sequence of characters. The following variable allows you to attempt to use such a modem.

s-terminal:*hangup-call* "a string" Variable

Defines the sequence of characters that will command a modem to hang up when given.

3. How to Start the KERMIT Program

To enter the KERMIT program, type ⟨SYSTEM⟩ K on the LAMBDA. This will return you to a previous KERMIT window, or create a new one if there is no previous one.

The KERMIT window consists of five window panes:

The Status Pane (upper left corner)
Displays the status of file transfers. It displays the number of packets of data that have been sent or received, along with the current state of the transfer process.

The Command Pane (upper right corner)
Displays a menu of mouse-sensitive KERMIT commands.

The HEATH Z29 Emulator Pane (middle of screen)
Used to communicate with the remote host: instead of going to the other machine's terminal and typing your input, you type it here. This window is in reverse video from the rest of the windows. (If the other computer can't be set up to interact with the emulator pane, you can still transfer files by typing commands on each machine's terminal.)

The Interaction Pane (bottom of screen)
This is where KERMIT prompts you for information (like filenames and pathnames), asks for confirmation before performing transfers, and states the success or failure of a transfer.

Abort, Exit, Break, and Resume Window
This window, along the bottom of the KERMIT frame, allows you to leave the KERMIT program. (In this version of KERMIT, it may have some bugs.)

To transfer files, you must have a version of KERMIT operating on the other machine also. So, after entering KERMIT on the LAMBDA, you must log into the other machine. To do this, click (L) `Connect` in the Command Menu. The local KERMIT will now switch you to the Z29 Terminal Emulator, from which you may communicate with the other machine. See the next chapter.

NOTE: If your two machines cannot communicate via the terminal emulator, you may still transfer files by typing the appropriate commands to the two terminals. To do this, set up KERMIT on the LAMBDA (but don't (L) `Connect`), and then go to the other machine, login, and start that machine's version of KERMIT. Set both machines to the same baud rate. You may now transfer files with or without server mode. Follow the directions in Chapter 5 of this document.

4. Using the Z29 Terminal Emulator

The LMI version of KERMIT contains a special window through which you can converse with the other machine. This is the Z29 terminal emulator, chosen because Z29s are a common type of terminal. The commands you will use from the terminal emulator to communicate with the remote machine are the ⟨NETWORK⟩ commands, found in Section 5.2 of this manual.

Both machines must be set at the same baud rate for the RS-232 port. To set the baud on the LAMBDA, click (L) [Set Baud Rate] in the Command Menu. A menu of baud rates from 50 to 19,200 will pop up; select one. The default is 9600.

To use the emulator, click (L) [Connect] on the Command Menu. You will now be connected to the other machine and should use the appropriate commands to login to it and set up its version of KERMIT.

The emulator recognizes most of the Z29 escape sequences and will do the appropriate action whenever it receives one. For example, see the following list for the LAMBDA keys which correspond to Z29 escape sequences:

For the Z29 CONTROL key, use the LAMBDA CTRLkey.

For the Z29 ESCAPE key, use the LAMBDA ⟨ALT MODE⟩.

For ESCAPE-X sequences on the Z29, use META-X keystrokes on the LAMBDA.

For the sequence CONTROL-Z-X on the Z29, use CTRL-META-X keystrokes on the LAMBDA.

When talking to an MIT EMACS-style editor (such as STEVE in VAX/NIL, and EMACS under TOPS-20) you can use the META and CONTROL keys just as you would in ZMACS on the LAMBDA.

Any ASCII characters you type will be sent to the other computer via the RS-232 link, and any characters received via the RS-232 will be displayed in the Z29 terminal emulator pane.

When you are finished with the emulator, click (L) [Disconnect] on the Command Menu or type ⟨NETWORK⟩-C (for close). The terminal emulator will shut down, closing the connection to the remote machine **without** logging you out from that machine, and the actual I/O stream will not be affected in any way. Both machines will still be running their version of KERMIT and you may transfer files. This is known as "escaping back" to KERMIT on the LAMBDA.

5. Overview of KERMIT Commands

The LMI version of KERMIT supports two command interfaces: the mouse/menu interface for *system* commands called at the local KERMIT, and the ⟨NETWORK⟩ key commands for use with the Terminal Emulator. The commands available to the mouse are those in the Command Menu. The ⟨NETWORK⟩ commands are called by giving ⟨NETWORK⟩ a single keystroke argument, a character key, for example. ⟨NETWORK⟩ supports some commands not available in the Command Menu.

Two commands work from either the mouse or ⟨NETWORK⟩: "Disconnect" and "Set baud rate."

Both sets of commands are described below.

5.1 The KERMIT Command Menu

The top right window, the command pane, contains the following commands for use from the local KERMIT:

Connect Establishes a virtual terminal connection with the remote host.

Disconnect
Closes the connection between the two machines that was made by the connect command.

Send files
Sends one or more files to a remote host.

Receive files
Receives one or more file from a remote KERMIT.

Server/send
Sends one or more files to a remote KERMIT in server mode.

Server/receive
Receives one or more files from a remote KERMIT in server mode.

Server/finish
Exits from server mode in a remote KERMIT, but does not log out.

Server/bye

Finishes with remote KERMIT in server mode and logs you out of the remote machine.

Set Baud Rate
Pops up a menu for selecting the serial baud rate of your RS-232 line.

Restart Program
Starts KERMIT from scratch, resetting all parameters to their default or initial value. A fresh serial stream will be allocated, resetting the associated hardware.

Review Parameters
Pops up a **choose-variable-values** menu to let you modify parameters such as the default local directory, the serial port device and the filename conversion mode.

Refresh windows
Refreshes all the windows in the display.

List Directory
Lists the files in the default local directory.

Help Accesses more detailed documentation on commands.

Remote Login Server
Enables the Remote Login Server to process remote logins and file transfers via the serial port device (for example, from a remote machine, you can login to a LAMBDA and perform file transfers).

Remote KERMIT Server
Same as Remote Login Server, but goes directly to KERMIT server mode without needing to log in. (Do `CTRL-⟨ABORT⟩ TO QUIT.`)

5.2 The NETWORK Commands

These commands are given as single-keystroke arguments to ⟨NETWORK⟩ and are used from the terminal emulator.

C (for Close)
Escapes back to KERMIT command level.

K (for Kill stream)
Sends current stream a :close message and disconnects.

⟨CLEAR SCREEN⟩
Clears the terminal emulator screen.

CTRL-⟨CLEAR SCREEN⟩
Clears the interaction pane (the bottom window).

F Flushes any unprocessed input in the serial input buffer.

CTRL-B **(for control baud)**
Sets the baud rate.

E Evaluates a LISP expression.

P (for Push)
Breaks to a LISP listener. Press ⟨RESUME⟩ to return to KERMIT.

B (for Break)
Transmits a break.

0 Transmits a null.

S or ⟨STATUS⟩
Shows serial stream status.

L Logs connection to a disk file (See paragraph below for more information).

CTRL-L Closes logging to a disk file.

Q Quits logging to a disk file temporarily.

R Resumes logging to a disk file.

?, ⟨HELP⟩**, or H**

Displays a table of ⟨NETWORK⟩commands.

The four "log" commands, "L, **CTRL-L**, Q, and R," allow you to keep track of everything displayed on the Z29 terminal emulator by putting it into a file. That is, after logging into a disk file, everything typed on the Z29 terminal emulator, will be saved into that file. A log is useful for monitoring your session and can also capture information from another computer that doesn't support the KERMIT protocol. For example, if you can communicate with another machine through the terminal emulator, you can ask it to "type" the contents of a file onscreen. As the contents are typed onscreen, they are also saved into the log file.

6. File Transfer

While there are many possible variations, KERMIT supports two main methods of file transfer. One method assumes that both the local and remote KERMIT provide a "server mode", the other method assumes neither KERMIT has a server mode. The server mode method is faster, easier, and more foolproof, but not all implementations of KERMIT provide it. Both methods are described below.

6.1 Sending Files Using Server Mode

NOTE: If your machines cannot communicate through the terminal emulator, you will login and start KERMIT on each machine separately, and then start at step number 4.

1. Connect to the other computer by clicking L `Connect` in the Command Menu.

2. Using the Z29 Terminal Emulator, login to the other computer.

3. Run the other computer's KERMIT program. Typically, this will mean typing **`Kermit`** to that computer's top level operating system. The KERMIT program should respond by printing a prompt, like KERMIT >. This means the other computer's KERMIT program is ready to accept input.

4. At the other computer's KERMIT prompt, type SERVER ⟨RETURN⟩. (It should print a message instructing you to return to your host computer and disconnect.)

5. Click L `Disconnect` on the LAMBDA Command Menu.

6. Click L `Server/send` on the LAMBDA Command Menu and then type the name of the LAMBDA file being sent. (If the file is not in the current directory, you must specify the pathname.) It will ask you to confirm that you want that file sent and give you the name it will be saved as on the other machine.

7. The file will be sent. The Status Pane will keep track of the packets as they are sent. The Interaction Pane will say something like *File sent successfully: LAM3: ROBERT; foo.lisp#2* when the operation is complete.

8. Click L `Server/finish` if you are finished sending and receiving files. This will close the connection with the remote computer without logging you out from it. To close the connection **and** logout from the remote computer, click L `Server/bye` instead.

NOTE: If you close the connection, you must reopen it again to transfer more files.

6.2 Receiving Files Using Server Mode

Before receiving files from another computer, you should review the parameters by clicking L `Review parameters` on the Command Menu. The three most important parameters are:

KERMIT-default-pathname
Specifies the LAMBDA directory into which the received file will be written.

File-Closing-Disposition
Specifies what the LAMBDA KERMIT program should do when an incomplete file transfer is aborted.

***Filnamcnv* (for "File Name Conversion")**
Sets up a file name conversion appropriate for the type of operating system on the other machine.

Here is the procedure for receiving files. Steps 1 through 5 are exactly the same as the procedure for sending files, described above. You need not repeat them if you are already connected to the other KERMIT's server (for example, if you have already sent some files).

6. Click L `Server/receive` on the LAMBDA KERMIT Command Menu. The LAMBDA will prompt you to type the name of the file that you want to receive as it is on the other computer.

7. The file on the other computer will now be sent to the LAMBDA and stored into the default directory on the LAMBDA. (You can change the default directory with `Review parameters` on the Command Menu.)

8. Click L `Server/finish` if you are finished sending and receiving files. This will close the connection with the remote computer without logging you out from it. To close the connection **and** logout from the remote computer, click L `Server/bye` instead.

6.3 Sending Files Without Using Server Mode

Steps 1 through 3 are the same as for sending files using server mode.

4. At the other computer's KERMIT prompt, type **`Receive`** RETURN.

5. Click L `Disconnect` on the LAMBDA Command Menu.

6. Click L `Send file` on the LAMBDA Command Menu. The LAMBDA KERMIT will then prompt you to type the name of the LAMBDA file you want to send.

7. The file will be transferred. KERMIT will inform you when the transfer is complete.

8. Click L `Connect` to connect back to the other computer. If you are finished transferring files, quit the other KERMIT, logout from the other computer, and click L `Disconnect`.

6.4 Receiving Files without Using Server Mode

Steps 1 through 3 are the same as for sending files using server mode.

4. At the other computer's KERMIT prompt, type **send** *filename* ⟨RETURN⟩.

5. Click L `Disconnect` on the LAMBDA's Command Menu.

6. Click L `Receive file` on the LAMBDA Command Menu. The LAMBDA KERMIT program will wait for the file to be received from the other computer.

7. The file will be transferred. KERMIT will inform you when the transfer is complete.

8. Click L `Connect` to connect back to the other computer. If you are finished transferring files, quit the other KERMIT, logout from the other computer, and click L `Disconnect`.

6.5 File Transfer Failure

File transfers can fail for various reasons. This version of KERMIT supports the transfer of only ASCII files. If you try to transfer LISP machine binary files (e.g. QFASL's), you will probably get checksum errors.

Files are sent in "packets" with single character checksums to provide accuracy. If a packet's checksum is inaccurate, the packet is resent. You may see warnings on your screen if this happens. It does NOT mean that your file failed to be transferred. If the transfer was correct, you will see this message on the interaction pane: "File transfered successfully: *filename* . . .".

If a packet does not get through, KERMIT may send it over again, up to approximately 10 tries.

"Timing out" is provided for. If for some reason the remote KERMIT is not getting data through to the LAMBDA KERMIT, it will know to quit waiting after a reasonable time (usually about 2 to 15 seconds).

6.6 Aborting File Transfers

If at any point you wish to abort a file transfer, type **CTRL-Z** instead of $\langle\boxed{\text{ABORT}}\rangle$. It may not take effect until the next time KERMIT samples the keyboard: if KERMIT is waiting for a packet (no more than 94 characters) of data, it won't sample the keyboard until the packet either finishes or times out. (This should take no more than about 20 seconds.) **CTRL-Z** is preferred over $\langle\boxed{\text{ABORT}}\rangle$, since it allows both the local and remote KERMITs to exit gracefully. However, if this fails or a drastic abort is needed, type **CTRL-**$\langle\boxed{\text{ABORT}}\rangle$.

7. Remote Login Server

Sometimes you may want to transfer files to and from a LAMBDA while actually working from another computer. The Remote Login Server allows you to log in to the LAMBDA over the connection. Once the two machines are connected, you should start up the other machine's version of KERMIT.

While in Remote Login Server mode, the LAMBDA waits to receive a character from the serial port. When it receives one, it responds by sending the prompt *Username:* back across the serial port. When the LAMBDA receives a username followed by a RETURN, it sends back the prompt *Password.* If the username and password sent back are valid, (or the LAMBDA password database is empty), the LAMBDA enters a command-interpreter-loop. This command interpreter is not a fully-functional loop; it provides only limited access to a few LAMBDA facilities.

The following variable and function allow you to set up a LAMBDA password database and then modify it. Once you have done this, you must use the correct password to login remotely to the LAMBDA.

s-terminal:*ps-kermit-login-passwords* nil Variable
An alist. This is the username password database.

It is settable as a site option (in the call to *defsite* in `SYS:SITE;SITE`) via the keyword `:KERMIT-LOGIN-ACCOUNTS`. Here is an example:

```
(defsite :x
 (:kermit-login-accounts (fred "password") (jim "failword") (jack
"noword"))
 ...)
```

s-terminal:add-ps-terminal-account *username password* Function
Adds or modifies the information in the password database.

Once you have logged in successfully to the LAMBDA through the remote login server, you must use the command interpreter's commands, instead of regular LAMBDA command syntax.

7.1 Command Interpreter Commands

When you have logged into a LAMBDA remotely, with the *remote login server*, you must use different commands from the regular LAMBDA commands. These are known as the command interpreter commands.

The command interpreter's prompt is `Kermit-Q>`. Commands should always be followed by a RETURN. Some commands take arguments. Commands which take arguments will prompt for them if they are not given on the command line. Here are the command interpreter commands:

cd	Changes (default) directory.
pwd	Prints (default) working directory.
Dir	Prints directory listing.
type	Prints a file on the screen.
eval	Reads, evaluates, and prints a LISP form.
help	Prints a listing of commands.
herald	Calls the LISP function *Print-Herald.*
time	Prints the current time.
logout	Goes back to mode of waiting for username/password.
server	Goes into KERMIT-server mode.

The most common way of transferring files is to **cd** to the directory with files you are interested in, call **dir** to list them, and then go into server mode to transfer these files to or from another LAMBDA or other computer.

Appendix A. Making a Special Cable

To make a cable for connecting the machines, use a shielded twisted pair for best results. In the following example, we will refer to the shield wire as "S", and the elements of the pair as "A" and "B". The cable in this example will work between a LAMBDA and a VAX:

For the LAMBDA, use a male DB-25 connector on Port B. For the VAX end, (which is "Data Terminal Equipment"), use a female DB-25 connector.

```
LAMBDA PORT-B (DCE)         VAX (DTE)
 pin - connection         connection - pin

   2 - A ......................... A -  2

   3 - B ......................... B -  3

   7 - S ......................... S -  7
     )  jumper              jumper  (
   1 -                               -  1

   4 -                               -  4
     )  jumper
   5 -                               -  5

   6 -                               -  6
     )  jumper
   8 -                               -  8
     )  jumper
  20 -                               - 20
```

CUCCA Kermit User Guide
Fifth Edition–July 1984

24-0100336-0002

Published by LMI 1000 Massachusetts Avenue. Cambridge MA 02138 USA

This manual was edited by Frank da Cruz and produced by the Columbia University Center for Computing Activities, New York, New York 10027.

TABLE OF CONTENTS

PREFACE TO THE 5TH EDITION, MARCH 1984

Since the 4th Edition of the *KERMIT Users Guide* was produced in July 1983, the KERMITs have been flying thicker and faster than anyone could keep up with. Old versions have improved, and implementations for many new systems have appeared. It is no longer practical to even attempt to cover all the implementations in a single manual. Therefore, this manual will try to describe a sort of "ideal" KERMIT program, one which has most of the features specified in the *KERMIT Protocol Manual*. Most real KERMIT programs will fall short of this description in some ways. After the main, system-independent part of the manual there are sections for several particular KERMIT programs, emphasizing their differences from the ideal, *at the time of this writing*. The system-dependent portions of this manual will rapidly become dated; current information about any particular KERMIT program can be found in the accompanying on-line help or documentation files, or built-in internal help text.

5th Edition, Revision 1, July 1984

The major sections of the manual are relatively unchanged. The chapters describing DECSYSTEM-20, MS-DOS, CP/M-86, CP/M-80, and Apple DOS Kermits have been updated to reflect new releases since last March. Meanwhile, a 2-part article describing the Kermit protocol was published in the June and July 1984 issues of BYTE Magazine, which is recommended reading for anyone who wants to know the reasons why a protocol like Kermit is necessary.

History and Acknowledgements

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

The initial objective was to allow users of our DEC-20 and IBM timesharing systems to archive their files on microcomputer floppy disks. The design owes much to the ANSI and ISO models, and ideas were borrowed from similar projects at Stanford University and the University of Utah. The protocol was designed to accommodate the "sensitive" communications front end of the full-duplex DEC-20 system as well as the peculiarities of half-duplex IBM mainframe communications. The protocol was soon implemented successfully on our IBM 4341 systems under VM/CMS by Daphne Tzoar of CUCCA.

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

After our initial success with KERMIT, we presented it at conferences of user groups like DECUS and SHARE, and we began to get requests for it from other sites. Since we had written down a description of the protocol, some sites wrote their own implementations for new computers, or adapted one of our implementations to run on additional systems, and sent back these new versions to us so that we could share them with others. In this way, KERMIT has grown to support about 50 different systems; it has been sent on magnetic tape from Columbia to hundreds of sites in dozens of countries, and has reached hundreds or thousands more through various user groups and networks.

To date, contributions to the KERMIT effort have been made by individuals at the following institutions: Stevens Institute of Technology, Cornell University, the University of Chicago, Rutgers University, Cerritos College, the University of Toronto, the University of Tennessee at Knoxville, the University of California at Berkeley, the University of Toledo, the University of Texas at Austin, the University of Michigan, Oakland University, the University of Wisconsin, University College Dublin, the University of Washington, ABC-Klubben Stockholm, the Helsinki University of Technology, the US National Institutes of Health, Digital Equipment Corporation, The SOURCE Telecomputing, SPSS Inc, Hewlett-Packard Laboratories, Litton Data Systems, RCA Laboratories, Atari Computer, and others. The list grows constantly.

The Kermit protocol was named after Kermit the Frog, start of the television series *THE MUPPET SHOW*, and is used by permission of Henson Associates, Inc.

Customizing This Manual

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

Each site, upon receiving a KERMIT tape, may decide which versions of KERMIT are important to it, and include the appropriate documentation in this manual. This is most conveniently done if your site has the Scribe text formatting system (from UNILOGIC Ltd in Pittsburgh PA, USA), with which this manual was produced. Scribe runs on a wide variety of systems. There are also Scribe subsets, such as Perfect Writer and Final Word, that run on various microcomputers.

The system-specific parts of the KERMIT User Guide are included with "**@INCLUDE**" statements at the end of the Scribe source file for this manual, whose filename is **USER.MSS**. You may add or delete **@INCLUDE** statements to suit your needs, and run the result through the text formatter to produce a customized manual.

Not all system-specific documentation is provided in **.MSS** (Scribe input) format, since some KERMIT contributors do not have Scribe at their sites. In that case, you will either have to add Scribe formatting commands, or else enclose the whole subfile in **@VERBATIM** brackets

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

If you are running a version of KERMIT for which adequate documentation has not been provided (after all, this is a distributed, volunteer effort!), please feel free to write some, preferably in Scribe input format, and send it back to Columbia so that others may benefit from it. Likewise if you produce a new implementation of KERMIT.

ORDERING INFORMATION

The KERMIT software is free and available to all. Columbia University, however, cannot afford to distribute free software on the scale required for KERMIT. Therefore, to defray our costs for media, printing, postage, materials, labor, and computing resources, we must request a moderate distribution fee from sites that request KERMIT directly from Columbia. The schedule is as follows:

Complete KERMIT Distribution *(Tape, Users Guide, and Protocol Manual)*	$100.00
Printed Documents *(Users Guide, Protocol Manual, or Any Source Listing)*	$5.00 each

Other sites are free to redistribute KERMIT on their own terms, and are encouraged to do so, with the following stipulations: KERMIT should not be sold for profit; credit should be given where it is due; and new material should be sent back to Columbia University at the address below so that we can maintain a definitive and comprehensive set of KERMIT implementations for further distribution.

To order KERMIT from Columbia University, send a letter requesting either:

(a) The manuals or source listings you desire (specify each one), *or*

(b) A 9-track magnetic tape in one of the following formats:

System	Tape Format	Densities
TOPS-10	BACKUP/Interchange, Unlabeled	1600
TOPS-20	DUMPER, Unlabeled	1600
IBM VM/CMS	EBCDIC, CMS Format	1600, 6250
	or EBCDIC, OS Standard Label	1600, 6250
UNIX	TAR	1600
Other	ASCII, ANSI Label, Format "D"	1600

(Specify system, format, and density.) The "Kermit collection" has grown so large that we can no longer fit it on a 2400 reel of magnetic tape at 800bpi. One copy of each manual will be included with the tape. We will supply the tape, packaging, and postage.

We can only make tapes in the formats listed above. <u>We cannot produce floppy disks</u>; bootstrapping procedures are provided to allow the microcomputer versions to be downloaded from the mainframe for which the tape is produced. The tape includes all source programs, documentation, and, when practical, binaries or hex. Unfortunately, our limited resources to not allow us to provide automatic updates to KERMIT recipients when new implementations, documentation, or bug fixes appear.

Send your letter to:

KERMIT Distribution
Columbia University Center for Computing Activities
7th Floor, Watson Laboratory
612 West 115th Street
New York, N.Y. 10025

Please list the machines and operating systems you expect to run KERMIT on, specify the tape format or the listings desired, and mention whether there are additional systems for which you require KERMIT or if you might be interested in attempting your own implementation for a new system. Make checks payable to *Columbia University Center for Computing Activities.*

KERMIT is available to users of the BITNET network via a server at host CUVMA. BITNET users may type "SMSG RSCS MSG CUVMA KERMSRV HELP" for further information. KERMIT is also available to users of ARPANET, via anonymous FTP from host COLUMBIA-20, in the area `PS:<KERMIT>`. And KERMIT is distributed regularly by various computer user groups such as DECUS and SHARE.

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

No warranty of the software nor of the accuracy of the documentation surrounding it is expressed or implied, and neither the authors nor Columbia University acknowledge any liability resulting from program or documentation errors.

1. INTRODUCTION

Everyone wants to get computers talking to one another. There are many ways to do this, and most of them are very expensive. But there is one way that is cheap and relatively easy: connect the two computers through their terminal (TTY) ports, tricking one computer (or both) into believing that the other is a terminal. This can be expected to work because the standard for connecting computers to terminals is almost universally followed, in both hardware (plug and signal: EIA RS-232) and software (character code: ASCII). Once two computers are connected in this way, cooperating programs can be run on each to achieve the desired communication by means of a communication *protocol*.

Why is a protocol necessary at all? Three major problems occur when you try to connect two computers via TTY line:

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

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

3. *Line Outages* -- A line may stop working for short periods because of a faulty connector, loss of power, or similar reason. On dialup or switched connections, such intermittent failures will cause carrier to drop and the connection to be closed, but for any connection in which the carrier signal is not used, the symptom will be lost data

To prevent corruption of data and to synchronize communication, cooperating computers can send control information to one another at the same time that they are transferring data. This intermingling of control information with data, and the resulting actions constitute a "protocol".

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

KERMIT transfers data by encapsulating it in "packets" of control information. This information includes a synchronization marker, a packet number to allow detection of lost packets, a length indicator, and a "checksum" to allow verification of the data. Lost or corrupt packets are detected, and retransmission is requested. Duplicated packets are discarded. In addition, various special control packets allow cooperating KERMITs to connect and disconnect from each other and to exchange various kinds of information. Very few assumptions are made about the capabilities of either computer, so the KERMIT protocol can work between many different kinds of systems.

Organization of This Manual

Section 2, *How to Use KERMIT*, tells all you need to know to transfer text files in most cases, and shows some specific examples.

If you follow the examples in Section 2 but you can't make a terminal connection or you can't transfer files successfully, consult Section 3, *When Things Go Wrong*.

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

Section 5, *KERMIT Implementations*, briefly lists the systems for which KERMIT is available *as of this writing*. The subsequent chapters describe selected particular implementations. You should read the appropriate section for each system with which you are using KERMIT; each section describes the file naming conventions and other system features that are important to KERMIT users, and lists the KERMIT commands for that system mainly in terms of their differences from the "ideal" KERMIT described in section 4.

2. HOW TO USE KERMIT

KERMIT is a protocol for reliable file transfer between computers over the ordinary serial telecommunication lines that are used to connect terminals to computers. The mechanics of using KERMIT to get a file transferred can be confusing until you get the hang of it. A little background material might make the process a bit easier to understand.

KERMIT is probably the cheapest way to put two computers into communication. The required hardware is usually already available, the software is free, and all components run as ordinary user programs, with no system modifications. This is in sharp contrast to a communication network, where there are dedicated high-speed communications channels and drivers, expensive software, and so forth. The network provides more services than KERMIT, usually at higher speed, and with greater convenience, because the network is usually part of the system. When a network is not available, KERMIT can fill in. But since KERMIT is not integrated with any particular system, but rather grafted on top of many different systems, it requires some extra work from those who use it.

2.1. The KERMIT Program

KERMIT embodies a set of rules for transferring files reliably between computers. In general, one computer is a large system (a *host*, for instance a timesharing system with many terminals), and the other is a personal computer (*PC*)[1]. The host believes that the PC is an ordinary terminal. In order for the KERMIT protocol to occur, a KERMIT *program* must be running on each end of the communication line -- one on the host, one on the PC.

The two Kermit programs exchange messages in a special language all their own, the *Kermit protocol*. The dialog runs something like, "Hi! I'm going to be sending files to you. When you send messages to me, please don't make them more than 80 characters long, and if you don't hear anything from me for 15 seconds, wake me up, OK?" *"OK."* "Now, here comes a file called FOO.TXT, OK?" *"OK."* "Here's the first piece..." *"Got it."* "Good, here's the second piece..." *"That second piece was junk."* "Well, then here it is again..." Et cetera. You don't see any of this. It's all packed into a concise code which the two Kermits can understand; they do all the worrying about transmission, error checking, character set translation, and so forth. Each message is called a *packet*, and each packet is in a special format that all Kermits can understand.

2.2. Talking to Two Computers at Once

Your task is just to get the two Kermits started. The confusion arises because you have to use a single keyboard and screen to talk to two different computers, two different programs. Let's talk about a common case: you are sitting at a personal computer (PC[2]), which has a serial communication port. The serial port is connected to a host computer using, say, a dialup modem[3].

[1] Host-to-host and PC-to-PC connections are also possible.

[2] The terms PC, micro, microcomputer, and workstation will all be used loosely in this document to denote a single-user system.

[3] The actual means of connection isn't important in this case -- it also could be a direct line to the host, some kind of switched line, etc.

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

Kermit, like most programs, has a *prompt*. The prompt is a symbol it types on the left margin to indicate that it is ready for you to type a command. Kermit's prompt is normally "`Kermit-`*xx*`>`". The *xx* identifies the implementation of Kermit; the Kermit that runs on the DEC-20 is called "Kermit-20" and its prompt is "`Kermit-20>`"; the Kermit that runs on Z80 and 8080-based microcomputers is called "Kermit-80" and its prompt is "`Kermit-80>`"; the Kermit on the IBM PC is "Kermit-86"[4], and so forth. If you become confused about who you are talking to, the prompt should provide a clue. In addition, most Kermits print an informative message like

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

when you CONNECT, and type another message like

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

when you return.

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

2.3. Transferring a File

To transfer a file, you must first get the attention of the PC's operating system. This is normally done by starting the PC, possibly inserting your system floppy disk first. Once you're at command level on your PC, you run Kermit. Then you tell Kermit to CONNECT you to the host. Now you're talking to the host -- at this point you must log in, and then run Kermit on the host.

Now you have a Kermit on each end of the wire. The next step is to tell *each* Kermit what to do. Suppose you want to transfer a file from the host to the PC; you would first tell the host Kermit to SEND the file, then "escape" back to the PC Kermit and tell it to receive the file. The transfer begins -- you can sit back and watch, or go make yourself a sandwich. The PC Kermit will continuously show packet and retry counts on your screen, and will notify you when the transfer is complete.

---

[4] Although the processor in the IBM PC is an 8088, it is programmed as though it were an 8086.

The desired file is now on your PC's disk. The Kermit protocol has ensured that the file arrived correctly and completely. Now you must clean up after yourself: CONNECT back to the host, exit from Kermit on the host, log out from the host, "escape" back to PC Kermit and exit from it. Now you can do whatever you had planned for your file -- edit it, print it on your PC printer, etc.

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

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

Now that we have a basic understanding of what KERMIT does and how it works, let's look at some more concrete examples. First you need to know what the basic Kermit commands are.

2.4. Basic KERMIT Commands

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

SEND *filespec*

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

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

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

RECEIVE

Receive a file or file group from the other Kermit. If an incoming file name is not legal, then attempt to transform it to a similar legal name, e.g. by deleting illegal or excessive characters. The name thus formed cannot be guaranteed to be unique, in which case previously existing files could be overwritten. Some versions of Kermit attempt to prevent this by warning you of filename collisions and taking, or allowing for, evasive action.

CONNECT

Make a "virtual terminal" connection to the remote system. On a PC or micro, this usually means to send all keyboard input out the serial port, and display all input from the serial port on the screen. To

	"escape" from a virtual terminal connection, type Kermit's *escape character* (e.g. CTRL-], control-rightbracket), followed by the letter "C" for "Close Connection".
SET	Establish various nonstandard settings, such as CONNECT escape character, file characteristics, communication line number, parity, or flow control.
SHOW	Display the values of SET options.
HELP	Type a summary of KERMIT commands and what they do.
EXIT	Exit from KERMIT back to the host operating system.
?	Typed anywhere within a KERMIT command: List the commands, options, or operands that are possible at this point. This command may or may not require a carriage return, depending on the host operating system.

2.5. Real Examples

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

2.5.1. PC to Host

In this example, the user is sitting at an IBM Personal Computer (PC), which is connected through its serial port to a DECSYSTEM-20 host computer. The IBM PC is *local*, the DEC-20 is *remote*. This example will also apply almost literally to any other microcomputer implementation of Kermit.

You have started up your PC and have the Kermit program on your disk. Begin by running Kermit on the PC. Use Kermit's CONNECT command to become a terminal to the DEC-20. In fact, the PC emulates the popular Heath-19 (or VT52) terminal, so it is desirable to tell the DEC-20 that your terminal is one of these. Login on the DEC-20 and run Kermit there. Here is an example of this procedure with commands you type underlined:

```
A>kermit                       ! Run Kermit on the PC.[5]
Kermit V1.20

Kermit-86>                     ! This is the Kermit prompt for the PC.
Kermit-86>connect              ! Connect to the DEC-20.
[Connecting to host, type control-] to return to PC.
Baud rate is 9600, connecting over COM1.]

                               ! You are now connected to the DEC-20.
CU20B                          ! The system prints its herald.
@terminal heath-19             ! Set your terminal type (optional).
@login my-id password          ! Login using normal login method.
```

(At this point, the DEC-20 prints various messages.)

```
@kermit                        ! Run Kermit on the DEC-20.
Kermit-20>                     ! This is Kermit-20's prompt.
```

[5] Everthing from a "!" mark to the end of line is commentary, not system typeout or part of a command.

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

The following example illustrates how to send files from the DEC-20 to the PC. Note the use of the "*" *wildcard* character to denote a *file group*.

```
Kermit-20>send *.for          ! Send all my FORTRAN files.
^]c                           ! Now return back to the PC by
                              ! typing the escape sequence, in this case
                              ! ^]C (Control-] followed by "C")
[Back at PC.]
Kermit-86>receive             ! Tell the PC files are coming.
```

If you take more than about 5 seconds to get back to Kermit-86 and issue the RECEIVE command, the first packets from Kermit-20 may arrive prematurely and appear on your screen, but no harm will be done because the packet will be retransmitted by the DEC-20 until the PC acknowledges it.

Once the connection is established, the PC will show you what is happening -- it first clears the screen and waits for incoming packets; as packets arrive, the current file name and packet number will be continuously displayed on the screen. When the PC's "Kermit-86>" prompt returns to your screen, the transfer is done. During file transfer, the microcomputer screen looks something like this:

```
                    IBM PC Kermit-86 V1.20

Number of Packets:  294                     Receiving...
Number of Retries:    2
File Name:    FOO.TXT
```

The packet and retry counts are continuously updated, and the word in the upper right tells the status of the transfer -- receiving, sending, complete, interrupted, or failed.

When the transfer is complete (most versions of KERMIT sound a beep to wake you up), you must CONNECT back to the DEC-20 host, EXIT from Kermit there, logout, and "escape back" to the PC as you did previously.

```
Kermit-86>connect             ! Get back to the DEC-20.
[Connecting to host. Type CTRL-]C to return to PC.]
Kermit-20>                    ! Here we are.
Kermit-20>exit                ! Get out of Kermit-20.
@logout                       ! Logout from the DEC-20.

Logged out Job 55, User MY-ID, Account MY-ACCOUNT, TTY 146,
  at 24-Jan-84 15:18:56,  Used 0:00:17 in 0:21:55

^]c                           ! Now "escape" back to the PC,
[Back at PC.]
Kermit-86>exit                ! and exit from the PC's Kermit.
```

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

To send files from the PC to the DEC-20, follow a similar procedure. First follow the instructions in the previous section to log in to the DEC-20 through the PC. Then in response to the host Kermit's "Kermit-20>" prompt you type RECEIVE rather than SEND. Now escape back to the PC and use the SEND command to send the local PC files to DEC-20 host. The PC will show you the progress of the transmission on its screen.

When the "Kermit-86>" prompt indicates that the transmission is complete you should follow the procedure shown above to logout from the DEC-20 host, except that you may first wish to confirm that the files have been stored correctly in your directory on the DEC-20.

2.5.2. Host to Host

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

In this example, you are already logged in to a DEC-20, and you use an *autodialer* to connect to an IBM 370-series system running VM/CMS through DEC-20 TTY port 12. The autodialer, in this example, is invoked from program called DIAL (idealized here, for simplicity), to which you merely supply the phone number.

```
@dial 765-4321/baud:1200
765-4321, baud 1200
[confirm]
Dialing your number, please hold...
Your party waiting is on TTY12:
@
```

Other methods exist for connecting two hosts with a serial line. Dedicated hookups can be made simply by running an EIA cable between TTY ports on the two systems.[6] For connecting to remote systems when no autodialer is available, a manual dialup connection is also possible, but tricky.[7] If you have a microcomputer that supports KERMIT, you may find it easier to first transfer from host A to the micro, then from the micro to host B.

The following procedure would be the same in any case, once a connection is made.

```
@
@kermit                          ! Run Kermit on the DEC-20.
Kermit-20>set ibm                ! Turn on handshaking, parity, local echo.
Kermit-20>set line (to tty) 12 ! Indicate the line we'll use.
Kermit-20>connect                ! And connect to it.
[KERMIT-20: Connecting over TTY12:, type <CTRL-Y>C to return.]

VM/370 ONLINE                    ! The IBM system prints its herald.

.login myuserid mypassword      ! Login to IBM system.

LOGON AT 20:49:21 EST THURSDAY 01/20/84
CUVMB SP/CMS PUT 8210 01/19/84
.
.kermit
KERMIT-CMS>.send profile exec ! Send a file.
^Yc                             ! KERMIT-20's escape sequence typed here.
[KERMIT-20: Connection Closed. Back at DEC-20.]
Kermit-20>receive               ! Tell Kermit-20 to RECEIVE.
```

[6] Such a connection, by the way, usually requires the receive and transmit leads (pins 2 and 3) be swapped in one of the RS-232 connectors; this is called a "null modem" cable.

[7] Here's one way: log in on port x on your system, and assign another port, y, to which you have physical access. Unplug the terminal from port y, and connect the terminal to a dialup modem. Dial up the remote computer and log in on it. Now, using a null modem cable, connect the modem directly to port y. Go back to your terminal on port x, run Kermit from it, and CONNECT to port y.

The transfer takes place now; Kermit-20 will print the names of incoming files, followed by dots or percents to indicate the packet traffic (a dot for every 5 packets successfully transferred, a percent for every timeout or retransmission). It is complete when when you see "[OK]", a beep is sounded, and the Kermit-20 prompt next appears. At that point we connect back to the remote IBM system, exit from the remote Kermit and log out.

```
.
PROFILE.EXEC.1 ..%%.[OK]
Kermit-20>connect            ! Get back to IBM and clean up.
[KERMIT-20: Connecting over TTY12:, type <CTRL-Y>C to return.]

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

CONNECT= 00:03:01 VIRTCPU= 000:00.12 TOTCPU= 000:00.60
LOGOFF AT 20:52:24 EST THURSDAY 01/20/84

^Yc                          ! Type Kermit-20's escape sequence
[KERMIT-20: Connection Closed. Back at DEC-20.]
Kermit-20>exit               ! All done with Kermit.
```

That's the whole procedure. The file is in your DEC-20 directory, completely readable, as PROFILE.EXEC -- note that KERMIT-CMS translated from the IBM EBCDIC character encoding into standard ASCII, and converted the space between the file name and file type to a dot.

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

2.5.3. Micro to Micro

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

Making the physical connection between two micros is tricky. If the two units are in close proximity[8], you can connect their serial ports with a null modem cable. However, different micros have different requirements -- some may want a male connector on their serial port, others a female; many require that certain of the RS-232 signals be held high or low[9]. In any case, you must also make sure the port speeds are the same at both ends.

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

[8] Why would you want to run Kermit between two PCs that are next to each other? One good reason is that if they are different models, their floppy disks are probably incompatible.

[9] By wiring certain of the pins in the connector together; for instance, some micros want DTR (Data Terminal Ready, pin 20) to be held high, and this might be accomplished by connecting it to CTS (Clear To Send, pin 5). See EIA Standard RS-232-C, and the appropriate manuals for your micro.

In this example, a DEC VT180 "Robin" CP/M microcomputer is connected to a Intertec "SuperBrain" CP/M micro, using a female-to-male null modem cable. Getting the cable right is the hard part. The connection can be tested by running Kermit and issuing the CONNECT command on both ends: typein from each micro should appear on the screen of the other.

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

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

```
A>kermit
Intertec SuperBrain Kermit-80 - V3.7
Kermit-80>receive
```

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

```
A>kermit
DEC VT18X Kermit-80 - V3.7
Kermit-80>send foo.hex
```

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

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

2.6. Another Way -- The KERMIT Server

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

Not all implementations of Kermit can be servers, and not all know how to talk to servers -- but most of the major ones can and do. The server is run on the remote computer, which would normally be a large host, such as the DEC-20. You must still connect to the remote host to log in and start the server, but you no longer have to tell one side to SEND and the other to RECEIVE, nor must you connect back to the remote side to clean up and log out when you're done. Using the server, you can do as many send and receive operations as you like without ever having to connect back to the remote host. Some servers also provide additional services, such as directory listings, file deletion, or disk usage inquiries.

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

Here's an example of the use of a Kermit server. The user is sitting at a CP/M-80 microcomputer and a DEC-20 is the remote host.

```
A>kermit                    ! Run Kermit on the micro.
Kermit V3.9A

Kermit-80>                  ! This is the micro Kermit's prompt.
Kermit-80>connect           ! Connect to the DEC-20.
[Connecting to remote host.  Type CTRL-]C to return to micro.]

CU20E                       ! The DEC-20 prints its herald.
@login my-id password       ! Log in normally.
```

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

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

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

^]c                         ! Now escape back to the micro.
[Connection closed, back at micro.]
Kermit-80>get *.pas         ! Get all my DEC-20 Pascal programs.
Kermit-80>send foo.*        ! Send all the "foo" files from my micro.
Kermit-80>exit              ! Exit from Kermit back to CP/M.
A>
```

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

```
A>kermit                    ! Run Kermit-80 some more.
Kermit-80>send file.pas     ! Send another file.
Kermit-80>bye               ! That's all.  Shut down the Kermit server.
A>                          ! Back at CP/M automatically.
```

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

Here are basic the commands available for talking to servers.

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

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

```
get *.c
```

This command is exactly equivalent to typing "send *.c" at the remote host followed by "receive" on the local host. Note that the local Kermit does not attempt to validate the filespec. If the server cannot parse it, or cannot access the specified file(s), it will send back an appropriate error message.

BYE — Shut down the remote server and exit from Kermit. This will cause the job at the remote end to log itself out. You need not connect back and clean up unless you get an error message in response to this command (for instance, if your logged-out disk quota is exceeded on the remote host).

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

3. WHEN THINGS GO WRONG

Connecting two computers can be a tricky business, and many things can go wrong. Before you can transfer files at all, you must first establish terminal communication. But successful terminal connection does not necessarily mean that file transfer will also work. And even when file transfer seems to be working, things can happen to ruin it.

3.1. Communication Line Problems

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

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

You must consult the appropriate manuals for the systems and equipment in question.

If all settings and connections appear to be correct, and communication still does not take place the fault may be in your modem. Internal modems (i.e. those that plug in to a slot inside the microcomputer chassis) are *not* recommended for use with KERMIT. Many microcomputer KERMIT programs are written to control the communication hardware explicitly; internal modems can interfere with that control.

KERMIT normally expects to have full control of the communication port. However, it is sometimes the case that some communications equipment controls the line between the two computers on either end. Examples include modems (particularly "smart" modems) port contention or selection units, multiplexers, local networks, and wide-area networks. Such equipment can interfere with the KERMIT file transfer protocol in various ways:

- It can impose *parity* upon the communication line. This means that the 8th bit of each character is used by the equipment to check for correct transmission. Use of parity will:
  - Cause packet checksums to appear incorrect to the receiver and foil any attempt at file transfer. In most cases, not even the first packet will get through.
  - Prevent the use of the 8th bit for binary file data.

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

  - Compose and interpret the checksums correctly.

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

Many packet-switched networks, such as GTE TELENET, require parity to be set.[10]

- Communications equipment can also interpret certain characters in the data stream as commands rather than passing them along to the other side. For instance, you might find your "smart" modem suddenly disconnecting you and placing a call to Tasmania. The only way to work around such problems is to put the device into "transparent" or "binary" mode. Most communication devices have a way to do this; consult the appropriate manual. In some cases, transparent mode will also cancel the parity processing and allow the use of the 8th bit for data.

3.2. The Transfer is Stuck

There are various ways in which Kermit file transfers can become stuck, but since many hosts are capable of generating timeout interrupts when input doesn't appear quickly enough, they can usually resend or "NAK" (negatively acknowledge) lost packets. Nevertheless, if a transfer seems to be stuck, you can type RETURN on the keyboard of most micros to simulate a timeout.

An interesting exception is the IBM mainframe (VM/CMS) Kermit -- it cannot time out its "virtual console" (i.e. the user's terminal), so when using Kermit from a micro to an IBM host, occasional manual wakeups may be necessary.

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

3.3. The Micro is Hung

The micro itself sometimes becomes hung for reasons beyond Kermit's control, such as power fluctuations. If the micro's screen has not been updated for a long time, then the micro may be hung. Try these steps (in the following order):

- Check the connection. Make sure no connectors have wiggled loose from their sockets. If you're using a modem, make sure you still have a carrier signal. Reestablish your connection if you have to.
- Press RETURN to wake the micro up. This should clear up any protocol deadlock. Several RETURNs might be necessary.
- If the problem was not a deadlock, restart the micro and then restart Kermit, CONNECT back to the host, get back to your job or login again, and restart the transfer. You may have to stop and restart Kermit on the remote host.

[10] TELENET uses MARK parity.

3.4. The Remote Host Went Away

If your local system is working but the transfer is hung, maybe the remote host or the remote KERMIT program crashed. Get back to command level on the local KERMIT (on microcomputer implementations, you may be able to do this by typing about five RETURNs, or one or more Control-C's). Issue the CONNECT command so that you can see what happened. If the remote system has crashed then you will have to wait for it to come back, and restart whatever file that was being transferred at the time.

3.5. The Disk is Full

If your local floppy disk or remote directory fills up, the Kermit on the machine where this occurs will inform you and then terminate the transfer. You can continue the transfer by repeating the whole procedure either with a fresh floppy or after cleaning up your directory. Some KERMIT programs allow you to continue the sequence where it left off, for instance on the DEC-20 by using the SEND command and including the name of the file that failed in the "(INITIAL)" field:

```
     Kermit-20>send *.for (initial) foo.for
```

See the Kermit-20 command summary for further information about the initial filespec.

3.6. Message Interference

You may find that file transfers fail occasionally and upredictably. One explanation could be that terminal messages are being mixed with your file packet data. These could include system broadcast messages (like "System is going down in 30 minutes"), messages from other users ("Hi Fred, what's that KERMIT program you're always running?"), notifications that you have requested ("It's 7:30, go home!" or "You have mail from..."). Most KERMIT programs attempt to disable intrusive messages automatically, but not all can be guaranteed to do so. It may be necessary for you to "turn off" such messages before starting KERMIT.

3.7. Host Errors

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

3.8. File is Garbage

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

3.9. Junk after End of File

When transferring a text file from a microcomputer to a mainframe, sometimes you will find extraneous characters at the end of the file after it arrives on the target system. This is because many microcomputers don't have a consistent way of indicating the end of a file. CP/M is a good example. The minimum unit of storage on a CP/M floppy is a "block" of 128 bytes. Binary files always consist of a whole number of blocks, but a text file can end anywhere within a block. Since CP/M does not record a file's byte count, it uses the convention of marking the end with an imbedded Control-Z character. If your microcomputer version of KERMIT is not looking for this character, it will send the entire last block, which may contain arbitrary junk after the "real" end of the file. To circumvent this problem, most microcomputer KERMITs have commands like SET FILE ASCII or SET FILE TEXT to instruct KERMIT to obey the CTRL-Z convention. Some microcomputer KERMITs operate in "text" mode by default, others in "binary" or "block" mode.

4. KERMIT COMMANDS

An "ideal" KERMIT program will be described here, which has most of the features specified in the *KERMIT Protocol Manual*. No KERMIT program will have all these commands or support all these options. The exact form of some of the commands may differ from version to version. Some KERMIT programs may support system-dependent options not described here. The intention of this description is to provide a base from which specific KERMIT programs can be described in terms of their differences from the "ideal."

4.1. Remote and Local Operation

Some KERMIT programs can be run in two ways, *remote* and *local*. A remote Kermit is usually running on a mainframe, which you have CONNECTed to through a PC or other computer. When KERMIT runs remotely, all file transfer is done over the job's controlling terminal line -- the same line over which you logged in, and to which you would type interactive commands. What the system thinks is your terminal is really another computer, usually a microcomputer, running its own copy of Kermit.

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

PC is Local, Mainframe is Remote:

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

The KERMIT program on the PC is a *local* Kermit. It can control the screen, the keyboard, and the port separately, thus it can update the screen with status information, watch for interrupt signals from the keyboard, and transfer packets on the communications port, all at the same time.

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

A KERMIT server is always remote, and must get its commands from a local KERMIT. The following descriptions will indicate when a command must be remote or local.

4.2. Command Interface

Most implementations (the UNIX version is the major exception) have an interactive keyword-style command interface, modeled after that of the DECSYSTEM-20, which is roughly as follows: In response to the "`Kermit-`*xx*`>`" *prompt* you may type a keyword, such as SEND, RECEIVE, or EXIT, possibly followed by additional keywords or operands, each of which is called a *field*. You can abbreviate keywords (but not file names) to any length that makes them distinguishable from any other keyword valid for that field. You can type a question mark at any time to get information about what's expected or valid at that point. The ESC and "?" features work best on full duplex systems (all but the IBM mainframe, so far), where the program can "wake up" immediately and perform the required function. On half duplex or record-oriented systems, the ESC feature is not available, and the "?" requires a carriage return to follow.

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

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

4.3. Notation

In the command descriptions, the following notation is used:

anything	A parameter – the symbol in italics is replaced by an argument of the specified type (number, filename, etc).
[*anything*]	An optional field. If omitted, it defaults to an appropriate value.
number	A whole number, entered in prevailing notation of the system.
character	A single character, entered literally, or as a number (perhaps octal or hexadecimal) representing the ASCII value of the character.
floating-point-number	A "real" number, possibly containing a decimal point and a fractional part.
filespec	A file specification, i.e. the name of a file, possibly including a search path, device or directory name, or other qualifying information, and possibly containing "wildcard" or pattern-matching characters to denote a group of files.
^x	A control character may be written using "uparrow" or "caret" notation, since many systems display control characters this way. Control characters are produced by holding down the key marked CTRL or Control and typing the appropriate character, e.g. X.

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

4.4. Summary of KERMIT Commands

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

For exchanging files:
  SEND, RECEIVE, GET

For connecting to a remote host:
  CONNECT, SET LINE, SET PARITY, SET DUPLEX, SET HANDSHAKE, SET ESCAPE, SET FLOW-CONTROL

For acting as a server:
  SERVER

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

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

For defining "macros" of SET commands:
  DEFINE

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

Getting information:
  HELP, STATISTICS, SHOW

Executing command files:
  TAKE

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

For non-protocol file capture or transmission:
  LOG SESSION, TRANSMIT

For closing log files:
  CLOSE

Leaving the program:
  EXIT, QUIT

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

4.5. The SEND Command

Syntax:

Sending a single file:

SEND *nonwild-filespec1* [*filespec2*]

Sending multiple files:

SEND *wild-filespec1* [*filespec2*]

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

Sending a File Group

If *filespec1* contains wildcard characters then all matching files will be sent, in directory-listing order (according to the ASCII collating sequence) by name. If a file can't be opened for read access, it will be skipped. The initial file in a wildcard group can be specified with the optional *filespec2*. This allows a previously interrupted wildcard transfer to continue from where it left off, or it can be used to skip some files that would be transmitted first.

Sending a Single File

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

SEND Command General Operation

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

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

SEND Remote Operation

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

SEND Local Operation

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

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

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

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

4.6. The RECEIVE Command

Syntax: RECEIVE [*filespec*]

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

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

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

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

RECEIVE Remote Operation

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

RECEIVE Local Operation

If you are running KERMIT locally, you should already have issued a SEND command[11] to the remote KERMIT, and then escaped back to DEC-20 Kermit.

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

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

4.7. GET

LOCAL ONLY -- Syntax: **GET** [*remote-filespec*]

The GET command requests a remote KERMIT server to send the file or file group specified by *remote-filespec*. Note the distinction between the RECEIVE and GET commands: RECEIVE puts KERMIT into a passive wait state, whereas GET actively sends a command to a server.

The GET command can be used only when KERMIT is local, with a KERMIT server on the other end of the line. This means that you must have CONNECTed to the other system, logged in, run KERMIT there, issued the SERVER command, and escaped back to the local KERMIT.

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 a continuous indication of the packet traffic. As in the RECEIVE command, you may type Control-X to request that the current incoming file be cancelled, Control-Z to request that the entire incoming batch be cancelled.

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.

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

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

[11] not SERVER -- use the GET command to receive files from a KERMIT server.

4.8. SERVER

REMOTE ONLY -- Syntax: **SERVER**

The SERVER command instructs KERMIT to cease taking commands from the keyboard and to receive all further instructions in the form of KERMIT packets from another system. A KERMIT server must be remote; that is, you must be logged in to the system through another computer, such as a microcomputer. In addition, your local KERMIT should have commands for communicating with remote servers; these include GET, FINISH, and BYE.

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

Any nonstandard parameters should be selected with SET commands before putting KERMIT in server mode, in particular the block check type and special file modes.

4.9. BYE

LOCAL ONLY -- Syntax: **BYE**

When running as a local Kermit talking to a KERMIT server, use the BYE command to shut down and log out the server. This will also close any debugging log files and exit from the local KERMIT.

4.10. FINISH

LOCAL ONLY -- Syntax: **FINISH**

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

4.11. REMOTE

LOCAL ONLY -- Syntax: **REMOTE** *command*

When running in local mode, talking to a remote KERMIT server send the specified command to the remote server. If the server does not understand the command (all of these commands are optional features of the KERMIT protocol), it will reply with a message like "Unknown KERMIT server command". If does understand, it will send the results back, and they will be displayed on the screen. The REMOTE commands are:

CWD [*directory*]	Change Working Directory. If no directory name is provided, the server will change to the default directory. Otherwise, you will be prompted for a password, and the server will attempt to change to the specified directory. If access is not granted, the server will provide a message to that effect.
DELETE *filespec*	Delete the specified file or files. The names of the files that are deleted will appear on your screen.
DIRECTORY [*filespec*]	

The names of the files that match the given file specification will be displayed on your screen. If no file specification is given, all files from the current directory will be listed.

DISK [*directory*]	Provide information about disk usage in the current directory, such as the quota, the current storage, the amount of remaining free space.
HELP	Provide a list of the functions that are available.
HOST [*command*]	Pass the given command to the server's host command processor, and display the resulting output on your screen.
KERMIT [*command*]	Pass the given command, which is expressed in the server KERMIT's own interactive-mode command syntax, to the server for execution. This is useful for changing settings, logging, and other functions.
RUN *program-name* [*command-line-argument*]	Have the remote KERMIT run the indicated program with the indicated command line; send the results back to your screen.
PROGRAM [*command*]	Send the command to the program started by most recent REMOTE RUN program, and display the results on the screen. If no command is given, send newline character.
TYPE *filespec*	Display the contents of the specified file on your screen.

4.12. LOCAL

Syntax: LOCAL *command*

Execute the specified command on the local system -- on the system where KERMIT to which your are typing this command is running. These commands provide some local file management capability without having to leave the KERMIT program, which is particularly useful on microcomputers.

CWD [*directory*]	"Change Working Directory" to the specified directory.
DELETE *filespec*	Delete the specified file or files.
DIRECTORY [*filespec*]	Provide a directory listing of the specified files.

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

4.13. CONNECT

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

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

C	Close the connection and return to the local KERMIT.
S	Show status of the connection.
B	Send a BREAK signal.
0	(zero) Send a NUL (0) character.
P	Push to the local system command processor without breaking the connection.
Q	Quit logging session transcript.
R	Resume logging session transcript.
?	List all the possible single-character arguments.
^]	(or whatever you have set the escape character to be) Typing the escape character twice sends one copy of it to the connected host.

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

4.14. HELP

Syntax: HELP

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

4.15. TAKE

TAKE *filespec*

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

4.16. EXIT, QUIT

EXIT

Exit from KERMIT.

QUIT is a synonym for EXIT.

4.17. The SET Command

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

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

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

The following parameters may be SET:

BAUD-RATE	Set the speed of the current communications port
BLOCK-CHECK	Packet transmission error detection method
DEBUGGING	Mode or log file
DELAY	How long to wait before starting to send
DUPLEX	For terminal connection, full (remote echo) or half (local echo)
ESCAPE	Character for terminal connection
FILE	For setting file parameters like name conversion and byte size
FLOW-CONTROL	Selecting flow control method, like XON/XOFF
HANDSHAKE	For turning around half duplex communication line
IBM	Set things up for communicating with an IBM mainframe
INCOMPLETE	What to do with an incomplete file
LINE	Terminal line to use for terminal connection or file transfer
PARITY	Character parity to use
PORT	For switching communication ports
PROMPT	For changing the program's command prompt
RECEIVE	Various parameters for receiving files
RETRY	How many times to retry a packet before giving up
SEND	Various parameters for sending files

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

SET BAUD-RATE

Set or change the baud rate (approximate translation: transmission speed in bits per second) on the currently selected communications device. The way of specifying the baud rate varies from system to system; in most cases, the actual number (such as 1200 or 9600) is typed. Systems that do not provide this command generally expect that the speed of the line has already been set appropriately outside of KERMIT.

SET BLOCK-CHECK

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

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

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

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

2-CHARACTER-CHECKSUM:	A 2-character, 12-bit checksum. Reduces the probability of an error going undetected to 1/4096, but adds an extra character to each packet.
3-CHARACTER-CRC:	A 3-character, 16-bit Cyclic Redundancy Check, CCITT format. In addition to errors in any odd number of bits, this method detects double bit errors, all error bursts of length 16 or less, and more than 99.99% of all possible longer bursts. Adds two extra characters to each packet.

The single character checksum has proven to be quite adequate in practice, much more effective than straightforward analysis would indicate, since all errors are *not* equally likely, and a simple checksum is well suited to catching the kinds of errors that are typical of telecommunication lines. The other methods should be requested only when the connection is *very* noisy.

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

SET DEBUG

Syntax: SET DEBUG *options*

Record the packet traffic, either on your terminal or in a file. Options are:

STATES	Show Kermit state transitions and packet numbers (brief).
PACKETS	Display each incoming and outgoing packet (lengthy).
LOG-FILE	Log the selected information (STATES or PACKETS) to the specified file. If log file not specified, then use the terminal if local.
OFF	Don't display debugging information (this is the default). If debugging was in effect, turn it off and close any log file.

SET DELAY

Syntax: SET DELAY *number*

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

SET DUPLEX

Syntax: SET DUPLEX *keyword*

For use when CONNECTed to a remote system. The keyword choices are FULL and HALF. FULL means the remote system echoes the characters you type, HALF means the local system echoes them. FULL is the default, and is used by most hosts. HALF is necessary when connecting to IBM mainframes. Half duplex is also called "local echo".

SET ESCAPE

Syntax: `SET ESCAPE` *character*

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

SET FILE

Syntax: `SET FILE` *parameter keyword*

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

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

SET FLOW-CONTROL

Syntax: `SET FLOW-CONTROL` *option*

For communicating with full duplex systems. System-level flow control is not necessary to the KERMIT protocol, but it can help to use it if the same method is available on both systems. The most common type of flow control on full duplex systems is XON/XOFF.

SET HANDSHAKE

Syntax: `SET HANDSHAKE` *option*

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

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

SET INCOMPLETE

Syntax: `SET INCOMPLETE` *option*

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

SET LINE

Syntax: SET LINE [*terminal-designator*]

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

SET PORT

Syntax: SET PORT *terminal-designator*

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

SET PARITY

Syntax: SET PARITY *keyword*

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

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

The choices for SET PARITY are:

NONE	(the default) eight data bits and no parity bit.
MARK	seven data bits with the parity bit set to one.
SPACE	seven data bits with the parity bit set to zero.
EVEN	seven data bits with the parity bit set to make the overall parity even.
ODD	seven data bits with the parity bit set to make the overall parity odd.

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

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

SET PROMPT

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

SET SEND

SET SEND *parameter*

Parameters for outgoing packets, as follows:

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

PACKET-LENGTH *number*
: Maximum packet length to send between 10 and 94 (decimal). Shortening the packets might allow more of them to get throu̇gh through without error on noisy communication lines. Lengthening the packets increases the throughput on clean lines.

TIMEOUT *number*
: How many seconds to wait for a packet before trying again.

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

PADDING *number*, PADCHAR *character*
: How much padding to send before a packet, if the other side needs padding, and what character to use for padding. Defaults are no padding, and NUL (0) for the padding character.

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

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

SET RECEIVE

Syntax: **SET RECEIVE** *parameter*

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

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

PACKET-LENGTH *number*
Maximum length packet for the other side to send, decimal number, between 10 and 94, decimal.

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

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

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

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

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

SET RETRY

SET RETRY *option number*

Set the maximum number of retries allowed for:

INITIAL-CONNECTION
How many times to try connecting before giving up, normally something like 15.

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

4.18. DEFINE

DEFINE *macroname [set-parameters]*

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 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
DEFINE UNIX PARITY NONE, DUPLEX FULL, HANDSHAKE NONE
DEFINE TELENET PARITY MARK, RECEIVE TIMEOUT 20
```

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.

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

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

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

```
DEFINE IBM
```

You can list all your macros and their definitions with the SHOW MACROS command.

4.19. SHOW

Syntax: `SHOW` *[option]*

The SHOW command displays the values of the parameters settable by the SET command. If a particular option is not requested, a complete display will be provided.

4.20. STATISTICS

Give statistics about the most recent file transfer, such as the total number of characters transmitted, the effective baud rate, and so forth.

4.21. LOG

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

Log the specified entity to the specified log file.

TRANSACTIONS	Direct KERMIT to log transactions, such as files successfully sent or received or files that could not be successfully sent or received. A transaction is useful recording the progress of a long, unattended multifile transfer.
SESSION	Create a transcript of a CONNECT session, when running a local KERMIT connected to a remote system, in the specified file. The log is closed when connection is closed. In some implementations, logging can be "toggled" by typing the connect escape character followed by Q (Quit logging) or R (Resume logging) or similar single-character commands. Session-logging is useful for recording dialog with an interactive system, and for "capturing" from systems that don't have KERMIT. No guarantee can be made that the file will arrive correctly or completely, since no error checking takes place.
DEBUGGING	Record debugging information in the specified file. There may be several options to select the desired information -- entire packets, state transitions, etc -- available via the SET DEBUGGING command.

4.22. TRANSMIT

Syntax: `TRANSMIT` *filespec*

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

5. KERMIT IMPLEMENTATIONS

Kermit has been written for a wide variety of systems, both mainframes and microcomputers. Kermit is not written in a portable language; rather, each implemenation is written in the language best suited for the particular machine. The specification, given in the *Kermit Protocol Manual*, is quite general and allows implementation on almost any machine.

Here's a brief table summarizing the known Kermit implementations, as of this writing. This list is constantly growing, and may be far out of date by the time you read it.

Machine	Operating System	Language
DECsystem-10,20	TOPS-10,20	MACRO-10,20
IBM 370 Series	VM/CMS, MVS/TSO	IBM Assembler
IBM 370 Series	MTS	Pascal
CDC Cyber 170	NOS	Fortran-77
Sperry/Univac-1100	EXEC	EXEC Assembler
Honeywell	MULTICS	PL/I
DEC VAX-11	VMS	Bliss-32, Macro-32
DEC PDP-11	RT,RSX,RSTS,MUMPS	MACRO-11 & others
DEC Pro-300 Series	P/OS	Bliss-16, Macro-11
VAX,PDP-11,SUN,etc	UNIX	C
PRIME	PRIMOS	PL/P
HP3000, Univac, etc	Software Tools	Ratfor
HP1000	RTE	Fortran
Apollo	Aegis	Fortran
Terak, HP-98x6, IBM PC	UCSD p-System	UCSD Pascal
8080, 8085, or Z80	CP/M-80	ASM
8086, 8088	PC-DOS, MS-DOS	MS MASM-86
8086, 8088	CP/M-86	DR ASM86
Apple II 6502	Apple DOS	DEC-10/20 CROSS
TRS80 I, III	TRSDOS	Z80 Assembler
Atari	DOS	Action!

The **8080 version** runs on the DEC VT180, DECmate II (CP/M), Heath/Zenith-89 and 100, Superbrain, Apple II/Z80, TRS-80 II (CP/M), Osborne, Kaypro, and others. There are **8086 MS DOS versions** for the IBM PC, DEC Rainbow, Wang PC, Compaq, Heath/Zenith-100, HP-150, Tandy 2000, Victor 9000, and others. The **8086 CP/M-86 version** runs on the DEC Rainbow and the NEC APC.

The remainder of the KERMIT User Guide is devoted to descriptions of selected KERMIT implementations. If a description of your version of KERMIT does not appear, look in the KERMIT area on your mainframe for an on-line documentation file. Even if your version is described below, the version of the manual you are reading may be out of date and the online information may be more current.

6. DECSYSTEM-20 KERMIT

Authors:	Frank da Cruz, Bill Catchings, Columbia University
Language:	MACRO-20
Version:	4.1(236)
Date:	July 3, 1984

KERMIT-20 Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
^X/^Y interruption:	Yes
Filename collision avoidance:	(Uses generation numbers)
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	Yes
Alternate block checks:	Yes
Terminal emulation:	Yes
Communication settings:	Yes
Transmit BREAK:	Yes
IBM communication:	Yes
Transaction logging:	Yes
Session logging:	Yes
Raw transmit:	No
Act as server:	Yes
Talk to server:	Yes
Advanced commands for servers:	Yes
Local file management:	Yes
Handle file attributes:	No
Command/init files:	Yes

KERMIT-20 is a program that implements the KERMIT file transfer protocol for the Digital Equipment Corporation DECSYSTEM-20 mainframe computer. It is written in MACRO-20 assembly language and should run on any DEC-20 system with version 4 of TOPS-20 or later.

The KERMIT-20 section will describe the things you should know about the DEC-20 file system in order to make effective use of KERMIT, and then it will describe the special features of the KERMIT-20 program.

6.1. The DEC-20 File System

The features of the DEC-20 file system of greatest interest to KERMIT users are the form of the file specifications, and the distinctions between text and binary files.

DEC-20 File Specifications

DEC-20 file specifications are of the form

```
    DEVICE:<DIRECTORY>NAME.TYPE.GEN;ATTRIBUTES
```

where the DIRECTORY, NAME, and TYPE may each be up to 39 characters in length, GEN is a generation (version number), and various attributes are possible (protection code, account, temporary, etc). Generation and attributes are normally omitted. Device and directory, when omitted, default to the user's own (or "connected") disk and directory. Thus NAME.TYPE is normally sufficient to specify a file, and only this information is sent along by KERMIT-20 with an outgoing file.

The device, directory, name, and type fields may contain uppercase letters, digits, and the special characters "-" (dash), "_" (underscore), and "$" (dollar sign). There are no imbedded or trailing spaces. Other characters may be included by prefixing them (each) with a Control-V. The fields of the file specification are set off from one another by the punctuation indicated above.

The device field specifies a physical or "logical" device upon which the file is resident. The directory field indicates the area on the device, for instance the area belonging to the owner of the file. KERMIT-20 does not transmit the device or directory fields to the target system, and does not attempt to honor device or directory fields that may appear in incoming file names; for instance, it will not create new directories.

The name is the primary identifier for the file. The type, also called the "extension", is an indicator which, by convention, tells what kind of file we have. For instance FOO.FOR is the source of a Fortran program named FOO; FOO.REL might be the relocatable object module produced by compiling FOO.FOR; FOO.EXE could an executable program produced by LOADing and SAVing FOO.REL, and so forth.

The DEC-20 allows a group of files to be specified in a single file specification by including the special "wildcard" characters, "*" and "%". A "*" matches any string of characters, including no characters at all; a "%" matches any single character. Here are some examples:

*.FOR	All files of type FOR (all Fortran source files) in the connected directory.
FOO.*	Files of all types with name FOO.
F*.*	All files whose names start with F.
F*X*.*	All files whose names start with F and contain at least one X.
%.*	All files whose names are exactly one character long.
.%%%	All files whose types are at least three characters long.

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

Text Files and Binary Files

The DEC-20, like most computers, has a file system with its own peculiarities. Like many other systems, the DEC-20 makes a distinction between *text files* and *binary files*. Text files are generally those composed only of printing characters (letters, digits, and punctuation) and "carriage control" characters (carriage return, line feed, form feed, tab). Text files are designed to be read by people. Binary files are designed to be read by a computer program, and may have any contents at all. If you use the DEC-20 TYPE command to display a text file on your terminal, the result will be intelligible. If you type a binary file on your terminal, you will probably see mainly gibberish. You can not always tell a text file from a binary file by its name or directory information, though in general files with types like .TXT, .DOC, .HLP are textual (as are "source files" for computer programs like text formatters and programming language compilers), and files with types like .EXE, .REL, .BIN are binary.

The DEC-20 has an unusual word size, 36 bits. It differs from most other systems by storing text in 7-bit, rather than 8-bit, bytes. Since text is encoded in the 7-bit ASCII character set, this allows more efficient use of storage. However, the word size is not a multiple of the normal byte size. The DEC-20 therefore stores five 7-bit characters per word, with one bit left over.

It is also possible to store files with other byte sizes. The common layouts of bytes within a word are:

7 Text Files: Five 7-bit bytes per word.

```
+------+------+------+------+------++
!      !      !      !      !      !!
+------+------+------+------+------++
0      7      14     21     28     35
```

Normally, bit 35 is unused and set to zero. However, in EDIT (or SOS, or OTTO) line-numbered files, bit 35 is set to 1 when the word contains a line number.

8 "Foreign" binary files: Four 8-bit bytes per word.

```
+-------+-------+-------+-------+---+
!       !       !       !       !   !
+-------+-------+-------+-------+---+
0       8       16      24      32  35
```

Bits 32-35 are unused.

36 "Native" binary files: One 36-bit byte per word.

```
+-----------------------------------+
!                                   !
+-----------------------------------+
0                                   35
```

All bits are used.

The minimum unit of disk allocation on the DEC-20 is a *page*, 512 36-bit words, or 2560 7-bit characters, or 2048 8-bit bytes. Any file that contains at least one bit of information occupies at least a full page on the disk. The directory information for a file includes the number of pages occupied on the disk, the bytesize of the file, and the number of bytes of that size which are in the file. This information can be seen by using the DEC-20 VDIRECTORY command, for instance

```
@vdir foo.*
```

```
  PS:<MY-DIRECTORY>
Name       Protection    Pages Bytes(Size) Creation
FOO.COM.1;P774242            1 384(8)      27-Dec-83
   MAC.1;P774242             1 152(7)      27-Dec-83
   .REL.1;P774242            1 39(36)      27-Dec-83
   .EXE.1;P774242            2 1024(36)    27-Dec-83

Total of 5 pages in 4 files
```

In this example, FOO.MAC occupies 1 page, and is composed of 152 7-bit bytes. This file is textual (program source for the MACRO assembler), 152 characters long. Programs which read text files (such as text editors, program compilers, the TYPE command, etc) determine the end of a file from the byte count specified in the directory. KERMIT-20 determines the end of file in the same way, so although FOO.MAC occupies an entire 2560-byte page of storage, only the first 152 characters are transmitted. Binary files, such as FOO.EXE (an executable DEC-20 program), tend to occupy full pages. In this case too, KERMIT-20 uses the byte count to determine the end of file.

Why do you need to know all this? In most cases, you don't. It depends on whether you are using the DEC-20 as your "home base".

Using a Microcomputer to Archive DEC-20 Files

Most computers (other than the DEC-10 and DEC-20) store characters in 8-bit bytes. Let's call any such system an 8-bit-byte system. This discussion applies to all 8-bit-byte systems, including all popular microcomputers, minicomputers like the DEC PDP-11 and VAX, and mainframes like the IBM 370. For simplicity, we'll focus on microcomputers.

KERMIT can send any "native" DEC-20 sequential file, text or binary, to an 8-bit-byte system and bring it back to the DEC-20 restored to its original form. If you are using a microcomputer to archive your DEC-20 files, you need never concern yourself with details of byte size or file format. The same holds true between two DEC-20s, or a DEC-10 and a DEC-20.

There is, however, one special complication of which you should be aware. Certain microcomputer operating systems, notably CP/M, do not have an entirely satisfactory way of indicating the end of file. The file length is recorded in blocks rather than bytes. For text files, the end of file is marked within a block by inserting a Control-Z after the last data character. Binary files, however, might easily contain Control-Z characters as data. Therefore, in order not to lose data, these systems must transmit binary files in complete blocks. If the binary file is of foreign origin (for instance, from a DEC-20), and it did not happen to fill up the last block when it was transferred to the micro, then when that file is sent back to the system of origin in "binary mode," junk will appear at the end (if it is sent back in "text mode," it might be truncated by any data byte that happened to correspond to Control-Z). For DEC-20 programs in .EXE format, this generally has no effect on the runnability or behavior of the program. But for other binary files, particularly internal format numerical data or relocatable program object (.REL) files, the junk could have bad effects. Extraneous data at the end of a .REL file will generally cause LINK to fail to load the file.

Using the DEC-20 to Archive Microcomputer Files

You can use KERMIT to send textual files from a microcomputer or any 8-bit system to the DEC-20 with no special provisions, since KERMIT-20 stores incoming characters in 7-bit bytes as text unless you explicitly instruct it otherwise. But KERMIT-20 has no automatic way of distinguishing an incoming binary file from an incoming text file.[12] Binary files from 8-bit-byte systems generally contain significant data in the 8th bit, which would be lost if the incoming characters were stored in 7-bit bytes, rendering the file useless when sent back to the original system. Thus if you want to use KERMIT to store foreign 8-bit binary data on the DEC-20, you must tell it to store such files with a bytesize of 8 rather than 7. This can be the source of much confusion and inconvenience. In particular, you cannot use a "wildcard send" command to send a mixture of text and binary files from an 8-bit-byte system to the DEC-20; rather, you must send all text files with KERMIT-20's file bytesize set to 7, and all 8-bit binary files with the bytesize set to 8.

Once you get the foreign binary file into the DEC-20, stored with the correct bytesize (as **FOO.COM** is stored in the example above), you need take no special measures to send it back to its system of origin. This is because KERMIT-20 honors the bytesize and byte count from the directory. For instance, if you told KERMIT-20 to **SEND FOO.***, every file in the example above would be transmitted in the correct manner, automatically.

Files KERMIT-20 Cannot Handle

The KERMIT protocol can only accommodate transfer of *sequential* files, files which are a linear sequence of bytes (or words).

Some files on the DEC-20 are not sequential, and cannot be successfully sent or received by KERMIT-20. These include directory files, files with holes (missing pages), ISAM files, and RMS files. These files require external information (kept in the DEC-20's file descriptor block and/or index table) in order to be reconstructed; when sending files, KERMIT-20 presently transmits only the file name and the contents of the file. External control information (file attributes) are not transmitted.

6.2. Program Operation

Kermit-20's prompt is "**Kermit-20>**". Kermit-20 will accept a single command on the Exec command line, like this:

```
@
@kermit send foo.bar

  the file is sent

@
```

or you can run the program interactively to issue several commands, like this:

```
@
@kermit

TOPS-20 KERMIT version 4.1(236)

Kermit-20>send foo.*

  files are sent

Kermit-20>statistics

  performance statistics are printed

Kermit-20>receive
```

12 Unless the incoming file has an "ITS Binary Header"; see below.

files are received

```
Kermit-20>exit
@
```

During interactive operation, you may use the TOPS-20 help ("?") and recognition (ESC) features freely while typing commands. A question mark typed at any point in a command displays the options available at that point; typing an ESC character causes the current keyword or filename to be completed (or default value to be supplied), and a "guide word" in parentheses to be typed, prompting you for the next field. If you have not typed sufficient characters to uniquely specify the keyword or filename (or if there is no default value) then a beep will be sounded and you may continue typing.

Command keywords may be abbreviated to their shortest prefix that sets them apart from any other keyword valid in that field.

If you have a file called `KERMIT.INI` in your login directory, KERMIT-20 will execute an automatic TAKE command on it upon initial startup. `KERMIT.INI` may contain any KERMIT-20 commands, for instance SET commands, or DEFINEs for SET macros to configure KERMIT-20 to various systems or communications media.

KERMIT-20 provides most of the commands possible for an "ideal" KERMIT program, as described in the main part of the *KERMIT User Guide*. The following sections will concentrate on system-dependent aspects of KERMIT-20.

KERMIT-20 disables terminal links, advice, and system messages in order to minimize interference with data transfer (and restores these to their previous value upon completion of a transfer). However, certain messages cannot be disabled by KERMIT because they are issued by KERMIT's superior controlling process, the TOPS-20 EXEC. These include mail notifications and alerts. Before running KERMIT-20 to transfer files, you should SET NO MAIL-WATCH and SET NO ALERT.

6.3. Remote and Local Operation

KERMIT-20 normally runs in remote mode, with the user sitting at a PC. But KERMIT-20 can also run in local mode. Local operation of KERMIT-20 is useful if the DEC-20 has an autodialer, or a hardwired connection to another computer. When in local mode, file transfer takes place over an assigned TTY line, and KERMIT-20 is free to update your screen with status information, and to listen to your keyboard for interrupt characters.

Local Operation of KERMIT-20:

```
 DECSYSTEM-20
+---------------------------------------+
|                                       |
|   +--------------------+              |
|   | Your Job           |              |
|   |                    |              |
|   |  +------------+    | <--Commands  | Your Job's
|   |  | KERMIT-20  +----+--------------+----------------0 You
|   |  |            |    | Display--->  | Controlling TTY
|   |  |            |    |              |
|   |  |            |    | <--Packets   | Kermit's
|   |  |            +----+--------------+----------------> Remote
|   |  +------------+    | Packets-->   | Assigned TTY      System
|   |                    |              |
|   +--------------------+              |
|                                       |
+---------------------------------------+
```

KERMIT-20 enters local mode when you issue a SET LINE *n* command, where *n* is the octal TTY number of any line other than your own controlling terminal.

6.4. Conditioning Your Job for Kermit

Kermit-20 does as much as it can to condition your line for file transfer. It saves all your terminal and link settings, and restores them after use. However, there are some sources of interference over which Kermit-20 can have no control. In particular, messages issued by superior or parellel forks could become mingled with Kermit packets and slow things down or stop them entirely. For this reason, before using Kermit-20 for any extended period, you should:

- Type the Exec commands SET NO MAIL-WATCH and SET NO ALERTS
- Make sure you don't have any print or batch jobs pending that were submitted with the /NOTIFY option.

After running Kermit, you can restore your mail-watch and alerts.

6.5. KERMIT-20 Commands

This section describes the KERMIT-20 commands, in detail where they differ from the "ideal" KERMIT, briefly where they coincide.

The SEND Command

Syntax:

Sending a single file:

SEND *nonwild-filespec1* (**AS**) [*filespec2*]

Sending multiple files:

SEND *wild-filespec1* (**INITIAL**) [*filespec2*]

The SEND command causes a file or file group to be sent from the DEC-20 to the other system. There are two forms of the command, depending on whether *filespec1* contains wildcard characters ("*" or "%"). KERMIT-20 automatically recognizes the two cases and issues the appropriate guide word, (AS) or (INITIAL), depending on the form of *filespec1*.

Sending a File Group

If *filespec1* contains wildcard characters then all matching files will be sent, in alphabetical order (according to the ASCII collating sequence) by name. If a file can't be opened for read access, it will be skipped. The initial file in a wildcard group can be specified with the optional *filespec2*. This allows a previously interrupted wildcard transfer from where it left off, or it can be used to skip some files that would be transmitted first.

Sending a Single File

If *filespec1* does not contain any wildcard characters, then the single file specified by *filespec1* will be sent. Optionally, *filespec2* may be used to specify the name under which the file will arrive at the target system; *filespec2* is not parsed or validated in any way by KERMIT-20, but lower case letters are raised to upper case, and leading "whitespace" (blanks and tabs) are discarded. If *filespec2* is not specified, KERMIT-20 will send the file with its own name.[13]

SEND Command General Operation

Files will be sent with their DEC-20 filename and filetype (for instance FOO.BAR, no device or directory field, no generation number or attributes). If you expect to be sending files whose names contain characters that would be illegal in filenames on the target system, and you know that the KERMIT on the target system does not have the ability to convert incoming filenames, you can issue the SET FILE NAMING NORMAL-FORM command to have KERMIT-20 replace suspect characters by X's.

Each file will be sent according to its bytesize and byte count from the directory unless you specify otherwise using SET FILE BYTESIZE, or unless the file has an "ITS Binary" header. If the bytesize is 8, then four 8-bit bytes will be sent from each DEC-20 36-bit word, and the low order four bits will be skipped. If other than 8, then five 7-bit bytes will be sent from each word, with the 8th bit of the 5th character set to the value of the remaining bit ("bit 35") from the word.[14]

If communication line parity is being used (see SET PARITY), KERMIT-20 will request that the other KERMIT accept a special kind of prefix notation for binary files. This is an advanced feature, and not all KERMITs have it; if the other KERMIT does not agree to use this feature, binary files cannot be sent correctly. This includes executable programs (like DEC-20 .EXE files, CP/M .COM files), relocatable object modules (.REL files), as well as text files with line sequence numbers.

KERMIT-20 will also ask the other KERMIT whether it can handle a special prefix encoding for repeated characters. If it can, then files with long strings of repeated characters will be transmitted very efficiently. Columnar data, highly indented text, and binary files are the major beneficiaries of this technique.

If you're running KERMIT-20 locally, for instance dialing out from the DEC-20 to another system using an autodialer, you should have already run KERMIT on the remote system and issued either a RECEIVE or a SERVER command. Once you give KERMIT-20 the SEND

---

[13] Control-V's, which are used to quote otherwise illegal characters in DEC-20 file specifications, are stripped.

[14] This is the same method used by the DEC-20 to encode 36-bit data on "ANSI-ASCII" tapes. It allows not only DEC-20 binary files, but also the line-sequence-numbered files produced by EDIT, SOS, or OTTO, which use bit 35 to distinguish line numbers from text, to be sent and retrieved correctly.

command, the name of each file will be displayed on your screen as the transfer begins; a "." will be displayed for every 5 data packets sucessfully sent, and a "%" for every retransmission or timeout that occurs (you may also elect other typeout options with the SET DEBUG command). If the file is successfully transferred, you will see "[OK]", otherwise there will be an error message. When the specified operation is complete, the program will sound a beep. If you see many "%" characters, you are probably suffering from a noisy connection. You may be able to cut down on the retransmissions by using SET SEND PACKET-LENGTH to decrease the packet length; this will reduce the probability that a given packet will be corrupted by noise, and reduce the time required to retransmit a corrupted packet.

During local operation, you can type Control-A at any point during the transfer to get a brief status report. You may also type Control-X or Control-Z to interrupt the current file or file group.

The RECEIVE Command

Syntax: `RECEIVE` [*filespec*]

The RECEIVE command tells KERMIT-20 to receive a file or file group from the other system. If only one file is being received, you may include the optional *filespec* as the name to store the incoming file under; otherwise, the name is taken from the incoming file header. Even if the name in the header is not a legal TOPS-20 file name, KERMIT-20 will store it under that name, in which case you can refer to it later only by quoting each illegal character (spaces, control characters, etc) with Control-V. If for some reason an incoming filename simply cannot be converted to legal form, the file will be saved as `-UNTRANSLATABLE-FILENAME-.KERMIT` (new generation). You may also use SET FILE NAMING NORMAL-FORM to have KERMIT-20 choose more conventional names for incoming files.

If an incoming file has the same name as an existing file, KERMIT-20 just creates a new generation of the same name and type, for instance `FOO.BAR.3`, `FOO.BAR.4`. The oldest generation will be automatically deleted, but you can still UNDELETE it.

Incoming files will all be stored with the prevailing bytesize, 7 by default, which is appropriate for text files. If you are asking KERMIT-20 to receive binary files from a microcomputer or other 8-bit system, you must first type SET FILE BYTESIZE 8. Otherwise, the 8th bit of each byte will be lost and the file will be useless when sent back to the system of origin.

If you have SET PARITY, then 8th-bit prefixing will be requested. If the other side cannot do this, binary files cannot be transferred correctly. In all cases, KERMIT-20 will request the other KERMIT to compress repeated characters; if the other side can do this (not all KERMITs know how) there may be a significant improvement in transmission speed.

If an incoming file does not arrive in its entirety, KERMIT-20 will normally discard it; it will not appear in your directory. You may change this behavior by using the command SET INCOMPLETE KEEP, which will cause as much of the file as arrived to be saved in your directory.

If you are running KERMIT-20 locally, you should already have issued a SEND command[15] to the remote KERMIT, and then escaped back to DEC-20 Kermit. As files arrive, their

[15] not SERVER -- use the GET command to receive files from a KERMIT server.

names will be displayed on your screen, along with "." and "%" characters to indicate the packet traffic; you can type Control-A during the transfer for a brief status report.

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

The GET Command

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

The GET command requests a remote KERMIT server to send the file or file group specified by *remote-filespec*. This command can be used only when KERMIT-20 is local, with a KERMIT server on the other end of the line specified by SET LINE. This means that you must have CONNECTed to the other system, logged in, run KERMIT there, issued the SERVER command, and escaped back to the DEC-20.

The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. If you need to include otherwise illegal characters such as "!" or ";" (the normal command comment delimeters), "?" (the command help character), "@" (the indirect command file indicator), or certain control characters, then you should precede each such character by a Control-V. Kermit-20 will discard these Control-V quoting prefixes before sending the file specification to the remote host.

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

```
Kermit-20>get
 Remote Source File: profile exec a1
 Local Destination File: profile.exec
```

As files arrive, their names will be displayed on your screen, along with "." and "%" characters to indicate the packet traffic. As in the RECEIVE command, you may type Control-A to get a brief status report, ↑X to request that the current incoming file be cancelled, ↑Z to request that the entire incoming batch be cancelled.

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

The SERVER Command

The SERVER command puts a remote KERMIT-20 in "server mode", so that it receives all further commands in packets from the local KERMIT. The KERMIT-20 server is capable (as of this writing) of executing many remote server commands, including SEND, GET, FINISH, BYE, REMOTE DIRECTORY, REMOTE CWD, REMOTE SPACE, REMOTE DELETE, REMOTE TYPE, REMOTE HELP.

Any nonstandard parameters should be selected with SET commands before putting KERMIT-20 into server mode, in particular the file bytesize. The DEC-20 Kermit server can send most files in the correct manner automatically, by recognizing the DEC-20 file bytesize. However, if you need to ask the DEC-20 KERMIT server to receive binary files from an 8-bit-byte system (that is, from almost any system that's not a DEC-10 or DEC-20) you must issue the SET FILE BYTESIZE 8 command before putting it into server mode, and then you must only send 8-bit binary files. You cannot send a mixture of text files and 8-bit binary files to a KERMIT-20 server.

Commands for Servers

When running in local mode, KERMIT-20 allows you to give a wide range of commands to a remote KERMIT server, with no guarantee the that the remote server can process them, since they are all optional features of the protocol. Commands for servers include the standard **SEND**, **GET**, **BYE**, and **FINISH** commands, as well as the **REMOTE** command, which has various options.

Syntax: **REMOTE** *command*

Send the specified command to the remote server. If the server does not understand the command (all of these commands are optional features of the KERMIT protocol), it will reply with a message like "Unknown KERMIT server command". If does understand, it will send the results back, and they will be displayed on the screen. The REMOTE commands are:

CWD [*directory*]	Change Working Directory. If no directory name is provided, the server will change to the default or home directory. Otherwise, you will be prompted for a password, and the server will attempt to change to the specified directory. The password is entered on a separate line, and does not echo as you type it. If access is not granted, the server will provide a message to that effect.
DELETE *filespec*	Delete the specified file or files. The names of the files that are deleted will appear on your screen.
DIRECTORY [*filespec*]	The names of the files that match the given file specification will be displayed on your screen, perhaps along with size and date information for each file. If no file specification is given, all files from the current directory will be listed.
HELP	Provide a list of the functions that are available from the server.
HOST [*command*]	Pass the given command to the server's host command processor, and display the resulting output on your screen.
SPACE	Provide information about disk usage in the current directory, such as the quota, the current storage, the amount of remaining free space.
TYPE *filespec*	Display the contents of the specified file on your screen.

The LOCAL Command

Syntax: **LOCAL** [*command*]

Execute the specified command on the local system -- on the DEC-20 where KERMIT-20 is running. These commands provide some local file management capability without having to leave the KERMIT-20 program.

CWD [*directory*] Change working directory, or, in DEC-20 terminology, CONNECT to the specified directory.

DELETE *filespec* Delete the specified file or files, but do not expunge them.

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

RUN [*filespec*] Attempts to run the specified file, which must be in ".EXE" format (.EXE is the default filetype), in an inferior fork. Control returns to KERMIT-20 when the program terminates. Once you have used this command, you can restart the same program by issuing a RUN command with no arguments. If you RUN SYSTEM:EXEC, then you will be able to issue TOPS-20 commands without leaving KERMIT; you can get back to KERMIT from the EXEC by typing the EXEC POP command. You can also use the RUN command to supply new functions to KERMIT by writing little programs in the language of your choice, for instance to conduct a signon dialog with a remote system when dialing out.

SPACE Show how much space is used and remaining in the current directory.

TYPE Display the contents of the specified file or files at your terminal. This works like the DEC-20 TYPE command, except that if a file has a bytesize of 8, KERMIT-20 will do 8-bit input from it rather than 7-bit. Also, the DEC-20 Control-O command discards output only from the file currently being displayed; if multiple files are being typed, then output will resume with the next file.

The LOCAL commands may also be used without the "LOCAL" prefix.

The CONNECT Command

Syntax: CONNECT [*number*]

Establish a terminal connection to the system connected to the octal TTY number specified here or in the most recent SET LINE command, using full duplex echoing and no parity unless otherwise specified in previous SET commands. Get back to KERMIT-20 by typing the escape character followed by the letter C. The escape character is Control-Backslash (^\) by default. When you type the escape character, several single-character commands are possible:

C Close the connection and return to KERMIT-20.
S Show status of the connection; equivalent to SHOW LINE.
P Push to a new Exec. POP from the Exec to get back to the connection.
Q If a session log is active, temporarily Quit logging.
R Resume logging to the session log.
B Send a simulated BREAK signal.
? List all the possible single-character arguments.
^\ (or whatever you have set the escape character to be)
Typing the escape character twice sends one copy of it to the connected host.

You can use the SET ESCAPE command to define a different escape character, and SET PARITY, SET DUPLEX, SET HANDSHAKE, SET FLOW, and SET SPEED to change those communication-line-oriented parameters. In order for the simulated BREAK signal to work, TOPS-20 must know the speed of the terminal. If it does not, you may use the SET SPEED command.

KERMIT-20 does not have any special autodialer interface. It assumes that the connection has already been made and the line assigned.

The HELP Command

Syntax: HELP [*topic* [*subtopic*]]

Typing HELP alone prints a brief summary of KERMIT-20 and its commands. You can also type

HELP *command*

for any Kermit-20 command, e.g. "help send" or "help set parity" to get more detailed information about a specific command. Type

HELP ?

to see a list of the available help commands.

The TAKE Command

Syntax: TAKE *filespec*

Execute KERMIT-20 commands from the specified file. The file may contain contain any valid KERMIT-20 commands, including other TAKE commands; command files may be nested up to a depth of 20.

The EXIT and QUIT Commands

Syntax: EXIT

Exit from KERMIT-20. You can CONTINUE the program from the TOPS-20 Exec, provided you haven't run another program on top of it. You can also exit from KERMIT-20 by typing one or more control-C's, even if it's in the middle of transferring a file. KERMIT-20 will always restore your terminal to its original condition, and you will be able to CONTINUE the program to get back to "KERMIT-20>" command level with current settings intact.

QUIT is a synonym for EXIT.

The SET Command

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

Establish or modify various parameters for file transfer or terminal connection. You can examine their values with the SHOW command. The following parameters may be SET:

BREAK	Adjust the BREAK simulation parameter
BLOCK-CHECK	Packet transmission error detection method
DEBUGGING	Record or display state transitions or packets
DELAY	How long to wait before starting to send
DUPLEX	For terminal connection, FULL or HALF
ESCAPE	Character for terminal connection
FILE	For setting file parameters like byte size
FLOW-CONTROL	For enabling or disabling XON/XOFF flow control

HANDSHAKE	For turning around half duplex communication line
IBM	For communicating with an IBM mainframe
INCOMPLETE	What to do with an incomplete file
ITS-BINARY	For recognizing a special 8-bit binary file format
LINE	TTY line to use for file transfer or CONNECT
PARITY	Character parity to use
PROMPT	Change the program's command prompt
RECEIVE	Various parameters for receiving files
RETRY	How many times to retry a packet before giving up
SEND	Various parameters for sending files
SPEED	Baud rate of communication line
TVT-BINARY	For negotiating binary mode on ARPANET

The DEFINE command may be used to compose "macros" by combining SET commands. Those SET commands which differ from the "ideal" KERMIT are now described in detail.

SET BREAK

Syntax: **SET BREAK** *n* Specify the number of nulls to be sent at 50 baud to simulate a BREAK signal when connected to a remote host via SET LINE and CONNECT.

SET DEBUG

Syntax: **SET DEBUG** *options*

Record the packet traffic, either on your terminal or in a file. Some reasons for doing this would be to debug a version of KERMIT that you are working on, to record a transaction in which an error occurred for evidence when reporting bugs, or simply to vary the display you get when running KERMIT-20 in local mode. Options are:

STATES	Show Kermit state transitions and packet numbers (brief).
PACKETS	Display each incoming and outgoing packet (lengthy).
OFF	Don't display or record debugging information (this is the normal mode). If debugging was in effect, turn it off and close any log file

The debugging information is recorded in the file specified by the most recent LOG DEBUGGING command.

SET ESCAPE

SET ESCAPE *octal-number*

Specify the control character you want to use to "escape" from remote connections back to KERMIT-20. The default is 34 (Control-\). The number is the octal value of the ASCII control character, 1 to 37 (or 177), for instance 2 is Control-B. After you type the escape character, you must follow it by a one of the single-character "arguments" described under the CONNECT command, above.

SET EXPUNGE

`SET EXPUNGE ON` *or* `OFF`

Tell whether you want a DELETE command (either the LOCAL DELETE command or a REMOTE DELETE command sent to a KERMIT-20 server) to expunge files as it deletes them. On the DEC-20, a deleted file continues to take up space, and may be "undeleted" at a later time in the same session. To expunge a deleted file means to remove it completely and irrevocably, freeing its space for further use. EXPUNGE is OFF by default; deleted files are not automatically expunged. SET EXPUNGE applies only to files that are deleted explicitly by KERMIT-20, and not to files that are implicitly deleted when new generations of existing files are created.

SET FILE

Syntax: `SET FILE` *parameter keyword*

Establish file-related parameters:

BYTESIZE *keyword or number*
Byte size for DEC-20 file input/output. The choices are SEVEN (7), EIGHT (8), and AUTO.

SEVEN (or 7) Always store or retrieve five 7-bit bytes per word. When sending a file, ignore the file bytesize and do 7-bit input from the file. There would be no reason to use this option except to explicitly force an 8-bit file to be treated as a 7-bit file.

EIGHT (or 8) Always store or retrieve four 8-bit bytes per word. When sending a file, ignore the file bytesize and do 8-bit input from the file. This command is necessary when receiving binary files from 8-bit-byte systems, such as most microcomputers.

AUTO Equivalent to SEVEN for incoming files, and for outgoing files means to use EIGHT if the DEC-20 file bytesize (as shown by the Exec VDIR command) is 8, otherwise use SEVEN. The default is AUTO.

The DEC-20 can send any mixture of file types in the correct way automatically, but you *must* set the file bytesize to 8 for any incoming 8-bit binary files, and to AUTO (i.e. 7) for any incoming text files or DEC-20 binary files.

NAMING UNTRANSLATED *or* NORMAL-FORM
If NORMAL-FORM the names of incoming or outgoing files will be converted to contain only uppercase letters, digits, and at most one period; any other characters will be translated to "X". If UNTRANSLATED, filenames will be sent and used literally. UNTRANSLATED is the default.

SET IBM

Syntax: `SET IBM ON` *or* `OFF`

SET IBM is really a predefined SET macro rather than a "hardwired" SET command; it can be redefined or undefined (see DEFINE); as distributed from Columbia, KERMIT-20 defines IBM to be "parity mark, handshake XON, duplex half".

SET IBM should be used when running KERMIT-20 in local mode, connected to an IBM or similar mainframe. If you have redefined the SET IBM macro, then your parameters will be used instead.

SET ITS-BINARY

Syntax: `SET ITS-BINARY ON` *or* `OFF`

Specify whether ITS-Binary file headers are to be recognized or ignored. By default, they are recognized. ITS binary format is a way (devised at MIT) of storing foreign 8-bit binary data on a 36-bit machine to allow automatic recognition of these files when sending them out again, so that you don't have to depend on the file byte size, or to issue explicit SET FILE BYTESIZE commands to KERMIT.

An ITS format binary file contains the sixbit characters "DSK8" left-adjusted in the first 36-bit word. If ITS-BINARY is ON, then KERMIT-20 will send any file starting with this "header word" using 8-bit input from the file even if the file bytesize is not 8, and will not send the header word itself. KERMIT-20 will also store any incoming file that begins with that header word using 8-bit bytesize, again discarding the header word itself. If ITS-BINARY is OFF, then the header word, if any, will be sent or kept, and i/o will be according to the setting of FILE BYTESIZE.

This facility is provided for compatibility with the file formats used on certain public-access CP/M libraries.

SET LINE

Syntax: `SET LINE` [*octal-number*]

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

```
    CONNECT 12
```

is equivalent to

```
    SET LINE 12
    CONNECT
```

If you type SET LINE with no number argument, you will deassign any previous assigned line and revert to remote mode.

The SHOW LINE command will display the currently selected communication line and its charactistics, including parity, duplex, handshake, flow control, thé speed if known, whether carrier is present (if it is a modem-controlled line), and whether KERMIT-20 is in local or remote mode.

SET RECEIVE

In addition to the full complement of SET RECEIVE commands described in the main part of the manual, you may also SET RECEIVE SERVER-TIMEOUT to a value between 0 and 94. This specifies the number of seconds between timeouts during server command wait, 0 specifies that no timeouts should occur during server command wait. When a KERMIT server times out, it sends a NAK packet. Some systems cannot clear piled-up NAKs from their input buffers; if you're using such a system to communicate with a KERMIT-20 server, and you expect to be leaving the server idle for long periods of time, you should use this command to turn off server command-wait timeouts.

SET SPEED

Syntax: `SET SPEED` *n*

Set the baud rate of the currently selected communication to *n*, the decimal baud rate, for instance 300, 1200, 4800. When operating in local mode, it may be necessary to issue this command in order to enable BREAK simulation.

SET TVT-BINARY

Syntax: `SET TVT-BINARY ON` *or* `OFF`

Only for users running KERMIT-20 on an ARPANET DEC-20, signed on to an ARPANET virtual terminal (TVT) from another host or through an ARPANET TAC. SET TVT ON causes KERMIT-20 to negotiate binary mode (8-bit) communication with the ARPANET during file transfer. Without this command, file transfer through a TVT would not work in most cases.

TVT-BINARY is OFF by default. If you normally use KERMIT-20 through the ARPAnet, you can put the command SET TVT-BINARY ON into your `KERMIT.INI` file.

CAUTION: This facility requires certain features in the Release 5 TOPS-20 ARPANET monitor, which may not be present in releases distributed by DEC. See the KERMIT-20 source code for details.

The DEFINE Command

Syntax: `DEFINE` *macroname* [*set-option* [, *set-option* [...]]]

The DEFINE command is available in KERMIT-20 for building "macros" of SET commands. The macro name can be any keyword-style character string, and the set options are anything you would type after SET in a SET command; several set options may be strung together, separated by commas. Example:

```
    define notimeout send timeout 0, receive timeout 0
```

Macro definitions may not include macro names. You can list all your macros and their definitions with the SHOW MACROS command. You can list a particular macro definition with HELP SET *macroname*.

The SHOW Command

Syntax: `SHOW` [*option*]

The SHOW command displays various information:

DAYTIME	Current date, time, phase of moon.
DEBUGGING	Debugging mode in effect, if any.
FILE-INFO	Byte size for DEC-20 file i/o, incomplete file disposition.
LINE	TTY line, parity, duplex, flow control, handshake, escape character, speed (if known), and session loggin information. Note that before release 6.0 of TOPS-20, the DEC-20 does not keep a record of the actual baud rate of a modem-controlled or "remote" TTY line.
MACROS	Definitions for SET macros.
PACKET-INFO	For incoming and outbound packets. Items under RECEIVE column

	show parameters for packets KERMIT-20 expects to receive, under SEND shows parameters for outgoing packets.
TIMING-INFO	Delays, retries, server NAK intervals.
VERSION	Program version of KERMIT-20. This is also displayed when KERMIT-20 is initially started.
ALL	(default) All of the above.

The STATISTICS Command

Give statistics about the most recent file transfer. For instance, here's what KERMIT-20 displayed after transmitting a short binary file, using repeated-character compression:

```
Maximum number of characters in packet:  80 received; 80 sent
Number of characters transmitted in 2 seconds
       Sent:       34            Overhead:       34
       Received:   107           Overhead:       -408
 Total received:   141           Overhead:       -374
Total characters transmitted per second:         70
Effective data rate:    2570 baud
Efficiency:             214.1667 per cent
Interpacket pause in effect: 0 sec

Timeouts: 0
NAKs:     0
```

Note that the data compression allowed the effective baud rate to exceed the actual speed of the communication line, which in this case happened to be 1200 baud. The efficiency is displayed only if the actual baud rate is known.

The LOG Command

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

Log the specified option to the specified file:

SESSION	During CONNECT, log all characters that appear on the screen to the specified file. The session log can be temporarily turned off during the remote session by typing the escape character followed by Q (for Quit logging), and turned on again by typing the escape character followed by R (for Resume logging). Default log is **SESSION.LOG** in the current directory.
TRANSACTIONS	During file transfer, log the progress of each file. The DEC-20 transaction log file looks like this:

```
KERMIT-20 Transaction Log File, Monday 27-Feb-1984
```

```
18:40:13: Opened Log: PS:<TIMREK>SAMPLE.LOG.1
18:40:31: -- Send Begins --
   8th bit prefixing: Off
   Block check type: 1
18:40:31: Opened File: PS:<SY.FDC>LOGIN.CMD.6
   Sending As "LOGIN.CMD"
   Sent: 547 7-bit bytes
18:40:34: Closed PS:<SY.FDC>LOGIN.CMD.6
18:40:34: Send Complete
18:40:50: -- Receive Begins --
   8th bit prefixing: Off
   Block check type: 1
18:40:50: Opened: PS:<TIMREK>AUTOEXEC.BAT.1
   Written: 186 7-bit bytes
18:40:51: Closed: PS:<TIMREK>AUTOEXEC.BAT.1
18:40:56: Closed Transaction Log
```

Transaction logging is recommended for long or unattended file transfers, so that you don't have to watch the screen. The log may be inspected after the transfer is complete to see what files were transferred and what errors may have occurred. Default log is `TRANSACTION.LOG` in the current directory.

DEBUGGING

Log STATES or PACKETS, as specified in the most recent SET DEBUGGING command, to the specified file. If log file not specified, then use TTY if local, or `DEBUGGING.LOG` in the current directory if remote. If no SET DEBUGGING command was previously issued, log STATES to the specified file. Also allow specification of bytesize for the log file, 7 (normal, default), or 8 (for debugging binary transfers when the parity bit is being used for data), for instance

```
LOG DEBUGGING BINARY.LOG 8
```

A 7-bit log file can be typed, printed, or examined with a text editor or searching program. An 8-bit log file can only be examined with a system utility like FILDDT. When logging packets, each packet is preceded by a timestamp, the current timeout interval (preceded by a slash), and "`R:`" or "`S:`" to indicate data being received and sent, respectively. Packet format is described in the *KERMIT Protocol Manual*.

SESSION is the default option. Thus the command "LOG" alone will cause CONNECT sessions to be logged in `SESSION.LOG` in the current directory. Any log files are closed when you EXIT or QUIT from KERMIT, and are reactivated if you CONTINUE the program. You may explicitly close a log file and terminate logging with the CLOSE command.

The CLOSE Command

Syntax: `CLOSE` [*option*]

Close the specified log file, SESSION, TRANSACTION, or DEBUGGING, and terminate logging.

6.6. Examples

Here are a few examples of the use of KERMIT-20. Text entered by the user is underlined.

Remote Operation

The following example shows use of KERMIT-20 as a server from an IBM PC. In this example, the user runs KERMIT on the PC, connects to the DEC-20, and starts KERMIT-20 in server mode. From that point on, the user need never connect to the DEC-20 again. In this example, the user gets a file from the DEC-20, works on it locally at the PC, and then sends the results back to the DEC-20. Note that the user can leave and restart KERMIT on the PC as often as desired.

```
A>kermit
Kermit-86>connect
@
@kermit
TOPS-20 KERMIT version 4.1(236)

Kermit-20>server

 Kermit Server running on DEC-20 host.  Please type your escape
 sequence to return to your local machine.  Shut down the server by
 typing the Kermit BYE command on your local machine.
^[c
Kermit-86>get foo.txt
     The transfer takes place.
Kermit-86>exit
A>
A>edit foo.txt ; (or whatever...)
A>
A>kermit
Kermit-86>send foo.txt
     The transfer takes place.
Kermit-86>bye
A>
```

The next example shows the special procedure you would have to use in order to send a mixture of text and binary files from a PC (or an 8-bit-byte system) to the DEC-20. Note that in this case, it's more convenient to avoid server mode.

```
Kermit-86>connect
@
@kermit
TOPS-20 KERMIT version 4.1(236)

Kermit-20>receive
^]c
Kermit-86>send *.txt
     Textual files are sent.

Kermit-86>connect
Kermit-20>set file bytesize 8
Kermit-20>receive
^]c
Kermit-86>send *.exe
     Binary files are sent.
```

```
Kermit-86>connect
Kermit-20>exit
@logout
^]C
Kermit-86>exit
A>
```

Local Operation

In this example, a program DIAL is used to direct an autodialer to call another computer (a DECSYSTEM-10); once the connection is made, DIAL starts KERMIT with an implicit CONNECT command for the appropriate communication line. DIAL is not part of KERMIT; if your system has an autodialer, there will be some site-specific procedure for using it.

```
@dial
Dial>dial stevens
STEVENS, 1-(201) 555-1234, baud:1200
[confirm]
Dialing your number, please hold...
Your party is waiting on TTY11:.
@
@kermit
TOPS-20 KERMIT version 4.1(236)

Kermit-20>connect 11
[KERMIT-20: Connecting over TTY11:, type <CTRL-\>C to return]

CONNECTING TO HOST SYSTEM.
Stevens T/S 7.01A(10) 20:20:04 TTY41 system 1282
Connected to Node DN87S1(101) Line # 57

Please LOGIN or ATTACH

.log 10,35
JOB 51 Stevens T/S 7.01A(10) TTY41
Password:
20:20   15-Dec-83       Thur

.r new:kermit

TOPS-10 KERMIT version 2(106)

Kermit-10>server

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

[KERMIT-20: Connection Closed.  Back at DEC-20.]
```

```
Kermit-20>set file bytesize 8
Kermit-20>get setdtr.cmd
^A for status report, ^X to cancel file, ^Z to cancel batch.
SETDTR.CMD.7 ^A
 Receiving SETDTR.CMD.7, file bytesize 8
 (repeated character compression)
 At page 1
 Files: 0, packets: 1, chars: 66
 NAKs: 0, timeouts: 0
.[OK]
Kermit-20>bye
Job 51  User F DA CRUZ [10,35]
Logged-off TTY41  at 20:22:58  on 15-Dec-83
Runtime: 0:00:01, KCS:33, Connect time: 0:02:39
Disk Reads:72, Writes:4, Blocks saved:160
....
Hangup? y
Click. Call duration was 193 seconds to area 201.
Dial>exit
```

Note the use of Control-A to get a status report during the transfer.

6.7. Installation

KERMIT-20 is built from a single MACRO-20 source file, **20KERMIT.MAC**. It requires the standard DEC-distributed tools MONSYM, MACSYM, and CMD; the following files should be in SYS: -- **MONSYM.UNV**, **MACSYM.UNV**, **MACREL.REL**, **CMD.UNV**, and **CMD.REL**. The program should work on all TOPS-20 systems as distributed, but many customizations are possible. The site manager may wish to change various default parameters on a site-wide basis; this may be done simply by changing the definitions of the desired symbols, under "subttl Definitions", and reassembling.

The most notable site dependency is the definition of "SET IBM". As distributed from Columbia KERMIT-20 defines "SET IBM" in a built-in SET macro definition as "parity mark, duplex half, handshake xon". This definition may be found at MACTAB+1, near the end of the impure data section. It may be changed or deleted, and the program reassembled.

Sites that do not have ARPANET may wish to delete the TVT-BINARY entries from SET command tables, SETABL and SETHLP.

7. VAX/VMS KERMIT

Authors:	Bob McQueen, Nick Bush, Stevens Institute of Technology
Language:	Bliss-32, Common Bliss
Version:	2.0
Date:	November 1983

KERMIT for the Digital Equipment Corporation VAX/VMS system is called "KERMIT-32" since the VAX is DEC's 32-bit line of computers. KERMIT-32 can be run from **SYS$SYSTEM:**. It will prompt for input from **SYS$COMMAND:**.

Kermit-32 can be run in either local or remote modes. In remote mode, transfers take place over the controlling terminal line. Ususally, Kermit-32 is used in remote mode as a "server", meaning that it will accept commands from the other Kermit. In local mode, Kermit-32 will perform transfers over a terminal line other than the controlling terminal. In local mode, Kermit-32 is capable of giving commands to a "server" Kermit. Note that in order to use Kermit-32 in local mode, the protection code for the terminal to be used must allow the user access. This is set by the system manager. Kermit-32 is put into local mode by using the SET LINE TTcnn: command.

Currently, VMS Kermit does not allocate the terminal line you are using for CONNECT or transfers. Therefore, when going between CONNECT and SEND or RECEIVE, VMS hangs up the phone for you. This is easily solved by using the DCL ALLOCATE command to allocate the terminal line before entering Kermit.

VMS KERMIT implements a large subset of "ideal" KERMIT, for both remote and local operation.

Here is a summary of the commands of KERMIT-32:

CONNECT [*dev:*]
: The CONNECT command will allow you to connect as a virtual terminal over the line that was specified by the SET LINE command, or to the terminal line specified in the command. The terminal line must be one which is accessible to the users process. This means tnat the applicable protection code for the terminal must have been set to allow your process to access it (done by the system manager). The format of the CONNECT command is

```
Kermit-32>CONNECT
```

or

```
Kermit-32>CONNECT TTcn:
```

where **TT***cn*: is the terminal line name to be used.

HELP [*keyword* [*keyword*...]]
: Give VMS-style help on KERMIT commands.

EXIT, QUITExit from Kermit-32.

RECEIVE The RECEIVE command is used to put Kermit-32 into remote mode waiting for a single file transfer transaction, or to have a local Kermit-32 request a file from the remote Kermit. If no file specification is given, Kermit-32 will wait for a file transfer initialization sequence from the other Kermit. This is most useful if the other Kermit does not support local server commands. In order for a file specification to be given, Kermit-32 must be running as a local Kermit (i.e. a SET LINE command must have been done). Kermit-32 will then request the other

Kermit (which must be running in server mode) to transfer the specified file (or set of files) to Kermit-32. The file specification must be in the format of the system on which the server Kermit is running. The format of the command is:

```
Kermit-32>RECEIVE
```

or

```
Kermit-32>RECEIVE file-specification
```

where "file-specification" is any valid file specification on the system on which the server Kermit is running.

GET *filespec*
: This command is identical to the RECEIVE *filespec* command. It is now the preferred command to cause the other Kermit (when running in server mode) to transmit a file to Kermit-32.

BYE
: This command will cause Kermit-32 (when in local mode) to tell the other Kermit (which should be in server mode) to exit from Kermit and, if applicable, terminate its job (or process, etc.). When Kermit-32 receives the acknowledgement that this is being done, it will exit to VMS.

FINISH
: This command will cause Kermit-32 (when in local mode) to tell the other Kermit (which should be in server mode) to exit from Kermit. After receiving the acknowledgement that this is being done, Kermit-32 will prompt for another command.

LOGOUT
: This command will cause Kermit-32 (when in local mode) to tell the other Kermit (which should be in server mode) to exit from Kermit and, if applicable, terminate its job (or process, etc.). When Kermit-32 receives the acknowledgement that this is being done, it will prompt for another command.

SEND
: The SEND command will allow you to send a file(s) to the other Kermit. The SEND command will allow file wild card processing as is found in VMS. If Kermit-32 is running in remote mode, the file will be sent on the controlling terminal line after waiting the number of seconds specified by the SET DELAY command. This gives the user time to escape back to the other Kermit and issue a receive command. If Kermit-32 is running in local mode, the file will be sent immediately on the terminal line specified by the SET LINE command. The command format is:

```
Kermit-32>SEND file-specification
```

Where "file-specification" is any normal VAX/VMS file specification.

SERVER
: This command will cause Kermit-32 to enter server mode. The other Kermit can then issue server commands to send and receive files without having to give SEND or RECEIVE commands to Kermit-32. Kermit-32 may be put into SERVER mode while running as either a remote Kermit (transmitting over the controlling terminal line), or as a local Kermit (transmitting over a terminal specified by a SET LINE command). Note that in order to correctly receive binary files while in SERVER mode, a SET FILETYPE BINARY must be done first. At this time there is no way for Kermit-32 to determine whether an incoming file is ASCII or binary. The format of the command is:

```
Kermit-32>SERVER
```

STATUS
: The current status of Kermit-32 will be displayed. This includes the number of characters that have been sent and received from the remote Kermit. Also included is an estimate of the effective baud rate of

The SET Command

The SET command is used to set various parameters in Kermit.

BLOCK_CHECK_TYPE*keyword*

where keyword is one of:

```
1_CHARACTER_CHECKSUM  or ONE_CHARACTER_CHECKSUM
2_CHARACTER_CHECKSUM  or TWO_CHARACTER_CHECKSUM
3_CHARACTER_CRC_CCITT or THREE_CHARACTER_CRC_CCITT
```

DEBUGGING

The SET DEBUGGING command is used to set the debug type out on the user's terminal. The command will accept either the keywords ON or OFF. Kermit-32 can only do debugging type out when running as a local Kermit (SET LINE command done). This is because the debugging type out would interfere with the file transfer if it were sent to the controlling terminal line in remote mode.

```
Kermit-32>SET DEBUGGING state
```

Where state is either 'ON' or 'OFF'.

DELAY

The DELAY parameter is the number of seconds to wait before sending data after a SEND command is given. This is used when Kermit-32 is running in remote mode to allow the user time to escape back to the other Kermit and give a RECEIVE command.

```
Kermit-32>SET DELAY number-of-seconds
```

Where number of seconds is the (decimal) number of seconds to wait before sending data.

ESCAPE

This command will set the escape character for the CONNECT processing. The command will take the octal value of the character to use as the escape character. This is the character which is used to "escape" back to Kermit-32 after using the CONNECT command. It defaults to CTRL-] (octal 35). It is usually a good idea to set this character to something which is not used (or at least not used very much) on the system being to which Kermit-32 is CONNECTing.

```
Kermit-32>SET ESCAPE octal-character-value
```

where *octal-character-value* is the ASCII value of the character to use as the escape character (in octal).

FILE_TYPE

This command will set the file type that Kermit is receiving. A file type of ASCII should be used to receive text files which are to be used as text files on the VMS system. The file type BINARY should be used for binary files, such as CP/M .COM files, which need to be kept in a format that allows the file to be returned without any changes.

```
Kermit-32>SET FILE_TYPE type
```

where *type* is one of:

ASCII	File type ASCII is for text files.
BINARY	File type BINARY is for non-text files. Note that binary files which are generated on a VMS system cannot be transferred to another VMS system without losing file attributes. This means

that (for example), an .EXE file cannot be transmitted with Kermit-32. (This problem should be resolved in a future version of Kermit).

IBM_MODE

For communicating with IBM mainframes; sets parity MARK, handshake XON, and local echo during CONNECT.

Kermit-32>SET IBM_MODE *keyword*

where *keyword* is either ON or OFF.

INCOMPLETE_FILE_DISPOSITION

The SET INCOMPLETE_FILE_DISPOSITION allows the user to determine what is done with a file that is not completely received.

Kermit-32>SET INCOMPLETE_FILE_DISPOSITION *keyword*

where *keyword* is either DISCARD or KEEP.

LINE

This will set the terminal line that you are using. The terminal line must be one which is accessible to the user's process. This means that the applicable protection code for the terminal must have been set to allow your process to access it (done by the system manager). You should also ALLOCATE the line from DCL before giving this command.

Kermit-32>SET LINE *device*:

The device must be a terminal line (e.g. TTA0:).

LOCAL_ECHO

The SET LOCAL_ECHO command specifies whether characters should be echoed locally when CONNECTing to another system.

Kermit-32>SET LOCAL_ECHO *keyword*

where keyword is either ON (local echo) or OFF (full duplex, normal case).

MESSAGE

This command sets the type of typeout Kermit-32 will do during transfers in local mode. Kermit-32 can type out the file specification being transferred, the packet numbers being sent an received, both or neither. The default is to type file specifications but not packet numbers.

Kermit-32>SET MESSAGE *type keyword*

Where type is either FILE or PACKET, and keyword is either ON or OFF.

PARITY

This command determines the type of parity to use on the transmission line. Kermit-32 normally uses characters which consist of eight data bits with no parity bit.

Kermit-32>SET PARITY *keyword*

where keyword is NONE (default), MARK, SPACE, EVEN, or ODD. If any parity other than NONE is specified, 8th-bit-prefixing will be requested for transmission of binary files.

RETRY

This command sets the maximum number of times Kermit-32 should try to send a specific packet.

Kermit-32>SET RETRY *keyword number* where *keyword* is either INITIAL_CONNECTION (for initial connection packet) or PACKET (for all other packets), and *number* is the decimal number of retries to attempt.

RECEIVE

It is possible to set various parameters associated with the receiving of the data from the remote Kermit. SET RECEIVE will enable you to set the various receive parameters.

PACKET_LENGTH

This will set the receive packet length, between 10 and 96. The default value is 80.

```
Kermit-32>SET REC PACKET_LEN 60
```

PADDING

This command will set the number of padding characters that will be sent to the other Kermit. The default value is 0.

```
Kermit-32>SET RECEIVE PADDING n
```

Where *n* is the decimal number of padding characters to use.

PADCHAR

This parameter is the padding character that is sent to the remote Kermit. The parameter must be an octal number in the range of 0 to 37 or 177. All other values are illegal. The default value is 0 (an ASCII NUL).

```
Kermit-32>SET RECEIVE PADCHAR nnn
```

where *nnn* is the ASCII value of the character to be used as a pad character (in octal).

START_OF_PACKET

This command will set the start of packet character for Kermit. The start of packet character must be in the range of 0 to 36 octal. The default value is 1 (ASCII SOH, CTRL-A). This value should only be changed if absolutely necessary. It must be set the same in both Kermits.

```
Kermit-32>SET REC START_OF_PACK 3
```

TIMEOUT

This will set the number of seconds before Kermit-32 will time out the attempt to receive a message. This time out is used to handle transmission errors which totally lose a message. The default value is 15 seconds.

```
Kermit-32>SET RECEIVE TIMEOUT n
```

where *n* is the number of seconds to wait for a message (in decimal).

END_OF_LINE

This will set the end of line character the Kermit-32 expects to receive from the remote Kermit. This is the character which terminates a packet. The default value is 15 (ASCII CR, CTRL-M).

```
Kermit-32>SET REC END_OF_LINE nnn
```

where *nnn* is the ASCII value of the character to use for the end of line character (in octal).

QUOTE — This will set the quoting character that Kermit-32 will expect on incoming messages. This is the character used to quote control characters. The default value is 43 (ASCII "#").

```
Kermit-32>SET RECEIVE QUOTE nnn
```

where *nnn* is the ASCII value of the quoting character (in octal).

SEND

It is possible to set various parameters associated with the receiving of the data from the remote Kermit. SET SEND will enable you to set the various SEND parameters. These parameters should not normally be set, since as part of the transfer initialization process the two Kermit's exchange their RECEIVE parameters. The capability of setting these parameters is provided so that the transfer initialization can be completed even if the default parameters are not correct.

PACKET_LENGTH — This will set the SEND packet length. The value for this parameter must be between 10 and 96. Packet lengths outside of this range are illegal. The default value is 80.

PADDING — This command will set the number of padding characters that will be sent to the other Kermit. The default value is 0.

PADCHAR — This parameter is the padding character that is sent to the remote Kermit. The parameter must be an octal number in the range of 0 to 37 or 177. All other values are illegal. The default value is 0 (an ASCII NUL).

START_OF_PACKET — This command will set the start of packet character for Kermit. The start of packet character must be in the range of 0 to 36 octal. The default value is 1 (ASCII SOH, CTRL-A). This value should only be changed if absolutely necessary. It must be set the same in both Kermit's.

TIMEOUT — This will set the number of seconds before Kermit-32 will time out a message it has sent to the other Kermit. message. This time out is used to handle transmission errors which totally lose a message. The default value is 15 seconds.

END_OF_LINE — This will set the end of line character the Kermit-32 will send to the remote Kermit. This is the character which terminates a packet. The default value is 15 (ASCII CR, CTRL-M).

QUOTE — This will set the quoting character that Kermit-32 will expect on incoming messages. This is the character used to quote control characters. The default value is 43 (ASCII "#").

The SHOW Command

The SHOW command will allow you to show the various parameters that are set with the SET command.

ALL	The SHOW ALL command will cause all of the parameters to be listed.
BLOCK_CHECK_TYPE	This command will type out what type of block check is being requested.
COMMUNICATIONS	This command will type out the communcations line related parameters. This includes the terminal line being used, the parity type, etc.
DEBUGGING	The SHOW DEBUGGING command will print the state of the debugging flag.
DELAY	This will display the number of seconds delay that Kermit will use before attempting to send or receive a file.
ESCAPE	This will display the current escape character for the CONNECT processing.
FILE_PARAMETERS	This will display the parameters related to files being used. This includes the file type and the incomplete file disposition.
FILE_TYPE	This will display the current file type that is used in sending the file to or receiving the from the micro computer.
INCOMPLETE_FILE_DISPOSITION	This will display the disposition of incompletely received files.
LOCAL_ECHO	This will display the status of the local echo flag.
PACKET	This will display the current settings of the send and receive packet parameters.
PARITY	This will display the current parity setting.
SEND	All of the send parameters will be displayed on the user's terminal.
RECEIVE	The current values of the RECEIVE parameters will be displayed on the user's terminal. Only the parmeters that can be set will be displayed.
RETRY	This command will show the maximum retry attempts that Kermit will attempt to send a message the remote.

8. IBM VM/CMS KERMIT

Author:	Daphne Tzoar, Columbia University
Version:	(*unnumbered*)
Date:	February 1983

Written in IBM 370 assembly language to run under VM/CMS on IBM 370-series mainframes (System/370, 303x, 43xx, 308x, ...). These are half duplex systems; the communication line must "turn around" before any data can be sent to it. The fact that a packet has been received from the IBM system is no guarantee that it is ready for a reply. Thus any Kermit talking to this system must wait for the line turnaround character (XON) before transmitting the next character.

IBM systems talk to their terminals through a communications front end (IBM 3705, 3725, COMTEN 3670, etc). These front ends generally insist on using the 8th bit of each character for parity. This means that binary files (files containing other than ordinary letters, digits, punctuation, carriage returns, tabs, and so forth) can *not* be correctly sent or received by these systems with Kermit (protocol version 1).

The IBM system under VM/CMS is unable to interrupt a read on its "console". This means that the IBM version of Kermit cannot timeout. The only way to "timeout" CMS Kermit is from the other side -- typing a carriage return to the micro's Kermit causing it to retransmit its last packet, or an automatic timeout as provided by Kermit-20. For this reason, CMS Kermit waits ten seconds before sending its first packet when sending files from VM/CMS. This gives the user sufficient time to return to the local Kermit and issue the Receive command. Otherwise, a protocol deadlock would arise requiring manual intervention by the user.

Also, VM/CMS stores files as records rather byte streams. VM/CMS Kermit has to worry about assembling incoming data packets into records and stripping CRLFs from incoming lines, and also appending CRLFs to -- and stripping trailing blanks from -- outgoing records.

The VM/CMS file specification is in the form

```
    FILENAME FILETYPE FILEMODE
```

(abbreviated FN FT FM). FM is equivalent to a device specification on DEC or microcomputer systems (**FN FT FM** would translate to **FM:FN.FT**). FILENAME and FILETYPE are at most 8 characters in length, each, and FILEMODE at most 2. When FILEMODE is omitted from a filespec, the user's own disk is assumed. Kermit-CMS sends only FILENAME and FILETYPE, and converts the intervening blank to a period for compatibility with most other operating systems. Kermit-CMS Commands:

SEND *fn ft* [*fm*]

> Send the specified file(s), using * or % as the wildcard characters (* will match any number of characters while % matches only one). Kermit-CMS assumes the file is located on the A disk, and sets the filemode to A1. If, however, the file is located on a different disk, the filemode must be cited. Also, note that if you use * for the filemode, Kermit-CMS will send only the first file that matches. Examples:
>
> > The command **SEND CEN SPSS** will send CEN SPSS A1. To send the same file located on your B disk, you must specify: **SEND CEN SPSS B**. **SEND * FORTRAN** will send all fortran files on your A disk. **SEND ABC% EXEC** will send all exec files with a four letter filename beginning with

ABC. If you have the file PLOT SAS on your A disk and your B disk, **SEND PLOT SAS *** will send PLOT SAS A1.

RECEIVE [*fn ft* [*fm*]]
Receive the file(s) sent from the other Kermit. If a file specification is not included, Kermit-CMS will use the name(s) provided by the other Kermit. Use the file specification to indicate a different filename or a disk other than the A disk (in this case, the file name and type must also be supplied or **= = FM** can be used.) Examples:

To receive files using the filename(s) sent by the micro, use: **RECEIVE**.
To save a file under a different name, specify: **RECEIVE ABC FORTRAN**.
To save the file under the same name but on the B disk, specify: **RECEIVE ABC FORTRAN B**, or **RECEIVE = = B**.

SET *parameter value*
Set the parameter to the specified value. Legal Set commands are:

RECFM *option*
Denotes the record format to be used when creating the file. Only fixed and variable length records are allowed, where variable is the default. Indicate the desired record format by either an F (fixed) or a V (variable).

LRECL *decimal-number*
Indicates the logical record length. The default is 80, and the maximum allowed is 256.

QUOTE *decimal-number*
The ASCII value of the control prefix character you wish to use in place of the default (#). It must be a single, printable character from among the following: 33-62, 96, or 123-126 (decimal).

END *decimal-number*
Indicates the ASCII value of the end-of-line character you choose to send. The default is CR (ASCII 13 decimal), but can be set to any two digit number between 00 and 31 (decimal).

PAC *decimal-number*
Allows the user to specify the packet size the micro should use when sending to Kermit-CMS. The range is 26-94, where 94 is the default.

SHOW *parameter*
Displays the current value of any variable that can be changed via the SET command.

STATUS Returns the status of the previous execution of Kermit-CMS. Therefore, STATUS will either display the message "Kermit completed successfully", or the last error encountered prior to aborting.

CMS Issues a CMS command from within Kermit-CMS.

CP Issues a CP command from within Kermit-CMS.

? Lists all legal Kermit-CMS commands.

This is a list of other salient facts about Kermit-CMS:

1. The commands are supplied with a help option, so a question mark can be typed to get the appropriate format or a list of options. The question mark, however, must be followed by a carriage return; Kermit-CMS will respond and display the prompt again. For instance, SET ? will list all valid options for the SET command.

2. When receiving files, if the record format is fixed, any record longer than the logical record length will be truncated. If the record format is variable, the record length can be as high as 256. For sending files, the maximum record length is 256.

3. Before connecting to the IBM mainframe from other systems (like the various microcomputer and PC Kermits, DEC-20 Kermit, etc), you should set the IBM flag ON so that echoing, parity, and handshaking can be done the way the IBM system likes.

4. Note that "(" and ")" act as word separators on the input line. Therefore, if you try to set the quote character to "(*" or "*(", for example, only the first character will be used.

5. Since some Kermits do not send an error packet when they "abort", Kermit-CMS does not always know the micro has stopped sending it information. Therefore, when you connect back to the IBM, Kermit-CMS may still be sending packets (they will appear on the screen). The user must hit a carriage return until Kermit-CMS has sent the maximum number of packets allowed and aborts. The error message, however, will not indicate that communication stopped because the micro aborted, but rather that no start of header character was found.

6. The minimum send packet size Kermit-CMS will allow is 26. This is necessary to avoid an error while sending the filename or an error packet. If the micro tries to set the value to be less than 26, Kermit-CMS will immediately abort with an error of "Bad send-packet size."

7. While the IBM's communication front end processor translates all incoming characters from ASCII terminals to EBCDIC, Kermit-CMS translates the data it reads back to ASCII (characters not representable in ASCII are replaced by a null). Not only is it easier to work with ASCII characters, but it makes things more consistent throughout the many versions of Kermit. When the packets are sent to the micro, Kermit-CMS converts all data back to EBCDIC. The ASCII to EBCDIC translation table can be found in the Appendix.

8. If a transfer becomes stuck, you can CONNECT back to the CMS system and type a lot of carriage returns -- each one will cause KERMIT-CMS to retransmit the current packet, until the retransmission limit is reached, and you will be back at "KERMIT-CMS>" command level.

9. Kermit-CMS supplies the micro and the user with numerous error messages. If the execution must be abnormally terminated, an error packet is sent to the micro before Kermit-CMS stops. The same message can be retrieved via the STATUS command when Kermit-CMS returns and displays the prompt. If Kermit-CMS aborted because the maximum amount of retries was exceeded (20 on initialization packets and 5 on others), the error message will display the most recent error (i.e. the last NAK Kermit-CMS encountered). If execution stops because the micro gave up, the error message will convey that to the user, but it is the micro's responsibility to pinpoint the error. The messages Kermit-CMS gives are as follows:

"Bad send-packet size"
: Sent when the micro attempts to set its receive buffer size to a value that is less than 26 (the minimum that Kermit-CMS will accept) or larger than 94, the maximum. It will also occur if Kermit-CMS tries to send a packet that is larger than the maximum specified.

"Bad message number"
: This and the following messages flag inconsistencies in a Kermit packet.

"Illegal packet type" -- This can be caused by sending server commands.
"Unrecognized State"
"No SOH encountered"
"Bad Checksum"
"Bad character count"
"Micro sent a NAK"
"Lost a packet"
"Micro aborted"
: The micro abnormally terminated the transfer.

"Illegal file name"
: When receiving the name of the file from the micro, Kermit-CMS expects it to be in the format 'filename.filetype'. If the filename, filetype, or dot is missing, Kermit-CMS will reject (NAK) the packet. Also, if either the filename or filetype exceeds eight characters, it will be truncated.

"Invalid lrecl"
: Kermit-CMS will abort on any file-system error it encounters when reading from the file it is to send. It can only send files with variable or fixed length record formats, therefore, Wylbur Edit or Packed format files will cause an error.

"Permanent I/O error"
: This signifies a permanent I/O error that occured when reading from an existing file. Execution is aborted immediately.

"Disk is read-only"
: Attempt to write on a read-only disk.

"Recfm conflict"
: If a filename conflict arises, Kermit-CMS will append the received file to the existing one, provided the record formats of the two are the same. Otherwise, this error will cause a halt of the execution.

"Disk is full"
: Refers to any error regarding limitations on a user's storage space. Most likely, it signifies that the receiving disk is full, but the error can also mean that the maximum number of files allowed has been reached, or virtual storage capacity has been exceeded, and so on.

"Err allocating space"
: Kermit-CMS keeps a table of all files it has sent to the micro, allocating extra space if more than ten files are sent at one time. If there is an error obtaining more space, Kermit-CMS will abort with this error.

Work on VM/CMS Kermit continues. Planned future enhancements include:

1. 8-bit quoting, to allow binary files to pass through communication front ends that insist on using the 8th bit for parity.

2. Ability to act as a Kermit Server.

3. Ability to SET LINE, so that Kermit-CMS can be used as a local Kermit, connecting to a remote host over another communication port.

9. UNIX KERMIT

Authors:	Bill Catchings, Bob Cattani, Chris Maio, Columbia University with fixes and contributions from many others.
Documentation:	Walter Underwood, Ford Aerospace (Palo Alto, CA)
Version:	(unnumbered)
Date:	October 1983

A sample, working implementation of the Kermit "kernel" was written in the C language, and widely distributed in the *Kermit Protocol Manual*. This kernel was intended merely to illustrate the protocol, and did not include a "user interface", nor some of the fancy features like server support, 8-bit quoting, file warning, timeouts, etc. Several sites have added the necessary trappings to make this a production version of Kermit, usually under the UNIX operating system.

The keyword style of user/program interaction favored by Kermit (program types prompt, user types command followed by operands, program types another prompt, etc) is contrary to the UNIX style, so UNIX implementations have a style more familiar to UNIX users. C versions of Kermit are running successfully on VAX and PDP-11 UNIX systems, IBM 370-compatible mainframes under Amdahl UTS, and the SUN Microsystems MC68000-based and other workstations.

UNIX filespecs are of the form

```
    dir1/dir2/dir3/ ... /filename
```

where the tokens delimited by slashes form a *path name*, and by convention are each limited to 14 characters in length. The final token in a path is the actual file name. By convention, it is of the form **name.type**, but there is nothing special about the dot separating name and type; to UNIX it's just another character, and there may be many dots in a filename.

In the tradition of UNIX, here's the UNIX KERMIT "man page".

NAME kermit - file transfer, virtual terminal over tty link

SYNOPSIS kermit c[lbe] [line] [baud] [esc]

kermit r[ddilb] [line] [baud]

kermit s[ddilb] [line] [baud] file ...

DESCRIPTION Kermit provides reliable file transfer and primitive virtual terminal communication between machines. It has been implemented on many different computers, including microprocessors (see below). The files transferred may be arbitrary ASCII data (7-bit characters) and may be of any length. The file transfer protocol uses small (96 character) checksummed packets, with ACK/NACK responses and timeouts. Kermit currently uses a five second timeout and ten retries.

The arguments to kermit are a set of flags (no spaces between the flags), three optional args (which, if included, must be in the same order as the flags which indicate their presence), and, if this is a Send operation a list of one or more files. (It is similar in some way to the tar command structure).

Kermit has three modes, Connect, Send, and Receive. The first is for a virtual terminal connection, the other two for file transfer. These modes are specified by the first flag, which should be c, s, or r, respectively. Exactly one mode must be specified.

The d flag (debug) makes kermit a bit more verbose. The states kermit goes through are printed along with other traces of its operation. A second d flag will cause kermit to give an even more detailed trace.

The i flag (image) allows slightly more efficient file transfer between Unix machines. Normally (on Kermits defined to run on Unix systems) newline is mapped to CRLF on output, CR's are discarded on input, and bytes are masked to 7 bits.. If this is set, no mapping is done on newlines, and all eight bits of each byte are sent or received. This is the default for non-Unix kermits.

The l flag (line) specifies the tty line that kermit should use to communicate with the other machine. This is specified as a regular filename, like "/dev/ttyh1". If no l option is specified, standard input is used and kermit assumes it is running on the remote host (ie. NOT the machine to which your terminal is attached).

The b flag (baud) sets the baud rate on the line specified by the l flag. No changes are made if the b flag is not used. Legal speeds are: 110, 150, 300, 1200, 2400, 4800, 9600. Note that this version of kermit supports this option on Unix systems only.

The e flag (escape) allows the user to set the first character of the two character escape sequence for Connect mode. When the escape character is typed, kermit will hold it and wait for the next character. If the next character is c or C, kermit will close the connection with the remote host. If the second character is the same as the escape character, the escape character itself is passed. Any character other than these two results in a bell being sent to the user's terminal and no characters passwd to the remote host. All other typed characters are passed through unchanged. The default escape character is '↑'.

The file arguments are only meaningful to a Send kermit. The Receiving kermit will attempt to store the file with the same name that was used to send it. Unix kermits normally convert outgoing file names to uppercase and incoming ones to lower case (see the f flag). If a filename contains a slash (/) all outgoing kermits will strip off the leading part of the name through the last slash.

EXAMPLE

For this example we will assume two Unix machines. We are logged onto "unixa" (the local machine), and want to communicate with "unixb" (the remote machine). There is a modem on "/dev/tty03".

We want to connect to "unixb", then transfer "file1" to that machine.

We type:

```
kermit clb /dev/tty03 1200
```

Kermit answers:

```
Kermit: connected...
```

Now we dial the remote machine and connect the modem. Anything typed on the terminal will be sent to the remote machine and any output from that machine will be displayed on our terminal. We hit

RETURN, get a "login:" prompt and login.

Now we need to start a kermit on the remote machine so that we can send the file over. First we start up the remote, (in this case receiving) kermit, then the local, (sending) one. Remember that we are talking to unixb right now.

We type:

```
     kermit r
```

(there is now a Receive kermit on unixb)

We type ^ (the escape character) and then the letter c to kill the local (Connecting) kermit: ^c

Kermit answers:

```
     Kermit: disconnected.
```

We type:

```
     kermit slb /dev/tty03 1200 file1
```

Kermit answers:

```
     Sending file1 as FILE1
```

When the transmission is finished, kermit will type either "Send complete", or "Send failed.", depending on the success of the transfer. If we now wanted to transfer a file from unixb (remote) to unixa (local), we would use these commands:

```
     kermit clb /dev/tty03 1200
       (connected to unixb)
     kermit s file9
       ^c (up-arrow c not control-c)
          (talking to unixa again)
     kermit rl /dev/tty03 1200
```

After all the transfers were done, we should connect again, log off of unixb, kill the Connect kermit and hang up the phone.

FEATURES

Kermit can interact strangely with the tty driver. In particular, a tty with "hangup on last close" set (stty hup), will reset to 300 Baud between kermit commands. It will also hang up a modem at that time. It is better to run with "stty -hup", and use "stty 0" to explicitly hang up the modem.

The KERMIT Protocol uses only printing ASCII characters, Ctrl-A, and CRLF. Ctrl-S/Ctrl-Q flow control can be used "underneath" the Kermit protocol (TANDEM line discipline on Berkeley Unix).

Since BREAK is not an ASCII character, kermit cannot send a BREAK to the remote machine. On some systems, a BREAK will be read as a NUL.

This kermit does have timeouts when run under Unix, so the protocol is stable when communicating with "dumb" kermits (that don't have timeouts).

DIAGNOSTICS

cannot open device
The file named in the line argument did not exist or had the wrong

permissions.

bad line speed
The baud argument was not a legal speed.

Could not create file
A Receive kermit could not create the file being sent to it.

nothing to connect to
A Connect kermit was started without a line argument.

10. MS-DOS KERMIT

Program:	Daphne Tzoar and Jeff Damens, Columbia University; contributions by many others.
Language:	Microsoft Macro Assembler (MASM)
Documentation:	Frank da Cruz, Columbia University; Herm Fischer, Litton Data Systems, Van Nuys CA.
Version:	2.26
Date:	July 1984

Kermit-MS Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
^X/^Y interruption:	Yes
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	Yes
Alternate block checks:	Yes
Terminal emulation:	Yes
Communication settings:	Yes
Transmit BREAK:	Yes
IBM mainframe communication:	Yes
Transaction logging:	No
Session logging:	Yes
Raw transmit:	No
Act as server:	Yes
Talk to server:	Yes
Advanced server functions:	No
Advanced commands for servers:	Yes
Local file management:	Yes
Handle file attributes:	No
Command/init files:	Yes
Command macros:	Yes

Kermit-MS is a program that implements the KERMIT file transfer protocol for the IBM PC and several other machines using the same processor family (Intel 8088 or 8086) and operating system family (PC-DOS or MS-DOS, henceforth referred to collectively as MS-DOS, versions 1.1, 2.0, and 2.1, and thereafter). This section will describe the things you should know about the MS-DOS file system in order to make effective use of Kermit, and then it will describe the Kermit-MS program.

Version 2 of MS-DOS Kermit runs on a variety of systems, including the IBM PC and XT, the HP-150, the DEC Rainbow 100 and 100+ (MS-DOS 2.05 and above), the Wang PC, and there is a "generic" MS-DOS version. Version 1 was adapted at various stages of development to run on other systems as well, including the Heath/Zenith 100, Tandy 2000,

Victor 9000 (Sirius-1), and Seequa Chameleon, and is still available for those systems until support for them and others is added to version 2.

The program operates under version 1.1 or 2.0 and above of DOS, although some features require the functionality of 2.0. It runs in approximately 80K of memory -- over and above the memory used by DOS -- which means that your system should have at least 128K of RAM to use version 2 of MS-DOS Kermit; smaller systems may still use Version 1.

10.1. The MS-DOS File System

The features of the MS-DOS file system of greatest interest to KERMIT users are the form of the file specifications, and the distinction between pre-MS-DOS 2.0 file names and newer file names which allow directory paths.

10.1.1. File Specifications

MS-DOS 2.*x* file specifications are of the form

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

where the DEVICE is a single character identifier (for instance, A for the first floppy disk, C for the first fixed disk, D for a RAM disk emulator), PATHNAME is up to 63 characters of identifier(s) (up to 8 characters each) surrounded by reverse slashes, NAME is an identifier of up to 8 characters, and TYPE is an identifier of up to 3 characters in length. Device and pathname may be omitted. The first backslash in the pathname may be omitted if the specified path is relative to the current directory. In the path field, "." means current directory, ".." means parent directory. Some DOS implementations (like Wang) may use slash "/" rather than backslash in the path field.

Pathname is normally omitted, and cannot be specified for MS-DOS 1.x or with those commands which allow MS-DOS 1.x use. Device and directory pathnames, when omitted, default to either the user's current disk and directory, or to the current directory search path as specified in the DOS PATH environment variable, depending on the context in which the file name appears.

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

`NAME.TYPE` is normally sufficient to specify a file, and only this information is sent along by Kermit-MS with an outgoing file.

The device, path, name, and type fields may contain uppercase letters, digits, and the special characters "-" (dash), "_" (underscore), and "$" (dollar sign). (For use only among MS-DOS processors, additional filename special characters allowed are "#&!%''(){}". DOS 1.x allows others as well.). There are no imbedded or trailing spaces. Other characters may be not be included; there is no mechanism for "quoting" otherwise illegal characters in filenames. The fields of the file specification are set off from one another by the punctuation indicated above.

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

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

`*.BAS`	All files of type BAS (all BASIC source files) in the current directory.
`FOO.*`	Files of all types with name FOO.
`F*.*`	All files whose names start with F.
`F?X*.*`	All files whose names start with F and contain X in the third position, followed by zero or more characters.
`?.*`	All files whose names are exactly one character long.

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

> *Note:* Kermit-MS uses the "?" character for help while commands are being typed, so the single-character wildcard in Kermit-MS commands is "=" rather than "?". For example
>
> ```
> Kermit-MS>send =.*
> ```
>
> would send files of all types whose names were exactly one character long.

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

10.1.2. File Formats

MS-DOS systems store files as bulk collections of 8 bit bytes, with no particular differences between text, program code, and binary files. ASCII text files consist of lines separated by carriage-return-linefeed sequences (CRLFs) which conforms exactly to the way Kermit represents text files during transmission. Since a non-MS-DOS receiving system might need to make distinctions as to file type, you may need to use various SET functions on the remote system to inform it that the incoming file is of some particular (non-default) type, such as binary. In transmitting files between Kermit-MS's, regardless of file contents, the receiving MS-DOS system is equally capable of processing text, code, and data, and in fact has no knowledge of how the bytes in the file are used.

MS-DOS (unlike CP/M) is capable of pinpointing the end of file with precision by keeping a byte count in the directory, so one would expect no particular confusion in this regard. However, certain MS-DOS programs continue to use the CP/M convention of terminating a text file with a Control-Z character, and won't operate correctly unless this terminating byte is present. Therefore, Kermit-MS users should be aware of a special SET EOF option for both incoming and outbound files, described below.

Non-MS-DOS systems may well be confused by nonstandard ASCII files from Kermit-MS. Files produced by Easywriter or Word Star, for example, may need to be converted to conventional ASCII format prior to transmission by commonly available "exporter" programs. Spreadsheet or database files usually need special formatting to be meaningful to non-

MS-DOS recipients (though they can be transmitted between MS-DOS systems with Kermit-MS). Furthermore, files created by word processors (such as BLUE or Easy Writer) that store formatting data at the end of the file, after the control-Z and before physical end, will require special processing via SET EOF to strip the formatting data, lest they confuse non-MS-DOS recipients.

10.2. Program Operation

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

Upon initial startup, the program executes any commands found in the file **MSKERMIT.INI** in the current path. This initialization file may contain command macro definitions, communications settings for one or more ports, or any other Kermit-MS commands. Here is a sample **MSKERMIT.INI** file:

```
set warning on  ; Enable filename collision avoidance.
;
; Define some macros
;
define unix set local-echo off, set flow xon, set timer off
def ibm set parity odd, set local on, set handsh xon, set timer on
def modem set port 2, set baud 1200
def noisy set block-check 3, set send packet-length 40
;
; Select a port
;
set port 1      ; Select COM1 for communications,
set baud 4800   ; setting the speed to 4800 baud,
connect         ; and make a terminal connection.
```

Note that comments may be included by prefixing them with a semicolon. The program can be run in several ways.

Interactive Operation:

To run Kermit-MS interactively, invoke the program from DOS command level by typing its name. When you see the command's prompt,

```
Kermit-MS>
```

you may type Kermit commands repeatedly until you are ready to exit the program, for example:

```
A>
A>kermit

IBM PC Kermit-MS V2.26
Type ? for help

Kermit-MS>send foo.*

  informational messages about the files being sent

Kermit-MS>get bar.*

  informational messages about the files being received

Kermit-MS>exit
A>
```

During interactive operation, you may edit the command you're currently typing to erase the character most recently typed (BACKSPACE or DEL), the most recent field (CTRL-W), or the entire command (CTRL-U). In addition, you may use the help ("?") and recognition (ESC) features freely while typing Kermit-MS commands. A question mark typed at almost any point in a command produces a brief description of what is expected or possible at that point; for this reason, Kermit-MS uses "=" for the single-character match wildcard in local filenames. ESC typed at any point, even in a local filename, will cause the current field to be filled out if what you have typed so far is sufficient to identify it, and will leave you in position to type the next field (or to type a "?" to find out what the next field is); otherwise, the program will beep at you and wait for you to type further characters.

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

Summary of Kermit-MS Command Characters:

BACKSPACE	Delete the character most recently typed. May be typed repeatedly to delete backwards. You may also use DELETE, RUBOUT, or equivalent keys.
CTRL-W	Delete the most recent "word", or field, on the command line. May be typed repeatedly.
CTRL-U	Delete the entire command line.
CTRL-C	Cancel the current command and return to the "**Kermit-MS>**" prompt.
?	Type a brief message describing what you are expected to type in the current field.
ESC	If enough characters have been supplied in the current field (keyword or file name) to uniquely identify it, supply the remainder of the field and position to the next field of the command. Otherwise, sound a beep.
=	Wildcard character for matching single characters in filenames, equivalent to MS-DOS "?".

Command Line Invocation:

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

```
    A>kermit send foo.bar
```

or

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

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

Batch Operation:

Like other MS-DOS programs, Kermit-MS may be operated under batch with either command line arguments and/or TAKE files; Kermit will also run interactively if invoked from batch, but it will read commands from the keyboard and not the batch file.

10.3. Kermit-MS Commands

MS-DOS Kermit implements a large subset of the commands of "ideal" Kermit. Here's a brief summary:

BYE	to remote server.
CLOSE	log file and stop logging remote session.
CONNECT	as terminal to remote system.
DEFINE	macros of Kermit-MS commands.
DELETE	local files.
DIRECTORY	listing of local files.
DO	a macro expansion.
EXIT	from Kermit-MS.
FINISH	Shut down remote server.
GET	remote files from server.
HELP	about Kermit-MS.
LOCAL	prefix for local file management commands.
LOG	remote terminal session.
LOGOUT	remote server.
PUSH	to MS-DOS command level.
QUIT	from Kermit-MS
RECEIVE	files from remote Kermit.
REMOTE	prefix for remote file management commands.
RUN	an MS-DOS program.
SEND	files to remote Kermit.
SERVER	mode of remote operation.
SET	various parameters
SHOW	various parameters.
SPACE	inquiry.
STATUS	inquiry.
TAKE	commands from file.

The remainder of this section concentrates on the commands that have special form or meaning for MS-DOS Kermit. Not all of the following commands are necessarily available on all MS-DOS systems, and some of the commands may work somewhat differently between DOS versions.

10.3.1. Commands for File Transfer

The file transfer commands are SEND, GET, and RECEIVE.

The SEND Command

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

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

filespec1 may contain a device designator, like "A:" and the wildcard characters "*" and/or "#". The current release of Kermit-MS, however, does not allow pathnames in the SEND command file specification.

If *filespec1* contains wildcard characters then all matching files will be sent, in the same order that MS-DOS would show them in a directory listing. If *filespec1* specifies a single file, you may direct Kermit-MS to send that file with a different name, given in *filespec2*. For instance, in the command

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

filespec2 begins with the first nonblank character after *filespec1* and ends with the carriage return; thus it may contain blanks or other unusual characters that may be appropriate on the target machine. Lower case letters in *filespec2* are raised to upper case for transmission.

If a file can't be opened for read access, standard MS-DOS recovery procedures will take place. For example:

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

If you select "Abort," you will be returned to DOS.

Files will be sent with their MS-DOS filename and filetype (for instance FOO.TXT, no device or pathname). Each file is sent as is, with no conversions done on the data, except for possibly adding or deleting a terminating Control-Z character (see the SET EOF command).

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

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

^X	(Control-X) Stop sending the current file and go on to the next one, if any.
^Z	Stop sending this file, and don't send any further files.
^C	Return to Kermit-MS command level immediately without sending any kind of notification to the remote system.
^E	Like ^C, but send an Error packet to the remote Kermit in an attempt to bring it back to server or interactive command level.

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

Control-X and Control-Z send the proper protocol messages to the remote Kermit to bring it gracefully to the desired state. Control-C leaves the remote Kermit in whatever state it happens to be in. Control-E "aborts" any protocol that is taking place.

The RECEIVE Command

Syntax: RECEIVE [*filespec*]

The RECEIVE command tells Kermit-MS to receive a file or file group from the other system. Kermit-MS simply waits for the file to arrive; this command is not to be used when talking to a Kermit server (use GET for that). You should already have issued a SEND command to the remote Kermit and escaped back to Kermit-MS before issuing the RECEIVE command.

If the optional filespec is provided, store the incoming file under that name. The filespec may include a device designator, or may consist of only a device designator. The incoming file is stored on the default or specified device (current directory in DOS 2.0 and thereafter). If no name was specified, the name from the incoming file header packet is used; if that name is not a legal MS-DOS file name, Kermit-MS will delete excessive characters from it, and will change illegal characters to the letter X.

If the optional filespec was provided, but more than one file arrives, the first file will be stored under the given filespec, and the remainder will be stored under their own names, but on the specified device.

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

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

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

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

^C, ^E, and CR operate in the same way as they do during SEND.

If the incoming file has the same name as a file that already exists, and WARNING is set ON, Kermit-MS will change the incoming name (and inform you how it renamed it) so as not to obliterate the pre-existing file. If WARNING is OFF, the original file will be overwritten; if you type ^X or ^Z to interrupt the transfer, you'll either get a partial new file, or else both the old and the new file of that name will be lost, depending on SET INCOMPLETE. In any case, when WARNING is off, files with the same name as incoming files will not survive.

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

The GET Command

Syntax: `GET` *remote-filespec*

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

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

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

If more than one file arrives, only the first will be renamed.

The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. It can contain whatever wildcard or file-group notation is valid on the remote system. As files arrive, their names will be displayed on your screen, along with packet traffic statistics and status messages. You may type `^X` to request that the current incoming file be cancelled, `^Z` to request that the entire incoming batch be cancelled, and `^C` or `^E` to return immediately to the `Kermit-MS>` prompt, exactly as described for the RECEIVE command.

10.3.2. Commands for Connecting and Disconnecting

The CONNECT command connects your PC as a terminal to the remote system, so that you can start up Kermit there. The BYE, FINISH, and LOGOUT commands allow you to shut down a remote Kermit server.

BYE	When communicating with a remote KERMIT server, use the BYE command to shut down the server, log out its job, and exit from Kermit-MS to DOS.
FINISH	Like BYE, FINISH shuts down the remote server. However, FINISH does not log out the server's job. You are left at Kermit-MS prompt level so that you can connect back to the job on the remote system.
LOGOUT	The LOGOUT command is identical to the BYE command, except you will remain at Kermit-MS prompt level, rather than exit to DOS, so that you can establish another connection.

The CONNECT Command

Establish an interactive terminal connection to the system connected to the currently selected communications port (e.g. COM1 or COM2) using full duplex (remote) echoing and no parity unless otherwise specified in previous SET commands. Get back to Kermit-MS by typing the escape character followed by the letter C. The escape character is Control-] by default.

You can use the SET ESCAPE command to define a different escape character, and on some systems (including the PC and XT) you can SET BAUD to change the baud rate, and SET PORT to switch between ports.

Terminal emulation is described in greater detail in section 10.4 below.

10.3.3. Commands for File Management

Kermit-MS provides commands or managing both local and remote files.

The REMOTE Commands

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

CWD [*directory*]	Change Working Directory on the remote host. Change the default source and destination area for file transfer and management. You will be prompted for a password, which will be erased as you type it. If you do not supply a password (i.e. you type only a carriage return), the server will attempt to access the specified directory without a password. If you do not supply a directory name, your default or login directory on the remote system will be assumed.
DELETE *filespec*	Delete the specified file or files on the remote system. In response, the remote host should display a list of the files that were or were not successfully deleted.
DIRECTORY [*filespec*]	The remote system will provide a directory listing of the specified files. If no files are specified, then all files in the default area (the current working directory) will be listed.
HELP	The remote host tells what server functions it is capable of.
HOST [*command*]	Send the command to the remote system's command processor for execution.
SPACE [*directory*]	Provide a brief summary of disk usage in the specified area on the remote host. If none specified, the default or current area will be summarized.
TYPE *filespec*	Display the contents of the specified remote file or files on the screen.

The LOCAL Command

The LOCAL keyword is a prefix for a number of commands. It indicates that the specified command is to be executed on the local MS-DOS system. The LOCAL prefix may be omitted. The local commands available are:

DELETE *filespec*	Deletes the specified file or files. As in DOS, the names of the deleted files are not listed, only the message "file(s) deleted" or "file(s) not found", and if you give the command "delete *.*", Kermit-MS will prompt "Are you sure?", like DOS.
DIRECTORY [*filespec*]	Lists the names, sizes, and creation dates of files that match the

	given file specification. If no filespec is given, the command is equivalent to `DIR *.*`.
SPACE	Performs the MS-DOS CHKDSK function by running the CHKDSK program from the current path, or default disk under DOS 1.1.
RUN *filespec*	Runs the specified file, which must be in .EXE or .COM format, from the specified path or according to the value of the PATH variable if no path was included in the filespec. This command requires MS-DOS 2.0 or higher.
PUSH	Invokes an MS-DOS command processor "under" Kermit-MS, either COMMAND.COM or whatever shell you have specified with COMSPEC. When you return to Kermit-MS (for instance, by typing the MS-DOS EXIT command), you will find Kermit-MS as you left it, with all settings intact. This command only works in MS-DOS 2.0 or higher.

The local RUN command has various uses, one of which is to supplement the features of Kermit-MS. For instance, suppose there is an involved procedure that you regularly perform on a certain remote system -- this might include giving commands to a modem to dial the system, looking for a particular herald or prompt, performing a login command sequence, running a selected application, and then running Kermit to send the results back to your PC. You could write a program in the compiled language of your choice, say C or BASIC, to send the desired commands to your modem and the remote system and to look for the appropriate responses. You could put all this in a Kermit-MS TAKE command file (see below), like

```
run update.com
receive
```

The program, called UPDATE in this case, does everything up to and including starting Kermit sending from the remote system. When the program terminates, the next Kermit-MS command, "receive," is executed from the command file. When the end of the command file is reached, interactive operation is resumed.

The TAKE Command

Syntax: TAKE *filespec*

Execute Kermit commands from the specified file, which may include an explicit path; if no path is specified, the value of the PATH variable is used; if PATH has no value, then the current disk and directory are searched. The command file may include TAKE commands, but it cannot include characters to be sent to a remote host during terminal emulation (i.e. after a CONNECT command). A command file may include comments prefixed by semi-colons.

The LOG Command

Syntax: LOG *filespec*

Specifies that all characters that appear on your screen during CONNECT will be recorded in the specified file. This allows you to "capture" files from a remote system that doesn't have Kermit, as well as to record remote command typescripts. The log is closed when you EXIT from Kermit-MS or when you issue an explicit CLOSE command.

10.3.4. The SERVER Command

Kermit-MS is capable of acting as a Kermit server, providing file transfer for users coming in through one of the communication ports. The current version of Kermit-MS can send files (the user on the other end types the GET command), receive files (the user types SEND), and terminate, giving control back to the console (user types BYE).

To put Kermit-MS into server mode, first issue any desired SET commands to select and configure the desired port, and then type the SERVER command. Kermit-MS will await all further instructions from the user Kermit on the other end of the connection, which may be hardwired or connected through an autoanswer modem. For example:

```
Kermit-MS>set port 1
Kermit-MS>set baud 1200
Kermit-MS>set timer on
Kermit-MS>set warning on
Kermit-MS>server
```

10.3.5. The SET Command

Syntax: SET *parameter* [*value*]

Establish or modify various parameters for file transfer or terminal connection. You can examine their values with the SHOW command. Note that there is no "set ibm" command; IBM mainframe communication parameters may be selected with a command macro (see below). The following SET commands are available in Kermit-MS:

BAUD	Communications port line speed
BELL	Whether to beep at the end of a transaction
BLOCK-CHECK-TYPE	Level of error checking for file transfer
DEBUG	Display packet contents during file transfer
DEFAULT-DISK	Default disk drive for file i/o
DESTINATION	Default destination device for incoming files
END-OF-LINE	Packet terminator
EOF	Method for determining or marking end of file
ESCAPE	Escape character for CONNECT
FLOW-CONTROL	Enable or disable XON/XOFF
HANDSHAKE	Half-duplex line turnaround option
HEATH19	Heath/Zenith-19 terminal emulation
INCOMPLETE	What to do with an incompletely received file
KEY	Specify key redefinitions, or "keystroke macros"
LOCAL-ECHO	Specify which host does the echoing during CONNECT
PARITY	Character parity to use
PORT	Select a communications port
PROMPT	Change the "Kermit-MS>" prompt to something else
RECEIVE	Request remote Kermit to use specified parameters
REMOTE	For running Kermit-MS interactively from back port
SEND	Use the specified parameters during file transfer
TAKE-ECHO	Control echoing of commands from TAKE files
TIMER	Enable/disable timeouts during file transfer
WARNING	Specify how to handle filename collisions

The SET commands that are peculiar to MS-DOS Kermit are now described in greater detail. The others behave as in "ideal" Kermit.

SET BAUD

Syntax: **SET BAUD** *rate*

Set the speed of the currently selected terminal communications port (COM1 by default) to 300, 1200, 1800, 2400, 4800, 9600 or other common baud rate. Some implementations do not support this command. In any case, Kermit-MS leaves the current communication port settings alone unless you issue explicit SET commands to change them.

SET BELL

Syntax: **SET BELL ON** *or* **OFF**

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

SET DEBUG

Syntax: **SET DEBUG ON** *or* **OFF**

ON	Display the Kermit packet traffic on your screen during file transfer. If the debugger is loaded, transfer control to it when CTRL-C is typed. In Heath-19 terminal emulation on the IBM PC, display unusual control characters in uparrow notation.
OFF	Don't display debugging information (this is the default). If debugging was in effect, turn it off.

SET DEFAULT-DISK

Syntax: **SET DEFAULT-DISK** *x*:

Specify the default disk drive to use for file transfer, directory listings, and so forth. Equivalent to typing the DOS command for changing disks.

SET DESTINATION

Syntax: **SET DESTINATION** *device*

Specify the device for incoming files, DISK or PRINTER. SET DESTINATION PRINTER will cause incoming files to be spooled directly to the printer. The normal destination is DISK.

END-OF-LINE

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

If the remote system needs packets to be terminated by anything other than carriage return, specify the decimal value of the desired ASCII character.

SET EOF

Syntax: **SET EOF** *option*

Controls how the end of file is handled. The options are:

CTRL-Z	Append a Control-Z character to the end of an incoming file, unless it already ends with a Control-Z. Certain MS-DOS text editors and

other applications require files to be in this format. For outbound files, treat the first Control-Z as the end of file, and do not send it nor any characters following it.

NOCTRL-Z (Default) Store incoming files exactly as is, and send MS DOS files exactly as is (according to their byte count).

SET ESCAPE

Syntax: SET ESCAPE *character*

Specify the control character you want to use to "escape" from remote connections back to Kermit-MS. The default is normally ^] (Control-Rightbracket). The character is entered literally, and should normally be chosen from the ASCII control range.

SET FLOW-CONTROL

Syntax: SET FLOW-CONTROL *option*

Specify the full duplex flow control to be done on the currently selected port. The current options are XON/XOFF and NONE. The specified type of flow control will be done during both terminal emulation and file transfer. If set to XON/XOFF, HANDSHAKE is automatically set to OFF.

SET HANDSHAKE

Syntax: SET HANDSHAKE *option*

Specify any half-duplex handshaking to be done on the currently selected port. The options are BELL, CR, LF, NONE, XOFF, or XON. The specified handshaking will be done during file transfer only. If HANDSHAKE is set to anything other than NONE. FLOW-CONTROL is automatically set to OFF.

SET HEATH19

Syntax: SET HEATH19 ON *or* OFF

Specify whether Kermit-MS should use its built-in software facility for emulating a Heath/ Zenith-19 (H19) terminal.

ON During CONNECT, incoming characters are to be examined for H19 terminal screen control commands (escape sequences), and if encountered, the commands are to be emulated on the PC screen. The H19 codes are a superset of the popular DEC VT52 codes, so if your system does not support the Heath-19, you may tell it that your terminal type is VT52 (or one of the many VT52 compatibles). The Heath-19 codes are listed in section 10.10, below.

OFF All incoming characters will be sent to the screen "bare", through DOS. If you have loaded a device driver into DOS for the CON: device, such as ANSI.SYS, then that driver will be able to interpret the codes itself. Most non-IBM systems have their own screen control code interpreter built into DOS or firmware, or available as a loadable device driver.

See section 10.4 for details about terminal emulation.

SET KEY

Syntax: `SET KEY` *key-specifier*

Specifies that when the designated key is struck during terminal emulation, the associated character string is sent. The key-specifier is one of the keywords F1, F2, ..., or SCAN followed by a scan code. Systems that have a BACKSPACE key also include BACKSPACE as a keyword.

If SCAN is used, it is followed by a *decimal* number to indicate the scan code of the key, which you would ascertain from your system reference manual, or else by using the Kermit-MS SHOW KEY command. SET KEY prompts you on a new line for the definition string. Certain characters, like ESC and CR, may not be entered literally into the string, but can be included by inserting escape codes of the form **\ooo**, a backslash followed by a 2- or 3-digit *octal* number corresponding to the ASCII value of the desired character. If some other key redefinition package, like ProKey, has been loaded, then its redefinitions will take precedence over Kermit's.

The SET KEY command is illustrated in the terminal emulation section, 10.4, below.

SET LOCAL-ECHO

Syntax: `SET LOCAL-ECHO` *option*

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

When you SET LOCAL-ECHO ON, the current HANDSHAKE (if any) is automatically enabled and full-duplex FLOW-CONTROL is automatically turned off. When you SET LOCAL-ECHO OFF, HANDSHAKE is also disabled, and the current mode of FLOW-CONTROL (if any) is enabled. If this behavior is undesired, you may override it by typing explicit SET HANDSHAKE or SET FLOW commands after entering the SET LOCAL-ECHO command.

SET PARITY

Syntax: `SET PARITY` *keyword*

Specify the character parity to be used on the currently selected port. The choices for SET PARITY are NONE (the default), ODD, EVEN, MARK, 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.

You will need to SET PARITY to ODD, EVEN, MARK, or possibly SPACE when communicating with a system, or over a network, or through modems, concentrators, multiplexers, or front ends that require or impose character parity on the communication line. For instance, GTE Telenet requires MARK parity. If you neglect to SET PARITY when the communications equipment requires it, the symptom may be that terminal emulation works partially, and file transfer does not work at all.

If you have set parity to ODD, EVEN, MARK, or SPACE, then Kermit-MS will request that binary files will be transferred using 8th-bit-prefixing. If the other side knows how to do 8th-bit-prefixing (this is an optional feature of the KERMIT protocol, and not all implementations of KERMIT have it), then binary files can be transmitted successfully. If NONE is specified, 8th-bit-prefixing will not be requested. Note that there is no advantage to using parity; it only slows Kermit file transfer down. The SET PARITY command is provided only to allow Kermit to adapt to hardware that insists upon using parity.

SET PORT

Syntax: **SET PORT** *number*

On machines with more than one communications port, select the port to use for file transfer and CONNECT. This command lets you use a different asynchronous adapter, or switch between two or more simultaneous remote sessions. Subsequent SET BAUD, PARITY, HANDSHAKE, FLOW, and LOCAL-ECHO commands will apply to this port only. SET PORT 1 selects COM1, SET PORT 2 selects COM2.

SET REMOTE

Syntax: **SET REMOTE ON** *or* **OFF**

If you wish to run Kermit-MS interactively through the back port, for instance after the operator has done CTTY COM1, you must give the command SET REMOTE ON; this suppresses the file transfer display screen, so that the display won't interfere with the file transfer itself.

SET RECEIVE

Syntax: **SET RECEIVE** *parameter value*

At the beginning of a protocol operation, request the remote Kermit to use the given value specified parameter, or inform Kermit-MS that the remote Kermit will be using it.

PACKET-LENGTH	Ask the remote Kermit to use the specified maximum length for packets that it sends to Kermit-MS. The normal (and maximum) length is 94. Use this command to shorten packets if the communication line is noisy; this will decrease the probability that a particular packet will be corrupted, and will reduce the retransmission overhead when corruption occurs, but it will increase the protocol overhead.
PADCHAR	Ask the remote Kermit to use the given character for interpacket padding. Kermit-MS should never require any padding.
PADDING	Ask the remote Kermit to insert the given number of padding characters before each packet it sends. This should never be necessary.
START-OF-PACKET	The remote Kermit will be marking the beginning of packets with something other than Control-A. This will be necessary only if the hosts or communication equipment involved cannot pass a Control-A through as data.
TIMEOUT	Ask the remote Kermit to time out after the given number of seconds if a packet expected from Kermit-MS has not arrived. Use this command to change the normal timeout interval.

SET SEND

Syntax: **SET SEND** *parameter value*

PACKET-LENGTH	Use the specified maximum length for outbound packets. Normally, Kermit-MS uses whatever length the other Kermit requests.

PADCHAR	Use the specified character for interpacket padding. Some hosts may require some padding characters (normally NUL or DEL) before a packet.
PADDING	How many padding characters to use between packets, normally zero.
QUOTE	Use the indicated printable character for prefixing (quoting) control characters and other prefix characters. The only reason to change this would be for sending a very long file that contains very many "#" characters (the normal control prefix) as data.
START-OF-PACKET	Mark the beginning of outbound packets with some control character other than Control-A. This will be necessary only if the remote host or the communication channel involved cannot accept a Control-A as data. The remote host must have been given the corresponding SET RECEIVE START-OF-PACKET command.
TIMEOUT	Change Kermit-MS's normal timeout interval; this command is effective only if TIMER is set to be ON; it is normally OFF so that the remote KERMIT can control timeouts.

SET TAKE-ECHO

Syntax: `SET TAKE-ECHO ON` *or* `OFF`

Specifies whether screen display should occur during implicit or explicit TAKE operations on `MSKERMIT.INI` or other Kermit-MS command files, and during evaluation of macro definitions. Handy for finding errors in command files.

SET TIMER

Syntax. `SET TIMER ON` *or* `OFF`

Enable or disable the timer that is used during file transfer to break the deadlock that occurs when an expected packet does not arrive. By default, the timer is OFF, because Kermit-MS is usually used in conjunction with a mainframe that is doing its own timeouts. During a file transfer, it is sufficient for one side to do the timing out and the mainframe is usually better equipped to adjust timeout intervals based on system load or other conditions. The timer should be set ON if you are communicating with a system that cannot do timeouts, such as IBM VM/CMS Kermit.

SET WARNING

Syntax: `SET WARNING` *option*

Specify what to do when an incoming file has the same name as an existing file in the default directory of the default device. If ON, Kermit will warn you when an incoming file has the same name as an existing file, and automatically rename the incoming file (as indicated in the warning message) so as not to destroy (overwrite) the pre-existing one. If OFF, the pre-existing file is destroyed, even if the incoming file does not arrive completely.

10.3.6. The SHOW Command

Syntax: SHOW *option*

Currently, most parameters that may be altered with SET commands are displayed by the STATUS command. The SHOW command is used for displaying macro definitions and key redefinitions.

The SHOW MACROS command displays the definitions of all currently defined macros.

The SHOW KEY command allows you to determine the scan code produced by pressing a given key, so that you can construct a SET KEY command to redefine the key. If the key already has a redefinition in effect, that too will be displayed. In this example, a DEC Rainbow user determines the scan code for the accent grave key, and then redefines that key to send ESC:

```
Kermit-MS>show key
Press a key: `
  Scan Code: 96
  Definition:
Kermit-MS>set key scan 96
Definition string: \33
Kermit-MS>show key
Press a key: `
  Scan Code: 96
  Definition: \33
Kermit-MS>
```

The SHOW KEY command only works on certain systems.

10.3.7. Command Macros

Kermit-MS provides a facility for combining commands into "macros." Command macro definitions may be included in your MSKERMIT.INI file, TAKEn explicitly from a specified file, or typed interactively, and may be invoked with the DO command.

The DEFINE Command

Kermit-MS command macros are constructed with the DEFINE command. The syntax is

DEFINE *macro-name* [*command* [, *command* [, ...]]]

Any Kermit-MS commands may be included. Example:

```
define telenet set parity mark, set baud 1200, connect
```

The DO Command

A Kermit-MS command macro is invoked using the DO command. For instance, Kermit-MS comes with a predefined macro to allow convenient setup for IBM communications; to invoke it, you would type

```
do ibm
```

The IBM macro is defined as "parity mark, handshake xon, local-echo on, timer on". You can delete or replace this definition by adding a new (perhaps null) definition, such as

```
define ibm parity even, handshake cr, local-echo on, timer on
```

or

```
define ibm
```

10.4. Terminal Emulation

When you issue the CONNECT command, your PC acts as a terminal connected to a remote computer through the currently selected port. The characters you type are sent out the port, and characters that arrive at the port are displayed on your screen. If you have not previously issued a SET PORT command, COM1 is used. If you have SET LOCAL-ECHO ON for the selected port, then Kermit-MS will display characters on the screen as you type them. If LOCAL-ECHO is OFF, then XON/XOFF flow control will be done unless you have SET FLOW-CONTROL OFF. If you have SET PARITY to anything other than NONE, Kermit-MS will add the appropriate parity to each outbound character, and strip any parity from incoming ones. While CONNECTed, you can also communicate with an autodialer or "smart modem" to control the communications line, hang it up, and the like; for instance, typing +++ to a Hayes-like modem will allow you to follow that by dialing or hangup commands.

If Heath-19 emulation is being done, incoming characters will be monitored for H19/VT52 escape sequences. These will be interpreted according to the table in section 10.10. In addition, keys on the numeric keypad will send H19/VT52 sequences unless you disable this feature in some way, for instance by pressing Num Lock on the IBM PC keyboard, or with key redefinitions.

> *Caution:* On some systems, such as the IBM PC and XT, Kermit-MS accesses the screen memory memory directly to perform certain H19 emulation functions such as character insert/delete and screen scroll. Without direct screen memory access, these functions would be painfully slow. Although Kermit-MS has been tested successfully on a variety of monochrome and color adapters and monitors, there may be combinations for which this method could cause video problems, such as snow. Should this occur, you can alleviate the problem by setting HEATH19 emulation OFF. In that case, however, you remove not only the problems, but also the desirable features of emulation. But Kermit-MS does permit you to load an external console device driver, such as IBM's **ANSI.SYS**, to provide any desired screen control.

Here are the terminal emulation options for the systems presently supported by Kermit-MS:

System	EscChar	Cabilities	Terminal Service
IBM PC, XT	^]	R M P K	Heath19 emulation
DEC Rainbow	^]	R P K	VT102 firmware
HP-150	^]	R	HP-2623 firmware
Wang PC	^A		Wang firmware
Generic DOS	^]		Depends on system

Under Capabilities, R means rollback, M means mode line, P means printer control, and K means key redefinition.

IBM PC/XT Kermit can disable Heath-19 emulation and use an external console device driver like **ANSI.SYS** instead.

When you first issue the CONNECT command, a message (on some systems, an inverse video "mode line") will display the most important facts about the connection you've just established, so that you can quickly diagnose any problems. The items displayed in the mode line include the escape character, port number, the baud rate, the parity, the echo, and how to get help, for instance:

```
+--------------------------------------------------------------------+
|EscChar:^],Port:1,Baud:9600,Parity:None,Echo:Remote,Type ^]? for Help|
+--------------------------------------------------------------------+
```

The escape character is used to regain the attention of Kermit-MS. When you type the escape character, Kermit-MS waits for you to follow it with a single character command. For instance, the single-character-command "?" produces a list of available single character commands, such as this:

?	Help -- prints the available single-character commands.
C	Close the connection and return to Kermit-MS prompt level.
S	Show the status of the connection.
B	Send a BREAK signal to the port.
0	(the digit zero) Send a NUL (ASCII 0) to the port.
Q	Temporarily quit logging the remote session.
R	Resume logging the remote session.
M	Toggle the mode line, i.e. turn it off if it is on & vice versa.
^]	(or whatever you have set the escape character to be) Typing the escape character twice sends one copy of it to the connected host.

Typing any other character (except the space bar, which is the "null command") after the escape character will cause Kermit-MS to beep, but will do no harm. The escape character can be changed to something other than Control-Rightbracket by using the SET ESCAPE command.

Kermit-MS includes several advanced features for use during terminal emulation, including screen scroll, printer control, and key redefinitions.

Screen Scroll

Kermit-MS provides several pages of screen memory, which may be scrolled up and down using keys as follows:

Function	IBM PC/XT	Rainbow	HP-150
Screen Down	PgDn	PrevScreen	Prev
Line Down	Ctrl-PgDn	Ctrl-PrevScreen	Shift-UpArrow
Screen Up	PgUp	NextScreen	Next
Line Up	Ctrl-PgUp	Ctrl-NextScreen	Shift-DownArrow
Top of Memory	Home		
Bottom of Memory	End		

There is presently no way to assign these functions to other keys.

Printer Control

A locally attached printer may be controlled in the normal manner, on most systems. Pushing the "Print Screen" key (shifted on some systems) will cause the current contents of the screen to be printed or spooled; holding down CTRL while depressing Print Screen will start or stop the spooling of incoming characters to the printer. ^P or ^N are sent to the host during terminal emulation, and do not toggle printing, as they do when you're talking directly to DOS.

CTRL-Print-Screen can be simulated with the Kermit-MS LOG PRN and CLOSE commands.

Key Redefinitions

Key redefinitions are useful for defining "keystroke macros" of login sequences, frequently issued commands, and so forth, and for setting up the terminal for use with host resident software designed to work with terminals that send predefined sequences from their function keys. For instance, here's a key redefinition file for arranging the DEC Rainbow keyboard into the normal ASCII keyboard layout:

```
; Make shift-comma send a left angle bracket
set key scan 556
<
; Shift-period sends a right angle bracket
set key scan 558
>
; Accent grave is where ESC is supposed to be
set key scan 96
\33
; Put accent grave on the ESC function key
set key f11
'
```

The SET KEY facility may be used provide the PC with a "meta" key for use with editors like EMACS or TVEDIT that can use "meta characters" as commands. A meta key is a shift key whose effect is to turn on the 8th (parity) bit of the character. For instance, on the IBM PC the scan codes produced by holding down ALT together with other keys can be determined using SHOW KEY, and then 8-bit ASCII equivalents with the 8th bit turned on can be defined using SET KEY; if the scan code produced by typing ALT-a, i.e. the letter "a" (ASCII 141, octal) with the ALT key held down, is 2078 (decimal), you would set the META equivalent to 141+200=341 (octal), or "\341" in octal SET KEY notation:

```
Kermit-MS>sho key
Press a key: ALT-a
  Scan Code: 2078
  Definition:
Kermit-MS>set key scan 2078
Definition String: \341
```

Whenever you type ALT-a with this definition in effect, Kermit-MS will transmit octal 341, rather than 141.

10.5. Installation of Kermit-MS

by Bill Catchings, Columbia University

If you already have Kermit on your PC, you can use it to obtain new versions of Kermit-MS when they appear on the central system at your site. If you do not have Kermit or any other reliable file capture facility on your PC, and there is no one from whom you can borrow a floppy disk to copy Kermit, then you should read the following instructions for initially "bootstrapping" Kermit-MS from a mainframe where it is stored onto your microcomputer.

There are at least three methods of initially getting Kermit-MS onto your PC:

1. Try again to find a copy on diskette.
2. Use another file capture facility to get it.
3. Type in and run a bootstrapping program.

10.5.1. Try Again To Find A Kermit Disk

Before explaining how to bootstrap Kermit onto your PC, a disclaimer must be made. Although a fair amount of thought and time has gone into these procedures, they are far from error free. If they were foolproof, there would be no need for a protocol such as Kermit. There are many places where things can go wrong, from something as simple as a typing mistake to something as unavoidable and probably inevitable as a communications line failure. By far the easiest and best way to install Kermit is from a floppy disk. Before you embark on any of the following procedures it is a good idea to check once again for a diskette to copy, even it it contains an old version of Kermit. The time you spend searching is likely to be far less frustrating than the time you spend trying to bootstrap Kermit by the methods described below.

10.5.2. Bootstrapping From the Communication Line

If you can't find a diskette with Kermit on it, there are two other methods available for bootstrapping MS-DOS Kermit onto your PC. The first method is to use a file capture method or other file transfer protocol to transfer the file to your PC. Some systems come supplied with facilities like this, and various public domain or commercial packages are available. The second method requires you to type in your own downloading program.

In either case, you must transmit the file from the system where it resides over a communication line and into your PC. Since version 2 of MS-DOS Kermit is much larger than version 1, it comes with a new bootstrapping procedure in which the executable program is encoded much more compactly than in the earlier "fix" files. The new encoding packs 3 .EXE file bytes into 4 printable characters in the MS*xxx*.BOO file, and also compresses adjacent zero bytes (of which there are many). The .BOO file contains only printable characters, to ensure that downloading can take place regardless of parity or other pecularariaries of the communication channel.

10.5.2.1. Use An Existing File Capture Facility

In the rest of this discussion of bootstrapping, the host-resident boot .BOO file will be referred to as MSKERMIT.BOO. In fact, the actual name will depend on which MS-DOS system you are using -- MSIBMPC.BOO for the IBM PC or XT, MSRB100.BOO for the Rainbow 100, etc

Use your file capture facility, whatever it may be, to get the file MSKERMIT.BOO onto your PC's disk, but first make sure you have enough room for it. Once the file is on your disk, you must run the BASIC program MSPCTRAN.BAS to decode the file back into KERMIT.EXE. This program can be downloaded by the same method you used with MSKERMIT.BOO. The program looks on your current disk and directory for the file MSKERMIT.BOO and outputs KERMIT.EXE to the same place. KERMIT.EXE is about 80K bytes, so make sure there is space for it on your disk or else you will have to start the program over. Since the program will take about twenty minutes to completely translate the file you will want to avoid running it more than once.

10.5.2.2. Type In Your Own Bootstrap

If you can't find some method for downloading the **.BOO** file and the BASIC program, the second way of bootstrapping Kermit is to use the programs **MSPCBOOT.BAS** and **MSBOOT.FOR** to download via your PC's asynchronous port from your host and translate it directly, "on the fly." You run the program **MSBOOT.FOR** on your host and then run the program **MSPCBOOT.BAS** in BASIC on your PC. The FORTRAN program sends the encoded **.EXE** file to the BASIC program, which decodes it and stores it in executable form on your current directory as **KERMIT.EXE**. A very rudimentary form of error checking is done to allow obviously corrupted records to be retransmitted. Follow this procedure:

1. First, you must establish a connection from your PC to the host system. A high speed connection is preferable; a "clean" line is preferable to a noisy one. In fact, a clean line is essential for this procedure. You must be able to log in to the host system over this connection. If your PC already has a terminal emulation facility, use that. If not, you might need to put your PC next to a real terminal and use that for logging in, then switch the connector to the PC at the crucial moment. If you are using a terminal, make sure the terminal and PC have their communication ports set to the same speed.

2. Ensure that the files **MSBOOT.FOR** and **MSKERMIT.BOO** are present on the host system. **MSBOOT.FOR** is listed below, in case you need to type it in.

3. Get back to your PC and type in **MSPCBOOT.BAS** on your PC; a listing appears below. There is no need to type in the comments (anything following an apostrophe); they are only there to clarify what the program is doing. Check very carefully for errors. You should check line 80 in the program to see that it reflects the way your system is actually set up. If necessary, substitute the correct baud rate for the supplied rate of 9600, and if you are not using **COM1:** make that change as well. If you are downloading from an IBM or other half-duplex mainframe, leave line 1000 as it is; otherwise, replace it by a RETURN statement. If you type it in directly to BASIC make sure you save the program before you run it, so you won't have to type it in again in case of error.

4. Get back to your host system and compile **MSBOOT.FOR**, if it needs compiling. Define logical unit numbers 5 and 6 to be the controlling terminal, and logical unit 7 to be the file **MSKERMIT.BOO**. On VAX/VMS systems, for example, use these commands:

   ```
   $assign sys$input for005
   $assign sys$output for006
   $assign mskermit.boo for007
   ```

   On a DECSYSTEM-20, do:

   ```
   @define 5: tty:
   @define 6: tty:
   @define 7: mskermit.boo
   ```

   On a DECsystem-10, do something like this:

   ```
   .assign tty: 5:
   .assign tty: 6:
   .assign dsk: 7:
   .rename for007.dat=mskerm.boo
   ```

   On an IBM system under VM/CMS, do this:

   ```
   .filedef 5 term ( lrecl 80 recfm v
   .filedef 6 term ( lrecl 80 recfm v
   .filedef 7 disk mskermit boo ( lrecl 77 recfm f perm
   ```

5. Set your host system up for downloading:

   - Ensure that your terminal does not automatically pause at the end of a screenful of output. For instance, on a DEC-20 you would issue the command "terminal no pause end-of-page".
   - Do whatever you can to disable messages from appearing at your terminal while these programs are running. This would include messages from other users, mail notification, alarms or alerts, system messages, and so forth. Such messages will interfere with the procedure, and probably render the result useless.
   - You should put your host terminal in "local echo" or "half duplex" mode, if possible.

6. Start the **MSBOOT** program on your host system.

7. Get back to the PC. If you have been using a terminal, switch the connector to the PC.

8. Now run the BASIC program, **MSPCBOOT.BAS**. This procedure will take at least twenty minutes and possibly longer depending on line speed. Watch your modem and/or disk lights for reassurance that something is happening.

By using one of these installation methods, you should now have a working version of Kermit. If you experience any problems or quirky behavior with the program, it's possible that some part of it was corrupted during the downloading procedure. Perhaps enough usable code remains to allow you to transfer **MSKERMIT.EXE** from the host. If not, you will have to repeat the downloading procedure.

Once you have Kermit-MS on your disk, you should make the disk available to other users for copying, so that they can be spared the tedium of this bootstrap procedure.

Here is a listing of **MSPCBOOT.BAS**. The "outdented" PRINT statements with line numbers ending in 5 may be included if you want incoming records to be displayed on the screen. You don't need to include the comments.

```
1     'Run this program on the PC in conjunction with a Fortran program
2     '(MSBOOT.FOR) on the mainframe to download Kermit to the PC.  This
3     'program will run for about thirty minutes, depending on line speed.
4     ' Bill Catchings, June 1984
5     ' Columbia University Center for Computing Activities

10    t$ = time$                        ' Save the time.
20    defint a-z                        ' All integer to gain some speed.
30    n$ = chr$(0)
40    z = asc("0")
50    t = asc("~")-z
60    def fnuchr%(a$)=asc(a$)-z
70    open "com1:9600,s,7,1,cs,ds,cd" as #1

100   print#1,"0 ,"                     ' Char constants "0", " " and ","
110   input#1,f$
120   if len(f$) < 5 then goto 110      ' In case the host echos the ACK.
130   input#1,n
135 print f$+" "+str$(n)
140   if n > 20 then goto 900
150   open f$ for output as #2
160   print "Outputting to "+f$
170   goto 300                          ' Correct version of the file.
```

```
200  gosub 1000                        ' Do turnaround char processing
210  print#1,"NO"                      ' Tell host data was incorrect.
220  goto 320

300  gosub 1000                        ' Do turnaround char processing
310  print#1,"OK"                      ' Say the line was all right.
320  input#1,x$
330  if len(x$) < 5 then goto 320      ' In case the host echos ACK/NAK
340  input#1,n
345 print x$+" "+str$(n)
350  if len(x$) <> n then goto 200     ' Length doesn't match, NAK it.
360  if x$ = "&&&&&&&&&&" then goto 800 ' End of file?
370  y$ = ""                           ' Set output string to null.
380  goto 500

400  print#2,y$;                       ' Print the output string.
410  goto 300                          ' Go get another line.

500  if len(x$) = 0 goto 400           ' Done with input string?
510  a = fnuchr%(x$)
520  if a = t then goto 700            ' Null repeat character?
530  q$=mid$(x$,2,3)                   ' Get the quadruplet to decode.
540  x$=mid$(x$,5)
550  b = fnuchr%(q$)
560  q$ = mid$(q$,2)
570  c = fnuchr%(q$)
580  q$ = mid$(q$,2)
590  d = fnuchr%(q$)

600  y$ = y$ + chr$(((a * 4) + (b \ 16)) and 255) ' Decode the quad.
610  y$ = y$ + chr$(((b * 16) + (c \ 4)) and 255)
620  y$ = y$ + chr$(((c * 64) + d) and 255)
630  goto 500                          ' Get another quad.

700  x$ = mid$(x$,2)                   ' Expand nulls.
710  r = fnuchr%(x$)                   ' Get the number of nulls.
715 print " Null: ",r
720  x$ = mid$(x$,2)
730   for i=1 to r                     ' Loop, adding nulls to string.
740   y$ = y$ + n$
750   next
760  print#2,y$;                       ' Print the nulls.
770  y$ = ""                           ' Clear the output buffer.
780  gotc 500

800  print "Processing complete, elapsed time: "+t$+" to "+time$
810  print "Output in "+f$
820  close #1,#2
830  goto 9999

900  print "?The format of the BOO file is incorrect"
910  goto 820

1000 x$ = input$(1,#1)            ' Make this line RETURN for full-duplex
1010 if x$ <> chr$(17) then goto 1000   ' Loop for a turn around char.
1020 return

9999 end
```

Here is a listing of MSBOOT.FOR, in case you can't find it on your host system:

```
C       This Fortran program should be run on the mainframe in conjunction
C       with a Basic program (MSPCBOOT.BAS) on the PC to transfer
C       MSKERMIT.BOO to the PC and translate it into KERMIT.EXE.  This
C       program uses a very rudimentary technique to try to insure that
C       the characters it sends arrive correctly.  It just sends a count
C       of the number of characters sent after each line.  In this way any
C       errors of character loss or insertion will be caught.  If a
C       character is just corrupted it will not be caught.  Hopefully if
C       this happens it will be in a non-critical part of the KERMIT.EXE
C       file.  The reason a simple checksum was not used was so that this
C       program could run on machines using either EBCIDIC or ASCII
C       characters.  This program should take about thirty minutes to run.

C       This program assumes that 5 and 6 are directed to the terminal and
C       7 is directed to the file MSKERMIT.BOO.

        INTEGER LINE(77), ACK(4), CHECK, OK, SPACE, COMMA

        WRITE(6,100)
100     FORMAT(' Ready to transfer data, now run MSPCBOOT.BAS on the PC.')

C       Get characters for constants (character constants are rough in
C       some FORTRANs!)
        READ (5,200) OK, SPACE, COMMA, ACK
200     FORMAT(4A1)
        GO TO 30

C       Get terminal handshake.
10      READ (5,200)ACK

C       Did the other side like it?  (Did they send OK?)
        IF (ACK(1) .NE. OK) GO TO 50

C       Yes, get new line from file.
20      READ (7,300,END=99)LINE
300     FORMAT(77A1)

C       Count the characters as some rudimentary check for noise.
        I = 1
30      IF (LINE(I) .EQ. SPACE) GO TO 40
        I = I + 1
        GO TO 30

C       Put in a comma followed by the count.
40      LINE(I) = COMMA

C       Write to TTY.
50      WRITE (6,400)LINE,I-1
400     FORMAT(' ',77A1,I2)
        GOTO 10

C       Send good-bye message.
99      WRITE (6,500)
500     FORMAT(' ',10('&'),',10')

        STOP
        END
```

10.6. Compatibility with Older Versions of MS-DOS Kermit

MS-DOS Kermit supports many different systems. Like CP/M-80 KERMIT, this support was added to the program piecemeal, at many sites, using conditional assembly. However, before allowing the program to grow into a complicated monolith like CP/M-80 KERMIT, we have broken the program up into separate modules, with system dependencies isolated into a single module consisting of compact collections of low-level primitives for console and port i/o.

The last monolithic (single source file) release of MS-DOS Kermit was 1.20. To this and earlier versions was added support for systems like the Seequa Chameleon, the Tandy 2000, the Victor 9000, the Heath/Zenith 100, and others. Eventually, support for these systems may be integrated with the new modular version. Meanwhile, implementations based on these old versions will have at least the following incompatibilies from the version described here:

- RECEIVE *filespec* is used instead of GET *filespec*. There is no GET command in older versions, and no way to specify a new name for an incoming file.
- No LOCAL or REMOTE commands.
- No 8th-bit prefixing, repeat counts, CRCs or 2-character checksums.
- No TAKE or initialization files.
- No command macros or command line arguments.
- No terminal session logging.

and others, depending on the specific version.

10.7. What's Missing

Kermit-MS has plenty of room for improvement. Features that need to be improved or added include:

- A built-in facility for sending files "raw" to the remote system, obeying current settings for parity, flow control, handshake, and so forth. This might include a script interpretation facility to allow remote sessions to be conducted automatically. For the present, this can be accomplished with a user-supplied program invoked with the Kermit-MS RUN command.
- Additional functionality when running in server mode -- directory listings, file deletion, execution of DOS commands, etc.
- More commands when talking to remote servers -- REMOTE RENAME, COPY, STATUS, WHO, etc.
- Filename conversion options (normal form, handling of fully qualified filespecs, etc.).
- Transaction file logging.
- Improved command parsing; for instance, accept default values for omitted trailing fields.
- A better built-in help facility.
- Support for Kermit file attribute packets.
- The Kermit-MS program is quite large. Much of the size is due to the deliberate decision to provide support for versions of MS-DOS prior to 2.0. At some point, this support should be removed. This will not only reduce the size of the program considerably, but also provide much more flexibility.

10.8. Program Organization

Kermit-MS version 2 is composed of separate assembler source files, assembled separately, and linked together. The modules are:

System/Device Independent:

MSKERM.ASM	Main program
MSSEND.ASM	File sender
MSRECV.ASM	File receiver
MSSERV.ASM	Server operation
MSFILE.ASM	File i/o
MSCMD.ASM	Command parser
MSTERM.ASM	CONNECT command
MSCOMM.ASM	Communications port buffering & flow control
MSSET.ASM	SET, SHOW, and STATUS commands
MSDEFS.H	Data structure definitions and equates

System/Device Dependent:

MSX*xxx*.ASM	System-dependent code for system *xxx*
MSY*xxx*.ASM	System-dependent screen and keyboard code
MSZ*xxx*.ASM	Modem control (modem-dependent)

The modular organization allows easier modification of the program, quicker transfer of modified portions from system-to-system. The modules are designed to be well-defined and self-contained, such that they can be easily replaced. For instance, someone who prefers windows and mice to typing commands could replace the command parsing module without having to worry about the effect on the other modules.

To assemble any of the kermit modules, file **MSDEFS.H** must be on the default disk.

All the Kermit implementations require the modules MSCMD, MSCOMM, MSFILE, MSKERM, MSRECV, MSSEND, MSSERV, MSSET, MSTERM.

The IBM PC version requires MSXIBM and MSYIBM as well.

The Rainbow version requires MSXRB and MSXDMB, MSXDMB must be the first object file given to the linker for Kermit to link properly for the Rainbow.

The HP150 version requires MSXHP150, the Wang version requires MSXWNG, and the generic version requires MSXGEN.

Once all the required object modules exist, they may be linked together to produce Kermit. For example, on the Rainbow:

```
A>link

   Microsoft Object Linker V2.00
(C) Copyright 1982 by Microsoft Inc.

Object Modules [.OBJ]: msxdmb mskerm msxrb mscomm msset mssend +
msrecv msserv msfile msterm mscmd
Run File [MSXDMB.EXE]: kermit
List File [NUL.MAP]: kermit

A>
```

10.9. Adding Support For New Systems

You can bring Kermit-MS to systems that are not explicitly supported in one of two ways -- attempt to run the "generic" MS-DOS Kermit on it, or add explicit code to support your system.

10.9.1. Generic MS-DOS Kermit

To get started with Kermit on a new system, try running "generic" MS-DOS Kermit; in many cases, it will run as is. The generic version accomplishes all its port and console i/o through DOS calls, and during terminal connection does not attempt to emulate any particular kind of terminal. In some cases, the generic version may still require some fiddling to run on a new system; for instance, different systems refer to their communication ports in different ways -- COM1, AUX, etc. It attempts to do this automatically by trying various DOS file handles for the communication port, and asking you to supply one if it does not succeed.

Generic MS-DOS Kermit will probably run no faster than 1200 baud, and it only works with DOS 2.0 or later.

10.9.2. Adding System-Dependent Code

The following is a guide to the system dependent module of Kermit-MS.

Specification for Kermit System-Dependent Modules

by Jeff Damens, Columbia University

All the system-independent global data structures used in Kermit-MS are defined in the file **MSDEFS.H**.

The routine **MSX***xxx***.ASM** contains system-dependent support for system *xxx*, except for terminal emulation, which is in **MSX***xxx***.ASM**, described below.

The routines in the MSX module may change any registers but the stack pointer and segment registers unless otherwise noted. A routine that returns via a RET instruction is said to return normally; a routine that skip returns is one that returns to three bytes past the normal return address.

Global variables that must be defined in the system-dependent module:

XOFSNT	byte. This should be set to a non-zero value if we are doing flow control and have sent an XOFF character to the remote host, zero otherwise.
MACHNAM	byte. A $-terminated string identifying the machine this version of Kermit is for; it is printed when Kermit starts up.
SETKTAB	byte. A keyword table associating terminal key names to 16-bit scan code values, used in the set key command. If the kermit version can accept arbitrary decimal values as scan codes, the word "SCAN" should appear in the table with a scan value of -1. If key redefinition is not implemented, the first byte of the table should be a zero.
SETKHLP	byte. A $-terminated string to be printed when ? is typed in the SET KEY command. This is usually simply a list of the key names in

SETKTAB. SETKHLP must be defined even if key redefinition is not implemented, to satisfy the linker; if key redefinition is not implemented, SETKHLP will never be displayed.

COUNT word. The number of characters in the serial input buffer, if known. This is how Kermit knows to send an XON if the serial handler has sent an XOFF. If the number of characters in the buffer isn't known, COUNT should be 0.

These are the required entry points for the system dependent dependent module **MSX*XXX*.ASM**.

SERINI

Parameters	None.
Returns	Normally, no return value.
Description	Perform any initialization that must be done before the serial port can be used, including setting baud rate, interrupt vectors, etc. Parity and baud rate should be set according to the values in the PORTINFO structure. The external variable PORTVAL points to the PORTINFO structure for the current port. Calling SERINI more than once without an intervening call to SERRST should have no effect.

SERRST

Parameters	None.
Returns	Normally, no return value.
Description	Undoes any initialization done by SERINI, including resetting the serial port, restoring any interrupt vectors changed by SERINI, etc. Calling this more than once without an intervening call to SERINI should be harmless.

CLRBUF

Parameters	None
Returns	Normally, no return value.
Description	Remove and discard from the serial port's input buffer any characters sent by the remote host that have not yet been read by Kermit, and set COUNT to 0. This is used before a file transfer to flush NAK's that accumulate in the buffer when the remote host is in server mode.

OUTCHR	
Parameters	A character in AH.
Returns	Skip returns if the character has been transmitted; returns normally if the character can not be transmitted because of a hardware error.
Description	Sends the character in AH out the currently selected serial port. OUTCHR can assume that SERINI will have been called previously. OUTCHR should call the external routine DOPAR to set the parity of the character if the communications hardware doesn't automatically set parity. Flow control should be honored; the external variable PORTVAL contains a pointer to a PORTINFO structure (as defined in MSDEFS.H) containing the current flow control definitions.

COMS	
Parameters	None.
Returns	Normally if a parse error is encountered, skip returns otherwise.
Description	Called by the SET PORT command. On a machine with multiple serial ports, COMS should parse for the name or number of a serial port and make that the port used by succeeding calls to SERINI, PRTCHR, OUTCHR, and SERRST. It should set the external variable PORTVAL to point to one of the external port structures PORT1 or PORT2, and set COMFLG in the FLAGS structure to 1 for port one, 0 for port 2. For implementations that use only one serial port, COMS should print a message to that effect and skip return.

VTS	
Parameters	None.
Returns	Normally if a parse error is encountered, skip returns otherwise.
Description	Parses for an ON or OFF, sets HEATH-19 emulation while in terminal emulation appropriately. The VTFLG field of the FLAGS structure should be set non-zero if HEATH-29 emulation is on, zero otherwise. If HEATH-19 emulation is not done, VTS should print a message and skip return.

DODEL	
Parameters	None.
Returns	Normally, no return value.
Description	Erases the character immediately to the left of the cursor from the screen, then backs up the cursor.

CTLU	
Parameters	None.
Returns	Normally, no return value.
Description	Move the cursor to the left margin, then clear the line.

CMBLNK

Parameters	None.
Returns	Normally, no return value.
Description	Clears the screen and homes the cursor.

LOCATE

Parameters	None.
Returns	Normally, no return value.
Description	Homes the cursor.

LCLINI

Parameters	None.
Returns	Normally, no return value.
Description	Performs any system-dependent initialization required by this implementation.

PRTCHR

Parameters	None.
Returns	Normally, with the next character from the currently selected serial port in AL. Skip returns if no character is available.
Description	Reads the next character from the current serial port. PRTCHR can assume SERINI has been called previously, and should handle flow control correctly.

DOBAUD

Parameters	None.
Returns	Normally, no return value.
Description	Sets the baud rate for the current port. The baud rate should be obtained from the BAUD field of the PORTINFO structure, pointed to by the external variable PORTVAL.

CLEARL

Parameters	None.
Returns	Normally, no return value.
Description	Clears from the cursor to the end of the current line.

DODISK

Parameters	None.
Returns	Normally, no return value.
Description	Sets the external variable DRIVES to the number of disk drives attached to the machine.

GETBAUD

Parameters	None.
Returns	Normally, no return value.
Description	Store current baud rate of the currently selected port in the BAUD field of the current PORTINFO structure, which is pointed to by PORTVAL. If the baud rate is to default to a particular value, this routine can store that value into the BAUD field instead.

BEEP

Parameters	None.
Returns	Normally, no return value.
Description	Rings the terminal bell.

PUTHLP

Parameters	A pointer to a string in AX.
Returns	Normally, no return value.
Description	Writes the null-terminated string given in AX to the terminal This is used to display help and status messages The IBM and Rainbow versions write the string in a reverse video box.

PUTMOD

Parameters	A pointer to a string in AX.
Returns	Normally, no return value.
Description	Writes the null-terminated string given in AX to the last line of the screen, in inverse video if possible.

CLRMOD

Parameters	None.
Returns	Normally, no return value.
Description	Clears the line written by PUTMOD.

POSCUR

Parameters	Row in DH, column in DL.
Returns	Normally, no return value.
Description	Positions the cursor to the row and column given in DX. Rows and columns both originate at 0 (not 1!).

SENDBR

Parameters	None.
Returns	Normally, no return value.
Description	Send a break to the current serial port.

SHOWKEY

Parameters	Pointer to a terminal argument block in AX (see TERM below).
Returns	Normally, with a string pointer in AX and the length of the string in CX.
Description	Called by the SHOW KEY command. Reads a key from the terminal and returns a string containing implementation-dependent information about the key. In the usual case, the string contains the key's (machine-dependent) scan code, and the key's definition (if any) from the terminal argument block. The length of the returned string should be returned in CX. The string may contain any characters; unprintable characters will be quoted when the string is printed. If the implementation does not support key redefinition, SHOWKEY may return a static string saying so.

TERM

Parameters	Pointer to terminal argument block in AX.
Returns	Normally, no return value.
Description	Do terminal emulation, based on argument block described below...

The terminal emulator is supplied in the file MSY*xxx*.ASM. The terminal argument block passed to the terminal emulator has the following fields:

FLGS — Byte containing flags. Flags are:

SCRSAM (80H)	If on, the terminal emulator shouldn't re-display the screen when entered.
CAPT (40H)	Capture output. If on, the routine passed in field CAPTR is called with each character sent to the screen.
EMHEATH (20H)	Emulate a Heath-19 terminal if on.

HAVTT (10H)	A key redefinition table is present.
TRNCTL (08H)	Print control character X as ↑X (useful for debugging).
MODOFF (04H)	Do not display emulator mode line if on.
LCLECHO (01H)	Echo keyboard characters on the screen in addition to sending them to the port.

PRT	Port to use for terminal emulation, used only in mode line. This is just a copy of COMFLG in FLAGS.
COLS	Number of columns on screen.
ROWS	Number of rows on screen.
CAPTR	Routine to call to with each character sent to the screen if CAPT flag is on. Characters are passed in AL.
BELLD	Bell divisor (used only on IBM).
KLEN	Number of keys in key redefinition table, if HAVTT flag is on.
KTAB	Address of key redefinition table. The key redefinition table is a table of KLEN 16-bit scan codes. Each (machine dependent) scan code represents a key that is redefined.
KRPL	Address of key replacement table. The key replacement table parallels the key redefinition table given in KTAB. Entries in the replacement table are 16-bit pointers to redefinitions. Each redefinition has a one-byte length, followed by the definition.
ESCC	Escape character (single byte). When this character is typed to the emulator, it should return.
BAUDB	byte. Bits describing the baud rate so it can be printed on the mode line. This is a copy of the BAUD field in the PORTINFO structure. Currently used only on the IBM. See `MSDEFS.H` for possible values.
PARITY	byte. Current parity to print on the mode line. This is a copy of PARFLG in the PORTINFO structure. Currently used only on the IBM. See `MSDEFS.H` for possible values.

10.10. Heath/Zenith-19 Control Codes

The Heath/Zenith-19 terminal is equivalent to the DEC VT52 with extensions for line and character insertion and deletion. Items marked with an asterisk are not currently supported by Kermit-MS H19 emulation.

Cursor Functions

Sequence	Mnemonic	Definition
ESC H	HCUH	Cursor Home
ESC C	HCUF	Cursor Forward
ESC D	HCUB	Cursor Backward
ESC B	HCUD	Cursor Down
ESC A	HCUU	Cursor Up

ESC I	HRI	Reverse Index
*ESC n	HCPR	Cursor Position Report
*ESC j	HSCP	Save Cursor Position
*ESC k	HRCP	Set Cursor to Previously Saved Position
ESC Y	HDCA	Direct Cursor Addressing, 1-based: 31+line# 31+col# (same as VT52)

Erasing and Editing

Sequence	Mnemonic	Definition
ESC E	HCD	Clear Display (Shift Erase)
ESC b	HBD	Erase Beginning of Display
ESC J	HEOP	Erase to End of Page (Erase Key)
ESC l	HEL	Erase Entire Line
ESC o	HEBL	Erase Beginning of Line
ESC K	HEOL	Erase to End of Line
ESC L	HIL	Insert Line
ESC M	HDL	Delete Line
ESC N	HDCH	Delete Character
ESC @	HEIM	Enter Insert Character Mode
ESC O	HERM	Exit Insert Character Mode

Configuration

Sequence	Mnemonic	Definition
*ESC z	HRAM	Reset to Power-Up Configuration
*ESC r Bn	HMBR	Modify Baud Rate: Bn= A=110, B=150, C=300, D=600, E=1200, F=1800, G=2000, H=2400, I=3600, J=4800, K=7200, L=9600, M=19200
*ESC x Ps	HSM	Set Mode(s): Ps= 1 = Enable 25th line 2 = No key click 3 = Hold screen mode 4 = Block cursor 5 = Cursor off 6 = Keypad shifted 7 = Alternate keypad mode 8 = Auto line feed on CR 9 = Auto CR on line feed
*ESC y Ps	HRM	Reset mode(s): Ps= 1 = Disable 25th line 2 = Enable key click 3 = Exit hold screen mode

4 = Underscore cursor
5 = Cursor on
6 = Keypad unshifted
7 = Exit alternate keypad mode
8 = No auto line feed
9 = No auto CR

*ESC <	HEAM	Enter ANSI Mode

Modes of Operation

Sequence	Mnemonic	Definition
*ESC [	HEHS	Enter Hold Screen Mode
*ESC \	HXHS	Exit Hold Screen Mode
ESC p	HERV	Enter Reverse Video Mode
ESC q	HXRV	Exit Reverse Video Mode
*ESC F	HEGM	Enter Graphics Mode
*ESC G	HXGM	Exit Graphics Mode
*ESC t	HEKS	Enter Keypad Shifted Mode
*ESC u	HXKS	Exit Keypad Shifted Mode
*ESC =	HAKM	Enter Alternare Keypad Mode
*ESC >	HXAM	Exit Alternate Keypad Mode

Additional Operations

Sequence	Mnemonic	Definition
*ESC }	HDK	Keyboard Disable
*ESC {	HEK	Keyboard Enable
ESC v	HEWA	Wrap Around at End of Line
ESC w	HXWA	Discard at End of Line
ESC Z	HID	Identify as VT52 (ESC / K)
*ESC]	HX25	Transmit 25th Line
*ESC #	HXMP	Transmit Page

The Heath-19 transmits the following sequences, but it will not respond to them if they are received. Kermit-MS will transmit them only if they are programmed with SET KEY.

ESC S	HF1	Function Key #1
ESC T	HF2	Function Key #2
ESC U	HF3	Function Key #3
ESC V	HF4	Function Key #4
ESC W	HF5	Function Key #5
ESC P	HF7	Function Key #7
ESC Q	HF8	Function Key #8
ESC R	HF9	Function Key #9

11. CP/M-80 KERMIT

Program:	Bill Catchings, Columbia University, with contributions from Bernie Eiben (DEC), Nick Bush (Stevens), John Bray (University of Tennessee), Bruce Tanner (Cerritos College), Greg Small (University of California at Berkeley), Kimmo Laaksonen (Helskini University of Technology), and many others.
Language:	8080 Assembler or MAC80
Version:	3.9A
Date:	6 June 1984
Documentation:	Frank da Cruz, Columbia University

KERMIT-80 Capabilities At A Glance:

Local operation:	Yes
Remote operation:	No
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
^X/^Y interruption:	Yes
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	No
Alternate block checks:	Yes
Terminal emulation:	Yes, VT52 and others
Communication settings:	Yes; duplex, parity
Transmit BREAK:	Yes; some versions
IBM communication:	Yes
Transaction logging:	No
Session logging (raw download):	Yes
Raw upload:	Yes
Act as server:	No
Talk to server:	Yes; SEND, GET, FIN, BYE
Advanced commands for servers:	No
Local file management:	Yes; DIR, ERA, SET DEFAULT disk
Handle file attributes:	No
Command/init files:	No
Printer control:	Yes, limited

Summary of CP/M

CP/M-80 (version 2.2) has only five built-in commands, and they all deal with files; other functions are done by invoking programs.

CP/M file specifications are of the form DEV:XXXXXXXX.YYY, where

DEV:	is a *device name*, normally the A: or B: floppy. If omitted, the device name defaults to your connected diskette.
XXXXXXXX	is a *filename* of up to 8 characters.

YYY	is the *file type*, up to 3 characters.

File names and file types may contain letters, digits, and some special characters, including dash, dollar sign, and underscore, but no imbedded spaces. Upper and lower case letters are equivalent.

"Wildcard" file-group specifications are permitted in file names and file types (but not device names) within certain contexts; a "*" matches a whole field, a "?" matches a single character, including space. Examples: "`*.F??`" specifies all files whose *types* start with F and are 1, 2, or 3 characters long; "`F?.*`" specifies all files whose names start with F and are no more than two characters long (before the trailing spaces).

The five CP/M commands are:

DIR *file*	Lists the the names of the specified files. The default file specification is "*.*". Example: "`DIR B:*.FOR`".
ERA *file*	Erases (deletes) the specified file(s); wildcards allowed.
REN *new old*	Changes the name of a file from *old* to *new*, e.g. "`REN NEW.FOR=OLD.FOR`".
SAVE	Saves the specified number of memory blocks into a file.
TYPE *file*	Types the specified file on the screen, e.g. "`TYPE FOO.TXT`".

The most important programs are:

STAT	Gives statistics on disk usage.
PIP	Peripheral Interchange Program. Copies files. In response to the "*" prompt, give a command of the form `disk:outfile=disk:infile` Wildcards ("*" for a whole field or "?" for a letter) can be used. Examples: "`A:=B:*.*`" to copy a whole disk, "`A:=B:*.FOR`" to copy all the Fortran programs from disk B to disk A. If the disk specification is omitted, your "connected" disk is assumed. Command line arguments are also accepted, e.g. "`PIP A:=B:*.*`".

For further information on CP/M, consult your microcomputer manual or a CP/M handbook.

KERMIT-80 Description

Implementations of Kermit-80 presently exist for the DEC VT180 (Robin), the DECmate II, the Intertec Superbrain, the Heath/Zenith 89 and Z100, the Apple II with the Z80 SoftCard, the Osborne 1, the TRS-80 II with CP/M, the Telcon Zorba, the Kaypro, the Vector Graphics CP/M system, the Morrow Decision I, the Nokia MikroMikko, and others. There is also a "generic" CP/M version that should run on most 8080-compatible CP/M 2.2 systems, but which may provide less performance, and another for CP/M 3.0.

Since Kermit-80 runs on a standalone micro, it is always in control of the screen -- it is always *local*. Thus, it always keeps the screen updated with the file name and the packet number, whether sending or receiving. Kermit-80 is capable of an imprecise or "fuzzy" timeout on an input request, and can break deadlocks automatically. In most cases, this is not important, because the KERMIT on the other side is most likely able to handle the timeouts. The timeouts done by Kermit-80 are fuzzy because they depend on the speed of the Z80 processor and other factors that can vary from system to system.

If despite the timeout capability, the transmission appears to be stuck (and you can tell that this has happened if the screen fails to change for a while) you can type carriage return to have the micro do what it would have done on a timeout, namely NAK the expected packet to cause to foreign host to send it again (or, if the micro is sending, to retransmit the last packet). Micro/micro or micro/IBM-mainframe transfers could require this kind of manual intervention.

File transfers may be interrupted in several ways.

Control-C	This will return you to Kermit-80 command level immediately, so that you can connect back to the remote system, or take any other desired action.
Control-X	When sending a file, this will terminate the sending of the current file with a signal to the KERMIT on the other side to discard what it got so far. If there are more files to be sent, KERMIT-80 will go on to the next one. When receiving a file, KERMIT-80 will send a signal to the remote KERMIT to stop sending this file. If the remote KERMIT understands this signal (not all implementations of KERMIT do), it will comply, otherwise the file will keep coming. In any case, the remote KERMIT will go on to the next file in the group, if any.
Control-Z	Like Control-X, except if a file group is being transmitted, this will stop the transmission of the entire group. If only a single file is being transmitted, it works exactly like Control-X.
Carriage Returns	If you type carriage return repeatedly Kermit-80 will retry the current packet up to its retry limit (somewhere between 5 and 16 times) and then, if no valid response was received, return to Kermit-80 command level.

Kermit-80 Commands

KERMIT-80 uses the DECSYSTEM-20 keyword style command language. Each keyword may be abbreviated to its minumum unique length. "?" may be typed to request a menu of the available options for the current field at any point in a command. ESC may be typed at any point in a command to fill out the current keyword or filename; if sufficient characters have not been typed to identify the current field uniquely, KERMIT-80 will sound a beep and allow you to continue from that point.

CONNECT Establish a "virtual terminal" connection to any host that may be connected to the serial port, i.e. pass all typein to the serial port and display all input from the serial port on the screen. Also, emulate a DEC VT52 to allow cursor control, screen clearing, etc., if VT52-EMULATION is ON (see below), in which case you should also set your terminal type on the remote host to VT52. (Some versions emulate other terminals.) The escape character differs from micro to micro; when you issue the CONNECT command, the micro will print a message telling you how to get back. The escape sequence is generally an uncommonly-used control character, like CTRL-backslash or CTRL-rightbracket, followed by a single letter "command".

C	Close Connection, return to `Kermit-80>` command level.
S	Display Status of connection, but maintain remote connection.
?	List available single-character commands.

0 (zero) Send a null (0) character.
B Send a BREAK signal. Only some systems provide this function.
^] (or whatever - a second copy of the escape character) Send the escape character itself to the remote host.

SEND *filespec*
Send file(s) specified by *filespec* to the remote Kermit. The *filespec* may contain CP/M wildcards.

RECEIVE Receive file(s) from the remote Kermit. Store them under the names provided in the file headers supplied by the remote host. If the names aren't legal, use as many legal characters from the name as possible (see the description of SET FILE-WARNING below). If there's a conflict, and FILE-WARNING is ON, warn the user and try to build a unique name for the file by adding "&" characters to the name.

GET *filespec*
When Kermit-80 is talking to a Kermit Server on the host, you should use the GET command to request the server to send files to you, for example: `get hlp:k*.hlp` Limitation: If you request an alternate block check type using the SET BLOCK command, the GET command will not communicate it to the remote server. If you want to have type 2 or 3 block checks done when getting files from the server, you have to issue the appropriate SET BLOCK command to the remote KERMIT before putting it in server mode.

LOG *filespec*
When CONNECTed to a foreign host as a terminal, log the terminal session to the specified diskette file. This functionality depends to some extent on the remote host's ability to do XON/XOFF flow control, and does not guarantee a complete transcript (after all, that's what the KERMIT protocol is for). The log file is closed when the connection is closed by typing the escape character followed by the single-character command "C".

TRANSMIT *filespec*
Send the specified file to the system on the other end of the connection as though it were being typed at the terminal, one line at a time. No KERMIT protocol is involved. You must manually confirm each line. This is useful for sending files to systems that don't have a KERMIT program. During transmission, you may type the escape character followed by one of these single-character commands:

C Cease transmission
R Re-transmit the previous line

BYE When talking to a remote Kermit Server, this command shuts down the server and logs it out, and also exits from Kermit-80 to CP/M command level.

LOGOUT Like BYE, but leaves you at Kermit-80 command level.

FINISH Like LOGOUT, but shuts down the remote server without logging it out. Leaves you at Kermit-80 command level; subsequent CONNECT commands will put you back at host system command level.

SET *parameter* [*value*]
Set the specified parameter to the specified value. Possible settings:

WARNING ON (or OFF)
Warn user of filename conflicts when receiving files from remote

host, and attempt to generate a unique name by adding "&" characters to the given name. ON by default.

VT52-EMULATION ON (or OFF)
: When connected as a terminal to a foreign host, controls whether the micro emulates a VT52 or runs in "native mode". VT52 emulation is ON by default, except on micros that already have terminal functionality built in, such as the DEC VT180 and DECmate (these act as VT100-series terminals). Some systems emulate other terminals, like the ADM3A.

LOCAL-ECHO ON (or OFF)
: When you CONNECT to a remote host, you must set LOCAL-ECHO ON if the host is half duplex, OFF if full duplex. OFF by default.

ESCAPE
: Change the escape character for virtual terminal connections. Kermit-80 will prompt you for the new escape character, which you enter literally.

BAUD
: Change the baud rate of the communications port. This command only works on some systems, and its actual operation can vary from system to system. Type SET BAUD followed by a question mark, and follow the directions. On systems that do not support this command, you must set the port baud rate from CP/M or other setup mechanism outside of KERMIT-80.

PARITY
: Sets parity for outgoing characters to one of the following: NONE, SPACE, MARK, EVEN, or ODD. On input, if parity is NONE, then the 8th bit is kept (as data), otherwise it is stripped and ignored. The parity setting applies to both terminal connection and file transfer. If you set parity to anything other than none, KERMIT-80 will attempt to use "8th bit prefixing" to transfer binary files. If the other KERMIT is also capable of 8th bit prefixing, then binary files can be transferred successfully; if not, the 8th bit of each data byte will be lost (you will see a warning on your screen if this happens).

TIMER ON (or OFF)
: Enable or disable the "fuzzy timer". The timer is off by default, because in the normal case KERMIT-80 is communicating with a mainframe KERMIT that has its own timer. Mainframe KERMIT timers tend to be more precise or adaptable to changing conditions. You should SET TIMER ON if you are communicating with a KERMIT that does not have a timer. You should SET TIMER OFF if you are communicating over a network with long delays.

IBM ON (or OFF)
: Allow the transfer of files to and from an IBM mainframe computer. This makes Kermit-80 wait for the IBM turnaround character (XON), ignore parity on input, add appropriate parity to output, and use local echoing during CONNECT. As distributed, KERMIT-80 uses MARK parity for IBM communication. If you don't give this command, IBM mode is OFF. Since IBM VM/CMS KERMIT does not have timeout capability, SET IBM ON also turns on the "fuzzy timer" automatically.

BLOCK-CHECK-TYPE
: The options are:

    1-CHARACTER-CHECKSUM

Normal, default, standard 6-bit checksum.

2-CHARACTER-CHECKSUM
: A 12-bit checksum encoded as two characters.

3-CHARACTER-CRC-CCITT
: A 16-bit CCITT-format Cyclic Redundancy Check, encoded as 3 characters.

The 2 and 3 character options should only be used under conditions of extreme line noise. Many implementations of KERMIT only support the single character checksum.

FILE-MODE
: Tells KERMIT-80 what kind of file it is sending, so that KERMIT can correctly determine the end of the file. SET FILE BINARY means to send all the 128-byte blocks of the file, including the last block in its entirety; SET FILE ASCII is used for text files, and transmission stops when the first Control-Z is encountered anywhere in the file (this is the CP/M convention for marking the end of a text file). If binary transmission is used on a text file, some extraneous characters (up to 127 of them) may appear at the end of the file on the target system. If ASCII transmission is used on a binary file, the entire file will not be sent if it happens to contain any data bytes that correspond to Control-Z.

DEFAULT-DISK
: This allows you to set the default disk as source and destination of file transfers. In addition, issuing this command causes you to switch to the specified disk and log it in, write-enabled. The selected disk appears in your KERMIT-80 prompt, for instance

```
Kermit-80 A:>
```

PORT
: Allows you to switch between different communication ports. This command is not available on all systems.

PRINTER
: ON or OFF. Turns copying of CONNECT session to printer on and off. No attempt is made to do buffering or flow control; it is assumed printer can keep up.

DIR
: This provides a directory listing of the specified files. If no files are specified, all files on the default disk are listed. File sizes, in K, are included. You may interrupt the listing at any time by typing any character. The listing (even if interrupted) concludes with a display of the amount of free storage left on the disk.

ERA
: This executes the CP/M ERA command on the specified file(s). The names of the files being erased are not displayed.

11.1. Generic KERMIT-80

"Generic Kermit-80" is a implementation of Kermit that should run on any 8080-compatible CP/M system no modification at all, or perhaps only a minor one. Unlike other Kermit-80 implementations, it contains no system-dependent manipulation of the serial port or screen. All I/O is done with standard CP/M BIOS calls, and I/O redirection is done using the CP/M IOBYTE function, which, according to the Digital Research *CP/M Operating System Manual*, is an optional feature of any particular CP/M implementation. If your system does not provide the IOBYTE function, Generic Kermit-80 will not work; furthermore, not all systems that implement IOBYTE do so in the same way.

The reason all Kermit-80 implementations aren't generic is that a good deal of speed is sacrificed by getting all services from the operating system. While a specific implementation of Kermit-80 may be able to operate at 4800, 9600, or even 19200 baud, Generic Kermit will fail to work on some systems at speeds in excess of 1200 baud.

Generic Kermit also differs from other Kermit-80 implementations in that it does not do fancy screen control during file transfer; it simply types the file names, packet numbers, and messages in sequence across and down the screen. This works best if you can put your micro or terminal in "autowrap" mode; otherwise the packet numbers will pile up in the rightmost column; the filenames and messages will always appear on a new line, however. Neither does generic Kermit-80 do termimal emulation; thus a generic Kermit-80 acts either as a "dumb terminal" (sometimes called a "glass TTY"), or else its own built in terminal firmware provides cursor control functions independent of the Kermit program.

Note that VT180 and DECmate-II Kermit are simply adaptations of Generic Kermit that do VT100 (ANSI) screen control during file transfer.

11.2. Installation

Kermit-80 was written originally for the Intertec SuperBrain in lowest-common-denominator 8080 code with the standard assembler, ASM (no macros, no advanced instructions), so that it can be assembled on any CP/M-80 system[16]. It has since been modified to run on many other systems as well. Kermit-80 should be able to run on any 8080-, 8085- or Z80-based microcomputer under CP/M with only minor modifications (see below).

All versions of Kermit-80 are assembled from the same source, with system dependencies taken care of by assembly-time conditionals. The most important system dependencies are terminal emulation (when CONNECTed to the remote host) and screen handling, which are dependent on the individual micro's escape codes (these features are table driven and easily modified for other CP/M systems), and the lowest level i/o routines for the serial communications port. The port routines are best done only with BDOS calls, but some systems do not allow this, primarily because the BDOS routines strip the parity bit during port i/o and the parity bit is used for data when transmitting binary files. Also, using BDOS calls, there's no way to poll the serial port; you must hang until input appears.

Kermit-80's i/o routines must check the port status and go elsewhere if no input is available; this allows for virtual terminal connection, keyboard interruption of stuck transmissions, etc. On systems that fully implement i/o redirection via the optional CP/M IOBYTE facility, this may be done by switching the IOBYTE definition. On others, however, IN/OUT instructions explicitly referencing the port device registers must be used.

CP/M-80 KERMIT version 3.8 and above includes a "fuzzy timer" that allows a timeout to occur after an interval ranging from 5 to 20 seconds (depending upon the speed of the processor and the operating system routines) during which expected input does not appear at the port. In this case, retransmission occurs automatically. In any case, you may type a carriage return during transmission to simulate a timeout when the transfer appears to be stuck.

---

[16]The 8080 assembler is distributed as a standard part of CP/M-80, whereas the fancier Z80 or macro assemblers are normally a commercial product

11.2.1. Downloading Kermit-80

If you already have a version of Kermit on your micro and you want to install a new version, simply use your present version to get the new one. If it's stored in the form of a .COM file, you can run it directly. If it's stored as a .HEX file, you must first LOAD it on your micro to produce a .COM file.

If you do not have a copy of KERMIT on your micro, and you cannot borrow a Kermit floppy but you do have access to a mainframe computer with a copy of the Kermit-80 distribution, you should read this section.

There are several ways to get Kermit from a host system to your micro. The easiest is to "download" the precompiled "hex" file into your micro's memory and then save it on the disk. The following is a procedure which, though far from foolproof, should allow you to get a version of Kermit to your CP/M based micro. It depends upon the host prompt, or at least the first character of the host prompt, being some character that cannot appear in a hex file (the valid characters for hex files are the digits 0-9, the upper case letters A-F, the colon ":", carriage return, and line feed). As soon as any other character is encountered, the transfer will terminate. If your host does not issue a prompt that will accommodate this scheme, you can achieve the same effect by adding an atsign "@" to the very end of the hex file before sending it from the host. The program below looks for an atsign (the normal DEC-20 prompt, hex 40). DECSYSTEM-10 users would look for a dot, hex 2E.

1. Look for the appropriate hex file in the host's KERMIT area. The name will be something like CPMROBIN.HEX, CPMHEATH.HEX, CPMOSBORN.HEX, etc. If you don't find it, but you do find a corresponding .ASM or .M80 file, you'll either have to build a new hex file on the host using a cross assembler (see below for how to do this on a DEC-10 or DEC-20), or else bring the M80 source file to your micro and assemble it there.

2. Connect to your host using a terminal or a terminal emulation facility. Ensure that your host does not have your terminal in "page mode". E.g. on the DEC-20, give the Exec command TERMINAL NO PAUSE END-OF-PAGE.

3. Tell the host to display the hex file at your terminal. E.g. on the DEC-20, give the Exec command TYPE KERMIT.HEX, *without a terminating carriage return.*

4. Return to your micro. Connect to a floppy disk with plenty of free space. Make sure your IOBYTE is set so that RDR: and PUN: correspond to the I/O port that is connected to the DEC-20 (this would normally be the case unless you have done something special to change things). Run DDT and type in the following (the comments should not be typed in; they are there just to tell you what's happening):

```
-ikermit.hex            ;Setup FCB for file KERMIT.HEX.
-a100                   ;Begin assembling code at 100.
0100 lxi h,ffe          ;Where to put HEX file.
0103 shld 300           ;Save the address.
0106 mvi e,d            ;Get a CR.
0108 mvi c,4            ;Output function.
010A call 5
010D mvi c,3            ;Input function.
010F call 5
0112 ani 7f             ;Turn off the parity.
0114 cpi 40             ;Our DEC-20 prompt atsign?
0116 jz 124             ;Yes, we have whole file.
0119 lhld 300           ;Get the pointer.
```

```
011C mov m,a            ;Else, store the char.
011D inx h              ;Increment the pointer.
011E shld 300           ;Save the pointer.
011F jmp 10d            ;Go around again.
0124 mvi a,1a           ;Get a control-Z.
0126 lhld 300           ;Get the pointer.
0129 mov m,a            ;Store the char.
012A shld 300           ;Save the pointer.
012D lxi h,1000         ;Pointer to file.
0130 shld 310           ;Save the pointer.
0133 mvi c,16           ;Make file.
0135 lxi d,5c
0138 call 5
013B lhld 310           ;Get the file pointer.
013E xchg               ;Put it in DE.
013F mvi c,1a           ;Set DMA.
0141 call 5
0144 mvi c,15           ;Write DMA to file.
0146 lxi d,5c
0149 call 5
014C lhld 310           ;Get the file pointer.
014F lxi d,80           ;Get the DMA size.
0152 dad d              ;Adjust file pointer.
0153 shld 310           ;Save it.
0156 lda 301            ;Get high order byte.
0159 cmp h              ;Have we passed the end?
015A jm 170             ;Yes.
015D jz 163             ;Maybe.
0160 jmp 13b            ;No.
0163 lda 300            ;Get low order byte.
0166 cmp l              ;Passed the end?
0167 jm 170             ;Yes.
016A jz 170             ;Yes.
016D jmp 13b            ;Not quite.
0170 mvi c,10           ;Close file.
0172 lxi d,5c
0175 call 5
0178 ret
0179
-g100,179               ;Execute the program.
-                       ;Reboot
```

Now there should be a file KERMIT.HEX on your connected disk.

5. Load this using the CP/M command LOAD to produce KERMIT.COM. This should be a runnable version of Kermit. Note that CP/M hex files have checksums on each line. If there were any transmission errors during the downloading process, the CP/M loader will notice a bad checksum and will report an error (something like "Illegal Format"). If you get any errors during loading, either fix the hex file locally with an editor, or repeat the previous step.

You now should have a running version of Kermit-80.

11.2.2. Building KERMIT.HEX

The source for Kermit-80 should be available on your host computer. It is written using 8080 assembler mnemonics, so it can be assembled on most 8080s and Z80s using the standard 8080 assembler provided with CP/M. If you have KERMIT.ASM on your CP/M system, you can assemble it directly on the micro using ASM, setting the desired assembly switches as explained below. If you don't have it, you can attempt to download the source file from the host using the procedure outlined above.

A cross assembler is provided that runs on the DEC-10 and DEC-20, called MAC80, contributed by Bruce Tanner at Cerritos College, that may be used for cross assembling KERMIT-80 as shown in this example (for TOPS-20):

1. Copy `PS:<KERMIT>CPMKxx.M80` to your directory, as `KERMIT.M80`. TOPS-10 filename rules must be followed. This is the 8080 assembler source file; *xx* is the current version number; for instance version 3.8 would be stored as `CPMK38.M80`.
2. Edit `KERMIT.M80` to set the conditional assembly switch for your machine. These are right near the top of the file. The switch for your machine should be set to TRUE and all the others to FALSE.
3. The "ibm-flag" setting is site dependent. As shipped from Columbia, it turns on half duplex line handshaking, using CTRL-Q as the turnaround character, sets LOCAL-ECHO ON, TIMER ON, and PARITY MARK. Make any changes required for your site.
4. The default FILE-MODE is ASCII as shipped from Columbia. This means that when text files are sent from the CP/M system, no extraneous characters will be sent after the end, but that outgoing binary files may be truncated erroneously if the user forgets to SET FILE BINARY. You can change the default to BINARY, so that no data is ever lost from any file, but text files will usually have extraneous junk at the end.
5. Run MAC80:

   ```
   @mac80
   *kermit=kermit
   *^Z
   ```

   The result will be in your directory as `KERMIT.HEX`.
6. Use Kermit, MODEM, or any other downloading mechanism to transfer `KERMIT.HEX` to the micro, or download it using the DDT program shown above.
7. On the micro, load the hex file.

   ```
   A>load kermit
   ```

   `KERMIT.COM` will appear on the floppy.
8. The new Kermit should be ready to run.

11.2.3. Generic Kermit-80

If your CP/M 2.2 system implements i/o redirection via the (optional) IOBYTE mechanism, you can probably run Generic Kermit on it, either without modification, or by a very simple change to the program. The standard CP/M IOBYTE is set up as follows:

```
I/O Byte assignments (four 2-bit fields for 4 devices at location 3)
:
bits 6+7        LIST field
   0            LIST is Teletype device (TTY:)
   1            LIST is CRT device (CRT:)
   2            LIST is Lineprinter (LPT:)
   3            LIST is user defined (UL1:)

bits 4+5        PUNCH field
   0            PUNCH is Teletype device (TTY:)
   1            PUNCH is high speed punch (PUN:)
   2            PUNCH is user defined #1 (UP1:)
   3            PUNCH is user defined #2 (UP2:)
```

```
bits 2+3        READER field
    0           READER is Teletype device (TTY:)
    1           READER is high speed reader (RDR:)
    2           READER is user defined #1 (UR1:)
    3           READER is user defined #2 (UR2:)

bits 0+1        CONSOLE field
    0           CONSOLE is console printer (TTY:)
    1           CONSOLE is CRT device (CRT:)
    2           CONSOLE in Batch-mode (BAT:);READER=Input,LIST=Output
    3           CONSOLE is user defined (UC1:)
```

(Here, bit zero is the least significant, "rightmost", bit).

I/O redirection is accomplished by switching the IOBYTE between two values, "batch i/o" and "normal i/o". In normal i/o mode, the keyboard is defined to be the console. In batch i/o mode, the serial port is defined to be the console. This switching is necessary because the console is the only device that can be tested to see if input is available, but KERMIT must shuttle back and forth between the keyboard *and* the serial port looking for input. Here are the batch and default i/o mode definitions used in "standard" Generic KERMIT-80:

```
batio   EQU     056H            ;I/O byte CON=BAT,LIST=CRT,READER=RDR
defio   EQU     095H            ;I/O byte CON=CRT,LIST=LPT,READER=RDR
```

Other systems may have other logical devices that point to the serial port, in which case you'll need to redefine these symbols to point to those devices and then reassemble the program (with assembly switch GENER set to TRUE, all others FALSE).

If your system runs CP/M 3.0, then the CPMPLUS version of KERMIT should run on your system without modification, except perhaps for screen control or baud rate setting.

12. CP/M-86 KERMIT

Authors:	Bill Catchings, Columbia University; Ron Blanford, University of Washington; Richard Garland, Columbia University.
Language:	Digital Research ASM86
Version:	2.7
Date:	May 1984
Documentation:	Frank da Cruz, Columbia

This version of KERMIT is designed to support any CP/M-86 system. So far it supports the DEC Rainbow-100 and the NEC Advanced Personal Computer (APC). It is very similar to CP/M-80 and MS DOS KERMIT.

CP/M-86 KERMIT-86 Capabilities At A Glance:

Local operation:	Yes
Remote operation:	No
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	Yes
^X/^Y interruption:	Yes
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	No
Alternate block checks:	No
Terminal emulation:	Yes, uses PC firmware (VT100)
Communication settings:	Yes; duplex, parity
Transmit BREAK:	Yes
IBM communication:	Yes
Transaction logging:	No
Session logging (raw download):	Yes
Raw upload:	No
Act as server:	No
Talk to server:	Yes; SEND, GET, FIN, BYE
Advanced commands for servers:	No
Local file management:	No
Handle file attributes:	No
Command/init files:	Yes
Printer control:	No

CP/M-86 Kermit closely resembles CP/M-80 Kermit, just as CP/M-86 is very similar to CP/M-80. In some respects, KERMIT-86 is superior to KERMIT-80: the program is better modularized to facilitate easy addition of support for new systems, alternate style user interfaces, etc; the port i/o is fully buffered and XON/XOFF is done to allow high-speed communication and accurate session logging. On the other hand, the current version does not include the CP/M-80 Kermit's local file management commands.

CP/M-86 KERMIT Description

Since Kermit-86 runs on a standalone micro, it is always in control of the screen -- it is always *local*. Thus, it always keeps the screen updated with the file name and the packet number, whether sending or receiving. Kermit-86 is capable of timing out an input request, and can thus break deadlocks automatically. In most cases, however, this is not desirable because the KERMIT on the other side is most likely better able to handle the timeouts; therefore, Kermit-86's timer is normally not used.

If despite the timeout capability, the transmission appears to be stuck (and you can tell that this has happened if the screen fails to change for a long while) you can type carriage return to have the micro do what it would have done on a timeout, namely NAK the expected packet to cause to foreign host to send it again (or, if the micro is sending, to retransmit the last packet). Micro/micro or micro/IBM-mainframe transfers could require this kind of manual intervention.

File transfers may be interrupted in several ways.

Control-C	This will return you to Kermit-86 command level immediately, so that you can connect back to the remote system, or take any other desired action.
Control-X	When sending a file, this will terminate the sending of the current file with a signal to the KERMIT on the other side to discard what it got so far. If there are more files to be sent, KERMIT-86 will go on to the next one. When receiving a file, KERMIT-86 will send a signal to the remote KERMIT to stop sending this file. If the remote KERMIT understands this signal (not all implementations of KERMIT do), it will comply, otherwise the file will keep coming. In either case, the remote KERMIT will go on to the next file in the group, if any.
Control-Z	Like Control-X, except if a file group is being transmitted, this will stop the transmission of the entire group. If only a single file is being transmitted it works exactly like Control-X.
Carriage Returns	If you type carriage return repeatedly Kermit-86 will retry the current packet up to its retry limit (somewhere between 5 and 16 times) and then, if no valid response was received return to Kermit-86 command level.

When KERMIT-86 is started, it looks for the file `KERMIT.INI`. If found, it executes KERMIT-86 commands from it before prompting you for commands.

12.1. Kermit-86 Commands

KERMIT-86 uses the DECSYSTEM-20 keyword style command language. Each keyword may be abbreviated to its minumum unique length. "?" may be typed to request a menu of the available options for the current field at any point in a command. ESC may be typed at any point in a command to fill out the current keyword or filename; if sufficient characters have not been typed to identify the current field uniquely, KERMIT-86 will sound a beep and allow you to continue from that point.

CONNECT Establish a "virtual terminal" connection to any host that may be connected to the serial port, i.e. pass all typein to the serial port and display all input from the serial port on the screen, using the system's own built-in support for ANSI (VT100-like) screen control. When you issue the CONNECT command, the PC will print a message telling you how to get back by typing an an escape sequence, an uncommonly-used control character, normally CTRL-backslash, followed by a single letter "command".

C	Close Connection, return to `Kermit-86>` command level.
?	List available single-character commands.
B	Send a BREAK signal.
Q	Quit logging the remote session.
R	Resume logging the remote session.
L	Toggle logging.
^\	(or whatever - a second copy of the escape character) Send the escape character itself to the remote host.

SEND *filespec*
Send file(s) specified by *filespec* to the remote Kermit, using the prevailing file mode (ASCII or BINARY; see SET). The *filespec* may contain CP/M wildcards.

RECEIVE Receive file(s) from the remote Kermit. Store them under the names provided in the file headers supplied by the remote host. If the names aren't legal, use as many legal characters from the name as possible (see the description of SET FILE-WARNING below). If there's a conflict, and FILE-WARNING is ON, warn the user and try to build a unique name for the file by adding "&" characters to the name.

GET *filespec*
When Kermit-86 is talking to a Kermit Server on the host, you should use the GET command to request the server to send files to you, for example: **`get hlp:k*.hlp`**

BYE When talking to a remote Kermit Server, this command shuts down the server and logs it out, and also exits from Kermit-86 to CP/M command level.

LOGOUT Like BYE, but leaves you at Kermit-86 command level.

FINISH Like LOGOUT, but shuts down the remote server without logging it out. Leaves you at Kermit-86 command level; a subsequent CONNECT command should put you back at host system command level.

EXIT Exit from KERMIT-86 back to CP/M.

QUIT Synonym for EXIT.

SET *parameter* [*value*]
Set the specified parameter to the specified value. Possible settings:

BAUD Change the baud rate of the communications port. This command only works on some systems, and its actual operation can vary from system to system. Type SET BAUD followed by a question mark, and follow the directions. On systems that do not support this command, you must set the port baud rate from CP/M or other setup mechanism outside of KERMIT-86.

DEBUG ON or OFF. If ON, displays incoming and outbound packets during file transfer. OFF by default.

ESCAPE Change the escape character for virtual terminal connections. Select a character in the control range that you will not be likely to need at the remote host; type the new character literally. Certain characters, like Control-X, cannot be specified.

FILE-TYPE Tells KERMIT-86 what kind of file it is sending, so that KERMIT can correctly determine the end of the file. SET FILE BINARY means to send all the 128-byte blocks of the file, including the last block in its entirety; SET FILE ASCII is used for text files, and transmission stops when the first Control-Z is encountered anywhere in the file (this is the CP/M convention for marking the end of a text file). If binary transmission is used on a text file, some extraneous characters (up to 127 of them) may appear at the end of the file on the target system. If ASCII transmission is used on a binary file, the entire file will not be sent if it happens to contain any data bytes that correspond to Control-Z. ASCII is the default.

FLOW-CONTROL
Select the desired type of flow control to be used on the communication line. The choices are NONE and XON/XOFF. XON/XOFF is the default. If the remote system is not full duplex or cannot do XON/XOFF, you should use NONE.

IBM ON (or OFF)
Allow the transfer of files to and from an IBM mainframe computer. This makes Kermit-86 wait for the IBM turnaround character (XON), ignore parity on input, add appropriate parity to output, and use local echoing during CONNECT. As distributed, KERMIT-86 uses MARK parity for IBM communication. If you don't give this command, IBM mode is OFF. Since IBM VM/CMS KERMIT does not have timeout capability, SET IBM ON also turns on the timeout facility automatically, as if you had typed "SET TIMER ON".

LOCAL-ECHO ON (or OFF)
When you CONNECT to a remote host, you must set LOCAL-ECHO ON if the host is half duplex, OFF if full duplex. OFF by default.

LOG Specify a log file on the current CP/M disk into which to record incoming characters during CONNECT. If the remote host can do XON/XOFF, then the log file will normally capture every character shown on the screen. When connected to the remote system, several single-character arguments to the connect escape character can be used to control logging -- Q (quit), R (resume), L (toggle). If you use R or L during connect without having previously specified a log file name, then **KERMIT.LOG** is used. An open log is closed when you escape back to the PC.

PARITY Sets parity for outgoing characters to one of the following: NONE, SPACE, MARK, EVEN, or ODD. On input, if parity is NONE, then the 8th bit is kept (as data), otherwise it is stripped and ignored. The parity setting applies to both terminal connection and file transfer. If you set parity to anything other than NONE, Kermit-86 will attempt to use "8th bit prefixing" to transfer binary files. If the other KERMIT is also capable of 8th bit prefixing, then binary files can be transferred successfully; if not, the 8th bit of each data byte will be lost (you will see a warning on your screen if this happens).

PORT Allows you to switch between different communication ports on the PC. This command is not available on all systems.

TIMER ON (or OFF)

Enable or disable the timeout facility. The timer is off by default, because in the normal case KERMIT-86 is communicating with a mainframe KERMIT that has its own timer. Mainframe KERMIT timers tend to be more precise or adaptable to changing conditions. You should SET TIMER ON if you are communicating with another KERMIT that does not have a timer. You should SET TIMER OFF if you are communicating over a network with long delays.

WARNING ON (or OFF)
Warn user of filename conflicts when receiving files from remote host, and attempt to generate a unique name by adding "&" characters to the given name. OFF by default.

SHOW Show the current settings of the SET parameters.

TAKE Take KERMIT-86 commands from the specified file. The file should not contain any TAKE commands; nested command files do not work.

12.2. Installation:

CP/M-86 KERMIT is broken up into several source modules:

`86KERCMD.A86`	Command parser
`86KERFIL.A86`	File handler
`86KERIO.xxx`	System Dependent I/O
`86KERMIT.A86`	Main Program
`86KERPRO.A86`	Protocol Module
`86KERTRM.A86`	Terminal Emulation
`86KERUTL.A86`	Utilities

The main program module, `86KERMIT.A86`, contains INCLUDE directives for the other files. The 86KERIO module is stored with suffixes that denote the machine for which the program is being built -- RB for Rainbow, APC for NEC APC. The program may be built on the CP/M-86 system by obtaining all the source files listed above, storing them on the current disk with the names indicated, renaming the appropriate `86KERIO.xxx` file to be `86KERIO.A86`, and then doing:

```
ASM86 86KERMIT $PZ   (takes about 6 minutes on the Rainbow)
GENCMD 86KERMIT      (takes less than a minute)
```

and, if desired,

```
REN KERMIT.CMD=86KERMIT.CMD
```

12.3. DEC Rainbow 100 Support

KERMIT-86 runs on the DEC Rainbow 100 or 100+ under CP/M-86/80, version 1 or 2, on the 8088 side. It uses the built-in firmware to emulate a VT102 ANSI terminal during CONNECT, and runs well at speeds up to 9600 baud.

You should be able to download the program using the old KERMIT on the Z80 side (Rainbow Kermit, VT180 Kermit, or generic CP/M-80 Kermit will do the job, but only under DEC CP/M-86/80 version 1.0), or an earlier version of KERMIT-86.

If you don't have an earlier version of KERMIT, then follow the directions for installing KERMIT-80 (yes, KERMIT-80) in the KERMIT-80 section of the KERMIT User Guide, but send the KERMIT-86 hex file instead. This works because the Rainbow can run CP/M-80 programs like DDT.

Another way to get KERMIT onto your Rainbow for the first time would be from a DEC VT-180 diskette. A VT-180 can use its own KERMIT to load RBKERMIT onto its disk, which can then be read directly by a Rainbow. Also, note that VT-180 KERMIT-80 can actually run on the Rainbow on the Z80 side under DEC CP/M-86/80 version 1 (but not version 2 or higher), at speeds of 1800 baud or lower.

12.4. NEC Advanced Personal Computer Support

(Contributed by Ron Blanford, University of Washington)

Currently only the standard serial port is supported, and not the H14 auxiliary port. The SET PORT command is not implemented.

While in Kermit's terminal emulation mode, local commands are initiated by a two-character sequence consisting of the "escape character" followed by one other character identifying the command. (Make the second character a '?' to see a list of the valid commands.) As distributed, the standard Kermit-86 uses the control-backslash character as the escape character in terminal mode. The trouble is that the CP/M-86 BIOS in the APC ignores a keyboard entry of Control-\ (i.e. holding down the CTRL key while striking the '\' key), making it difficult (impossible) to use this method to get out of terminal mode.

One solution is to perform a "SET ESCAPE ↑" command before entering terminal mode to change the escape character to a caret (or any other character the APC keyboard will generate). This command could be placed in your **KERMIT.INI** file for automatic execution every time Kermit is started.

The simpler solution is to realize that the character code for a Control-\ is a hexadecimal 1C, and that this is the code generated by the INS key on the numeric keypad. Once you can remember that every reference to Control-\ should be interpreted as a reference to the INS key, this is actually easier to use than the two-key Control-\ sequence.

In the standard CP/M-86 BIOS, the unshifted DEL key generates a Control-X character (hexadecimal 18). This is the CP/M command to erase the current input line, and is very useful for local processing. Most mainframes do not use the Control-X character at all, so it becomes much less useful during terminal emulation. The DEL character (hexadecimal 7F), on the other hand, is often used by mainframes and can only be generated on the APC by holding down the SHIFT key while striking the DEL key (this capability is not mentioned anywhere in the documentation).

Because the Control-X character is so seldom used while the DEL character is commonly used, the initialization procedure in Kermit-86 modifies the CP/M-86 BIOS so that the DEL key generates the DEL character whether shifted or not. Control-X can still be generated if necessary by holding down the CTRL key while striking the 'X' key. The CP/M-86 BIOS is returned to its original state when Kermit terminates.

The APC uses escape sequences which have been standardized by the American National Standards Institute (ANSI) to control cursor movement, screen erasing, and character attribute manipulation. Perhaps the best-known other terminal which follows ANSI guidelines is the DEC VT100. The APC only recognizes a few of the more important ANSI commands, and not the complete set which the VT100 supports.

The ANSI/VT100 features that the NEC APC supports are:

- direct cursor addressing (by row and column)
- relative cursor addressing (up, down, left, right)

- line erasing (cursor to end, beginning to cursor, entire line)
- screen erasing (cursor to end, beginning to cursor, entire screen)
- character attributes (underline, reverse video, blink, but not bold)

In addition, the first four grey function keys (unshifted) generate the escape sequences associated with PF1 through PF4 on the VT100 keyboard. The arrow keys and numeric keypad DO NOT generate the corresponding VT100 sequences.

These functions are enough to support simple command line editing on most systems, and allow mailers or paged file display programs to clear the screen before each display. Underlining and reverse video are also useful in some applications. This is not enough to support the more sophisticated screen control required by screen editors such as EMACS or KED. In addition, due to a bug in the implementation of the CP/M-86 BIOS, the sequence ordinarily used to home the cursor (esc [H) does not work correctly; a patch for CP/M to correct this problem is distributed with APC Kermit-86.

13. APPLE-DOS KERMIT

Authors: Antonino N. J. Mione, Stevens Institute of Technology
Peter Trei, Columbia University
Documentation: Antonino N.J. Mione, Stevens Institute of Technology
Peter Trei, Columbia University
Version: 2.1(45)
Date: July 1984

Kermit-65 Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
Transfers text files:	Yes
Transfers binary files:	Yes
Wildcard send:	No
^X/^Y interruption:	No
Filename collision avoidance:	Yes
Can time out:	No
8th-bit prefixing:	Yes
Repeat count prefixing:	No
Alternate block checks:	No
Terminal emulation:	Yes (VT52)
Communication settings:	Yes; local echo, parity
Transmit BREAK:	Yes
IBM communication:	Yes
Transaction logging:	No
Session logging (raw download):	No
Raw upload:	No
Act as server:	No
Talk to server:	Yes
Advanced commands for servers:	No
Local file management:	No
Handle file attributes:	No
Command/init files:	No
Printer control:	No

KERMIT-65 is a program that implements the KERMIT file transfer protocol for the Apple][micro computer system. It is written in 6502 assembly language and should run on any Apple][or Apple][Plus system running DOS 3.3. This section will describe the things you should know about the DOS 3.3 file system in order to make effective use of KERMIT, and then it will describe the special features of the KERMIT-65 program.

13.1. The DOS 3.3 File System

Items of importance which will be discussed in this section include Filenames and File Characteristics.

Apple DOS Filenames

Filenames under Apple DOS may contain almost any ASCII character (including space). It is not recommended that special characters, (i.e. control characters or spaces) be used in a filename to be transferred by Kermit-65 since they may cause problems when parsing the filename.

Filenames may be up to 40 characters in length. No wildcarding of any kind can be done in KERMIT-65.

Apple DOS File Characteristics

All files in Apple DOS have a file type associated with them which is contained in the directory entry for the file but is not part of the filename itself. There are four types of files in DOS 3.3. They are:

- APPLESOFT BASIC
- INTEGER BASIC
- BINARY
- TEXT

All file types have their data stored in eight-bit bytes although not all of them need the eighth bit. The two file types containing basic programs required the eighth bit due to the nature of the data being stored. BINARY files are images of memory copied into a file. Often, these are machine code programs. These files require all eight bits. TEXT files normally contain only printable or carriage control characters They are stored in the form of seven-bit ASCII characters but the eighth bit should always be set since Apples manipulate all text internally as 'NEGATIVE ASCII'.

When transmitting text files, the byte size should be set to seven-bit. When transmitting anything else the user must insure that both Kermits are handling eight bit data so that no information is lost. If an eight-bit data path is not available (i.e. the remote Kermit needs to do parity checking with the eigth bit), then eight-bit quoting should be used. Of course, BINARY files as well as Apple BASIC files will not have much meaning on a different system. If the user desires to edit a BASIC file on a mainframe, for instance, he must convert it to a TEXT file before sending it over. After receiving the file back on the Apple, the user may convert it back to BASIC once again. The reason BASIC files would be meaningless to a different machine is that the Apple stores BASIC keywords as single character tokens to save space and processing time. To convert a BASIC program to and from a TEXT file, consult the Apple DOS 3.3 Manual.

File information can be obtained by issuing the CATALOG command. For example:

```
]CATALOG

DISK VOLUME 010

  *A 002 HELLO
   B 078 KERMIT
   A 002 READER
   T 005 TESTFILE
```

]

When KERMIT-65 is receiving a file, the file it creates on diskette will be of the type indicated by the FILE-TYPE parameter. The file will always be left in an unlocked state after it is closed by KERMIT-65. When sending a file, KERMIT-65 will use the FILE-TYPE parameter to determine how to detect an End-of-file condition. Thus, it is important to have this set properly in all cases.

Recommendations for archiving files

When using a large system for archiving purposes, there is no reason to convert Apple Basic programs into text files before sending them since there is no need to edit them on the mainframe.

The FILE-TYPE parameter must always be set correctly when sending and receiving files. Also, when sending files which require eight-bit FILE-BYTE-SIZEs, this parameter must also be set properly since KERMIT-65 does not change it automatically based on FILE-TYPE.

The procedure for archiving TEXT files is:

- Run Kermit on remote system
- SET FILE-BYTE-SIZE SEVEN-BIT ! On KERMIT-65
- SET FILE-TYPE-MODE TEXT ! On KERMIT-65
- Send files

The procedure for archiving APPLESOFT, INTEGER, and BINARY files is:

- Run Kermit on remote system
- Set File-byte-size to Eight-bit ! On Remote Kermit
- SET FILE-BYTE-SIZE EIGHT-BIT ! On KERMIT-65
- SET FILE-TYPE-MODE APPLESOFT ! (or INTEGER, or BINARY) On KERMIT-65
- Send files

13.2. Program Operation

Hardware Considerations

Prior to using KERMIT-65 for transferring files, the modem interface must be set to handle data in a certain manner. Firstly, the data format should be 8 data bits and 1 stop bit. Secondly, the card should be set to no parity. The baud rate (if adjustable) must be set to whatever rate the modem can handle. For the Apple Com card and the D.C. Hayes Micromodem, these parameters are set correctly by default, so very little has to be done. For the Apple Super Serial Card, however, these have to be set before using KERMIT-65. In all cases, use the same procedure to connect to the mainframe host as is indicated in the section below on Installing KERMIT.

Some mainframes cannot handle data in the format of 8 data bits and 1 stop bit because they may need parity checking (i.e. most IBM machines). In this case, 7 data bits and 1 stop bit plus some parity setting (other than none) will usually work. When talking with such mainframes, binary and basic files on the Apple cannot be transferred unless Eighth-bit-quoting is used.

Conversing with KERMIT-65

KERMIT-65's prompt is "KERMIT-65>".

To run KERMIT-65 and issue commands to it, type the following:

```
]BRUN KERMIT

STEVENS/CU - APPLE ][ KERMIT-65 - VER 2.1

Kermit-65>SEND TESTFILE

file is sent

Kermit-65>STATUS

performance statistics are printed

Kermit-65>Other commands
            .
            .
            .

Kermit-65>EXIT
]
```

KERMIT-65 uses a TOPS-20 style command parser.

During interactive operation, you may use the ?-prompting help feature ("?") and recognition (ESC) features while typing commands. A question mark typed at any point in a command displays the options available at that point; typing an ESC character causes the current keyword to be completed (or default value to be supplied). If you have not typed sufficient characters to uniquely specify the keyword or filename (or if there is no default value) then a beep will be sounded and you may continue typing. Keywords may be abbreviated to any prefix that is unique.

There are several different Apple][s which can run KERMIT-65. Kermit will have no problems running on an Apple][or Apple][+ system. It will run on the Apple //e as well, however, the 80-column board cannot be used at this time.

Of the different communication devices available for the Apple][, three are supported:

- Apple Com Card
- D.C. Hayes Micromodem
- Apple Super Serial Card

It is possible that other cards may have operational characteristics very similar or identical to one of the devices above. If this is the case, it may work using one of the currently available device drivers. The user may want to try each of the above options to see if any of them work.

KERMIT-65 must be told which device is being used so that it may run with the correct device driver. It also must be told in which slot the card resides. This may be done with the 'SET' .command (documented below).

13.3. Remote and Local Operation

KERMIT-65 is normally run in local mode. It may be run as a remote Kermit as well although there is no advantage to doing things that way. KERMIT-65 supports User-mode commands for talking to a Server. It does not currently support server mode.

13.4. KERMIT-65 Commands

The SEND Command

Syntax: SEND *filespec*

The SEND command causes a file to be sent from the Apple to the remote system. The Filespec is the name of the file on the Apple diskette to be sent. The parser will not accept control characters and certain special characters in a filename (i.e. a comma), so the user may have to rename the file before it is sent. The user may also have problems in filename compatibility with remote Kermits. If the remote Kermit does not have the facilities to beat the filename into a format that its system likes, the user may have to rename the file before sending it. The default disk drive used for file transfers is the drive used to boot the system or the last drive accessed with a DOS command. This can be changed with the 'SET DEFAULT-DISK' command (explained below). Either the slot or the drive or both may be altered.

As a file is being sent, the screen displays either 'SENDING PACKET...' or 'WAITING PACKET...' followed by the absolute packet number since start of transmission. If a packet must be transmitted several times and it reaches the maximum retry count, the transfer will fail and the 'KERMIT-65>' prompt will return. If the remote Kermit sends an error packet, the text of the packet will be displayed on the screen and the prompt will return.

Currently, a packet can be retransmitted manually by typing anything on the keyboard. If a 'Q' is typed, the entire transmission will be aborted.

The RECEIVE Command

Syntax: RECEIVE [*filespec*]

The RECEIVE command tells KERMIT-65 to receive a file or file group from the other system. If only one file is being received, you may include the optional *filespec* as the name to store the incoming file under; otherwise, the name is taken from the incoming file header. If the name in the header is not a legal filename in DOS 3.3, KERMIT-65 will attempt to change it into something legal. There are very few things that are illegal in DOS 3.3. If FILE-WARNING is on and an incoming file has a name identical to a file already existing on the diskette, KERMIT-65 will issue a warning to the user and attempt to modify the filename to make it unique.

GET

Syntax: GET *remote-filespec*

The GET command requests a remote KERMIT server to send the file or file group specified by *remote-filespec*. This command can be used with a KERMIT server on the other end.

The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally.

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

The CONNECT Command

Syntax: **CONNECT**

Establish a terminal connection to the remote system. Get back to KERMIT-65 by typing the escape character followed by the letter C. The escape character is Control-@ by default. When you type the escape character, several single-character commands are possible:

B	Send a BREAK Signal.
C	Close the connection and return to KERMIT-65.
S	Show status of the connection.
0	Send a null.
Connect-escape	Send the Connect-escape character itself.
?	List all the possible single-character arguments.

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

When 'CONNECTED', KERMIT-65 will be passing characters entered on the keyboard to the remote system, and passing characters from the remote system to the Apple screen. If VT52-EMULATION is turned on, Kermit will trap escape codes and simulate the appropriate functions of a VT52 terminal.

On an Apple][+ with an incomplete keyboard, special characters can be obtained by prefixing regular characters with a right-arrow. Also, Uppercase is shown in inverse and lowercase characters are displayed as normal uppercase characters.

Here are the rules for using the special 2/2+ input, to get all printable ASCII characters, and how they appear on the screen.

Special meanings are applied in various contexts to certain characters. The left and right arrow keys do special things, and sometimes the escape key does as well.

For letters, the keyboard is always in either default UPPERCASE mode or default lowercase mode. When in UPPERCASE, all letters typed are sent out as uppercase. In lowercase, all letters are sent as lowercase. To reverse the case for the next character only, hit the right-arrow ("prefix") key. To switch the default case, hit the prefix-key twice in a row.

For funny characters, the prefix key is also used to get the unusual punctuation characters which are not on the Apple keyboard. Here they are: (To represent the prefix character I am using the letter p).

To get	Type	Appearence
Left Square Bracket	p(	[
Right Square Bracket	p)	]
Left Curly Bracket	p<	{
Right Curly Bracket	p>	}
Underline	p-	
Backslash	p/	\
Tilde (wiggle)	p^	~
Vertical Line	p.	|

The left-arrow key sends a rubout.

With left-arrow and right arrow doing special things, its a little hard to enter their characters (↑H and ↑U respectivly). There is therefore an escape from prefix mode sequence. If you type prefix-ESC, the next character is sent without any interpretation.

Currently, it is impossible to turn this special input and display on and off, so its a bit of a pain if you are on a Apple 2e.

The HELP Command

Syntax: **HELP**

Typing HELP alone prints a brief summary of the KERMIT-65 commands.

The EXIT and QUIT Commands

Syntax: **EXIT**

Exit from KERMIT-65. You can restart the program, provided you haven't run anything else, by typing 'CALL 2049'.

Syntax: **QUIT**

This is merely a synonym for EXIT.

The SET Command

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

Establish or modify various parameters for file transfer or terminal connection. You can examine their values with the SHOW command. The following parameters may be SET:

DEBUGGING	TERSE or VERBOSE packet information.
DEFAULT-DISK	Which Diskette drive is used for file transfer?
DEVICE-DRIVER	Which communication device is being used?
EIGHT-BIT-QUOTING	Should eight-bit-quoting be used?
ESCAPE	Character for terminal connection.
FILE-BYTE-SIZE	SEVEN or EIGHT significant bits in a byte.
FILE-TYPE	Of Apple DOS file being sent/received.
FILE-WARNING	Warn users if incoming file exists?
IBM	For communicating with an IBM mainframe
KEYBOARD	][+ or //e keyboard.
LOCAL-ECHO	Full or half duplex switch.
PARITY	Character parity to use
RECEIVE	Various parameters for receiving files
SEND	Various parameters for sending files
SLOT	Which slot # is communication device in?
VT52-EMULATION	Should Kermit emulate a VT52 when connected?

SET DEBUGGING

Syntax: **SET DEBUGGING** *options*

Record the packet traffic on your terminal. Options are:

TERSE	Show packet info only (brief).
VERBOSE	Display packet field descriptions with packet info (lengthy).
OFF	Don't display debugging information (this is the default).

SET DEFAULT-DISK

Syntax: **SET DEFAULT-DISK** *parameter value*

This command will tell KERMIT-65 which disk drive should be used for file transfers. The two parameters which may be set separately are SLOT and DRIVE. The value for SLOT ranges from 1 to 7. The value for DRIVE is either 1 or 2.

SET DEVICE-DRIVER

Syntax: **SET DEVICE-DRIVER** *parameter keyword*

This command will tell KERMIT-65 what type of communication device is being used. Currently, three different cards are supported, however, other unsupported cards may work similar enough to one of the three available that it may be possible to use them. KERMIT-65 must also be told where the card is in the machine (see the SET SLOT command). The options for this set command are:

APPLE-COM-CARD	The old Apple communication card (300 baud).
DC-HAYES	The D.C. Hayes Micromodem II (300 baud).
SUPER-SERIAL-CARD	The Apple Super Serial Card (300-19.2k baud).

SET ESCAPE

Syntax: **SET ESCAPE** *hexidecimal-number*

Specify the control character you want to use to "escape" from remote connections back to KERMIT-65. The default is 0 (Control-@). The number is the hex value of the ASCII control character, 1 to 37, for instance 2 is Control-B.

SET FILE-BYTE-SIZE

Syntax: **SET FILE-BYTE-SIZE** *parameter*

Byte size for Apple DOS file I/O. The choices are SEVEN-BIT or EIGHT-BIT.

SEVEN-BIT
: When sending a file, shut the H.O. bit. When receiving, turn the H.O. bit on. This is done since text files are the only files which should be sent as SEVEN-BIT files and text files only make sense to the Apple if they are encoded in 'negative ascii' values.

EIGHT-BIT Always send and receive the bytes intact. All 8 bits are significant.

SET FILE-TYPE

Syntax: **SET FILE-TYPE** *parameter keyword*

This will inform KERMIT-65 what type of DOS file is being sent or received. It is important that this is set correctly since KERMIT-65 must create a file of the appropriate type when receiving (and it has no way of knowing what kind of file it is). When KERMIT-65 is sending, it must also know the type of file since that tells it how to detect the actual end-of-file. The options for this parameter are APPLESOFT, INTEGER, TEXT and BINARY.

APPLESOFT	The file being sent/received is an Applesoft Basic program.
INTEGER	The file being sent/received is an Integer Basic program.
TEXT	The file being sent/received is an ASCII Text file.
BINARY	The file being sent/received is a Binary image.

SET FILE-WARNING

Syntax: **SET FILE-WARNING ON** *or* **OFF**

This tells KERMIT-65 whether to warn the user about incoming filenames conflicting with existing files or not.

SET IBM

Syntax: **SET IBM ON** *or* **OFF**

SET IBM actually sets a number of parameters. When setting it on, the command also turns on LOCAL-ECHO (half-duplex) and sets PARITY to MARK. When setting IBM OFF, LOCAL-ECHO will revert back to OFF and PARITY will be set to NONE. It should be used when doing file transfers with an IBM or similar mainframe.

SET KEYBOARD

Syntax: **SET KEYBOARD 2P** *or* **2E**

SET KEYBOARD tells KERMIT-65 if the user has a full keyboard (2E) or not (2P). If the user is on an Apple][+, this should be set to 2P (which is the default). When set to that, certain character translations are available by using the right-arrow key as a prefix character.

SET SLOT

Syntax: **SET SLOT** *parameter*

This option tells KERMIT-65 in which slot the communication device is located. The range for the parameter is 1-7.

The SHOW Command

Syntax: SHOW [*option*]

The SHOW command displays various information:

ALL	All parameter settings (this is quite long).
DEBUGGING	Debugging mode.
DEFAULT-DISK	Which Diskette drive is used for file transfer?
DEVICE-DRIVER	Which communication device is being used?
EIGHT-BIT-QUOTING	Should eight-bit-quoting be used?
ESCAPE	Character for terminal connection.
FILE-BYTE-SIZE	SEVEN or EIGHT significant bits in a byte.
FILE-TYPE	Of Apple DOS file being sent/received.
FILE-WARNING	Warn users if incoming file exists?
IBM	For communicating with an IBM mainframe
KEYBOARD	][+ or //e keyboard.
LOCAL-ECHO	Full or half duplex switch.
PARITY	Character parity to use
RECEIVE	Various parameters for receiving files
SEND	Various parameters for sending files
SLOT	Which slot # is communication device in?
VT52-EMULATION	Should Kermit emulate a VT52 when connected?

The above options are analogous to the equivalent SET commands.

The STATUS Command

Give statistics about the most recent file transfer. This includes information such as number of characters sent/received, number of data characters sent/received, and last error encountered.

13.5. Customizing, Building, and Installing KERMIT-65

Standard Installation

The Procedure

The procedure to bootstrap an assembled KERMIT object file to the Apple is as follows:

1. On the Apple, Type in the APPLBT.BAS program supplied (See below). It is recommended that the user save this program as it may be needed to bootstrap newer versions of KERMIT or APPHXL in the future. Also, type the APPLESOFT program in with none of the REMs. It will execute quicker and take up less room.

2. Call and login to the mainframe on which the KERMIT-65 object file resides. Do the following:

-]IN#n ! Where n is between 1 and 7
- For Communication card, do the following:
  - Dial number for computer system.
  - Seat phone receiver in modem cradle.
  -]<Cntrl/A><Cntrl/F> !Full duplex, 300 baud
- For the D.C. Hayes Micromodem, do the following:
  -]<Cntrl/A><Cntrl/F> ! Full duplex, 300 baud
  - MICROMODEM II: BEGIN TERM <Cntrl/A><Cntrl/Q>
  - MICROMODEM II: DIALING: nnn-nnnn<cr> ! nnn-nnnn is number of computer system
- For the Apple Super Serial Card, do the following:
  -]<Cntrl/A>
  - APPLE SSC:0D ! 8 data bits, 1 stop bit
  -]<Cntrl/A>
  - APPLE SSC:0P ! Force no parity
  -]<Cntrl/A>
  - APPLE SSC:nB ! n = 6 for 300 baud, n = 8 for 1200 baud
  - Dial number for computer system.
  - Seat phone receiver in modem cradle.
  -]<Cntrl/A>
  - APPLE SSC:T ! Terminal mode, now talking to remote host

3. In your directory on the mainframe, the following files should be present:

   - APPLBT.FOR
   - APPHXL.HEX
   - APPLEK.HEX

   Compile and execute APPLBT.FOR. This will be used along with APPLBT.BAS on the Apple to load the APPHXL program. Once APPLBT is executing on the mainframe, give control back to the Apple and then run APPLBT.BAS on the Apple. For either the Communication Card or the D.C. Hayes Micromodem, the procedure is:

   - <Cntrl/A><Cntrl/X><cr> ! Give control to Apple's Brain
   -]LOAD APPLBT.BAS<cr>
   -]LOMEM:9500<cr>
   -]RUN<cr>

4. Relocate and save APPHXL. Type the following:

   -]CALL -151<cr> ! Enter Apple's system monitor

- *9000<2000.2280M<cr> ! Move APPHXL from $2000 to $9000
- *<Cntrl/C><cr> ! Reenter APPLESOFT
-]BSAVE APPHXL,A$9000,L$280<cr> ! Save APPHXL to disk

5. Now simply start executing APPHXL.

   -]CALL -151<cr> ! Enter monitor
   - *9000G<cr> ! Start APPHXL
   - SLOT FOR MODEM CARD? (1 TO 7)n ! 'n' is slot # of card (no <cr>)
   - ENTER FILENAME TO LOAD APPLEK.HEX<cr> ! Tell APPHXL what to load

   APPHXL will print the byte count and load address for each line it is receiving as it loads it into memory. At the end of each line, it will print '[OK]' if the line was received properly or '[CHECKSUM ERROR]' if there was a problem with it.

6. When APPHXL finishes type the following to the Apple:

   ```
   ]BSAVE KERMIT,A$801,L$4E00<cr> ! Save KERMIT to disk
   ```

7. The user may set up a turn-key system by having the hello file on the disk load and run KERMIT. The user may also run KERMIT-65, change the defaults which are supplied in the program such as SLOT and DEVICE-DRIVER, and then resave KERMIT-65. The next time it is run, the user will not have to set these values again. If the user does not set up a turn-key system, he must start KERMIT-65 by typing:

```
]BRUN KERMIT<cr>         ! Execute KERMIT-65 on the Apple
```

The Apple will display the following:

```
                STEVENS/CU - APPLE ][ KERMIT-65  VER. 2.1
                KERMIT-65>
```

The user is now ready to transfer files.

NOTE: For those users with the Apple Super Serial Card, the <cntrl/A> intercept character should be shut off by typing <cntrl/A>Z to the card before Kermit is run. The reason for this is that Kermit uses <cntrl/A> as a Start-of-Header character and this character may never reach the program if the card is taking it to be a command.

APPLBT.BAS - The APPLESOFT Bootstrap program

```
10  REM  - LOADER FOR HXLOAD
11  OAD = 0
100 N$ = "0123456789ABCDEF"
110 D$ =  CHR$ (4)
130  PRINT D$;"IN#2" : REM CHANGE '2' TO SLOT OF COMM. CARD IF NECESSARY
135  PRINT  CHR$ (1); CHR$ (6)
136  PRINT D$;"PR#2" : REM CHANGE '2' TO SLOT OF COMM. CARD IF NECESSARY
137  PR#2  : REM THIS LINE SHOULD BE HERE ONLY FOR THE APPLE COMM. CARD
140 C3 = 0
150  HOME
199  REM  - REQUEST NEXT LINE
200  REM - PUT A DOT ON THE SCREEN FOR EACH LINE RECEIVED
201 C3 = C3 + 1: POKE 1024 + C3, ASC (".")
202 L$ = "":Y2% = 1: PRINT
203  GET A$:L$ = L$ + A$:Y2% = Y2% + 1: IF Y2% < 81 THEN 203
205 C1 = 0:C2 = 0:I = 0
208  IF  LEFT$ (L$,1) >  = "0" AND  LEFT$ (L$,1) <  = "9" THEN 220
210 L$ =  RIGHT$ (L$, LEN (L$) - 1): GOTO 208
220 LL =  LEN (L$)
249  REM - FETCH THE DATA BYTE COUNT FOR THIS LINE
250  GOSUB 1000:C1 = C1 + B:C0 = B
```

```
255  IF CO = 0 THEN 990
259  REM - CONSTRUCT THE LOAD ADDRESS FOR THIS LINE
260  GOSUB 1000:C1 = C1 + B:AD = B: GOSUB 1000:C1 = C1 + B
     :AD = AD * 256 + B
265 REM - IF THE LATEST VERSION OF CROSS IS USED, THIS SHOULD NOT
     BE NEEDED : REM AD = AD - 28672
266  IF AD < OAD THEN 990
267 OAD = AD
270  FOR X = 0 TO CO - 1
275  REM - GO GET A BYTE AND PUT IT IN THE NEXT MEMORY LOCATION
280  GOSUB 1000:C1 = C1 + B
290  POKE AD + X,B
300  NEXT X
310  GOSUB 1000:C2 = B: GOSUB 1000:C2 = C2 * 256 + B
320  IF C1<>C2 THEN POKE 1024+C3,ASC("E")
330  GOTO 201
990  FOR X = 1 TO 1000: NEXT X
995  PRINT D$;"IN#0": PRINT D$;"PR#0": HOME : END
999  REM  - GET BYTE
1000  GOSUB 1501:B = N1: GOSUB 1501:B = B * 16 + N1
1010  RETURN
1500  REM - GET NIBBLE
1501  IF  LEN (L$) = 0 THEN N1 = 0: RETURN
1510 H$ =  LEFT$ (L$,1)
1511  IF  LEN (L$) = 1 THEN L$ = "": GOTO 1525
1515 L$ =  RIGHT$ (L$, LEN (L$) - 1)
1520  REM - RETURN VALUE OF HEX NIBBLE
1525  FOR X1 = 1 TO 16
1530  IF H$ =  MID$ (N$,X1,1) THEN 1610
1540  NEXT X1
1550 REM - DIGIT WAS NOT FOUND, RETURN ZERO
1560 N1 = 0: RETURN
1600  REM
1610 N1 = X1 - 1: RETURN
```

APPLBT.FOR - The Mainframe Bootstrap program

```
      CHARACTER LINE*80,SENTNL*1
      OPEN (UNIT=00,FILE='APPHXL.HEX',MODE='ASCII')
   10 READ (UNIT=05,FMT=20) SENTNL
   20 FORMAT (A1)
      READ (UNIT=00,FMT=25,END=899) LINE
   25 FORMAT(A80)
      WRITE (UNIT=05,FMT=30) LINE
   30 FORMAT(A80)
      GO TO 10
  999 READ (UNIT=05,FMT=20) SENTNL
      STOP
      END
```

Alternative Installation Procedures

HEXloading from Diskette

Once the user has a working version of KERMIT-65 on his system, he can use this to load the .HEX files of future versions of KERMIT. There is another hexload program available called APPDXL. This program will load a hex file from an Apple diskette into memory. To use this procedure, do the following:

1. Start executing APPHXL.

   -]BLOAD APPHXL ! Get APPHXL into memory
   -]CALL -151<cr> ! Enter monitor
   - *9000G<cr> ! Start APPHXL
   - SLOT FOR MODEM CARD? (1 TO 7)n ! 'n' is slot # of card (no <cr>)
   - ENTER FILENAME TO LOAD APPDXL.HEX<cr> ! Tell APPHXL what to load

   APPHXL will print what it is receiving on the screen as well as loading it into memory.

2. Relocate and save APPDXL. Type the following:

   -]CALL -151<cr> ! Enter Apple's system monitor
   - *9000<2000.2500M<cr> ! Move APPDXL from $2000 to $9000
   - *<Cntrl/C><cr> ! Reenter APPLESOFT
   -]BSAVE APPDXL,A$9000,L$500<cr> ! Save APPDXL to disk

3. Use Kermit-65 to transfer the new version of itself over. Make the Apple file a Text file. WARNING: This file will take LOTS of space (about 180 sectors) so make sure the disk is reasonably empty. The transfer takes a LONG time also, so please be patient.

4. Start executing APPDXL.

   -]BRUN APPDXL<cr> ! Start APPDXL
   - ENTER FILENAME TO LOAD APPLEK.HEX<cr> ! Tell APPDXL what to load

5. When APPDXL finishes type the following to the Apple:

```
]BSAVE KERMIT,A$801,L$4E00<cr> ! Save KERMIT to disk
```

The new version of Kermit is now on disk.

Using KERMIT-65 to transfer APPLEK.BIN

There is yet another way to Bootstrap a new version of KERMIT onto an Apple. If the user has an older version of KERMIT-65 and has access to a machine with a valid copy of APPLEK.BIN, they can simply transfer APPLEK.BIN using their version of KERMIT. Be sure to set the File-byte-size to Eight-bit, and the File-type-mode to Binary before transfering the file since this is the actual object code. No special loading or conversion is needed. The file will be placed on the disk and ready to run.

Files Supplied for KERMIT-65

The following files should be supplied on the distribution tape:

- APPLBT.BAS - Initial bootstrap program to load APPHXL
- APPLBT.FOR - Program on mainframe to talk to APPLBT.BAS
- APPHXL.M65 - Source of program to load KERMIT-65
- APPHXL.HEX - Assembled version of Hex load program
- APPDXL.M65 - Source of program to load KERMIT-65 from Apple diskette
- APPDXL.HEX - Assembled version of Disk Hex load program
- APPLEK.M65 - Source for the KERMIT-65 program
- APPLEK.HEX - Assembled version of KERMIT-65
- APPLEK.BIN - Assembled version of KERMIT-65 (Eight-bit Binary object code)
- CROSS.MAC - CROSS Microprocessor Assembler (Source)
- CROSS.EXE - CROSS Microprocessor Assembler (Object)

Customizing and Building KERMIT-65

The source code to KERMIT-65 is in 6502 Assembler. It has been formatted for CROSS which is a micro-Cross Assembler program which runs on DECsystem-10s and DECSYSTEM-20s. Customizations would be made the easiest if CROSS were available.

KERMIT-65 currently supports the following communications devices:

- FTASER - The Apple Communication card
- FTHAYS - The D.C. Hayes Micromodem.
- FTSSC - The Apple Super Serial Card

All device drivers are included in the assembled version and may be used by issuing a 'SET DEVICE-DRIVER' command to Kermit. If any of the device drivers are not needed, it(they) may be excluded by setting the appropriate feature test to zero in the Feature test section of the source code. Excluding one or more device drivers can reduce the size of the object code greatly. DO NOT disable all device drivers since KERMIT-65 will then have no way of talking over the communication device.

The feature test FTCOM must be set to the type of computer for which KERMIT-65 is being assembled. The only machine KERMIT-65 is available for currently is the Apple][. This parameter must be set to FTAPPL.

After setting any options necessary in APPLEK.M65, rename it to KERMIT.M65, and do the following:

- .R CROSS<cr> ! Run CROSS Microprocessor Assembler
- *KERMIT.HEX/PTP:KIM=KERMIT.M65/M65 ! Generate .HEX file

This command will produce an ASCII HEX file which can be downline loaded onto the Apple using APPHXL. If a listing is desired, one can be produced by adding ",KERMIT.LST" after the "/PTP:KIM" in the command line to CROSS.

APPENDIX I THE ASCII CHARACTER SET

ASCII Code (ANSI X3.4-1968)

There are 128 characters in the ASCII (American national Standard Code for Information Interchange) "alphabet". The characters are listed in order of ASCII value; the columns are labeled as follows:

Bit	Even parity bit for ASCII character.
ASCII Dec	Decimal (base 10) representation.
ASCII Oct	Octal (base 8) representation.
ASCII Hex	Hexadecimal (base 16) representation.
EBCDIC Hex	EBCDIC hexadecimal equivalent for Kermit translate tables.
Char	Name or graphical representation of character.
Remark	Description of character.

The first group consists of nonprintable 'control' characters:

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
0	000	000	00	00	NUL	^@, Null, Idle
1	001	001	01	01	SOH	^A, Start of heading
1	002	002	02	02	STX	^B, Start of text
0	003	003	03	03	ETX	^C, End of text
1	004	004	04	37	EOT	^D, End of transmission
0	005	005	05	2D	ENQ	^E, Enquiry
0	006	006	06	2E	ACK	^F, Acknowledge
1	007	007	07	2F	BEL	^G, Bell, beep, or fleep
1	008	010	08	16	BS	^H, Backspace
0	009	011	09	05	HT	^I, Horizontal tab
0	010	012	0A	25	LF	^J, Line feed
1	011	013	0B	0B	VT	^K, Vertical tab
0	012	014	0C	0C	FF	^L, Form feed (top of page)
1	013	015	0D	0D	CR	^M, Carriage return
1	014	016	0E	0E	SO	^N, Shift out
0	015	017	0F	0F	SI	^O, Shift in
1	016	020	10	10	DLE	^P, Data link escape
0	017	021	11	11	DC1	^Q, Device control 1, XON
0	018	022	12	12	DC2	^R, Device control 2
1	019	023	13	13	DC3	^S, Device control 3, XOFF
0	020	024	14	3C	DC4	^T, Device control 4
1	021	025	15	3D	NAK	^U, Negative acknowledge
1	022	026	16	32	SYN	^V, Synchronous idle
0	023	027	17	26	ETB	^W, End of transmission block
0	024	030	18	18	CAN	^X, Cancel
1	025	031	19	19	EM	^Y, End of medium
1	026	032	1A	3F	SUB	^Z, Substitute
0	027	033	1B	27	ESC	^[, Escape, prefix, altmode
1	028	034	1C	1C	FS	^\, File separator
0	029	035	1D	1D	GS	^], Group separator
0	030	036	1E	1E	RS	^^, Record separator
1	031	037	1F	1F	US	^_, Unit separator

The last four are usually associated with the control version of backslash, right square bracket, uparrow (or circumflex), and underscore, respectively, but some terminals do not transmit these control characters.
The following characters are printable:

First, some punctuation characters.

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
1	032	040	20	40	SP	Space, blank
0	033	041	21	5A	!	Exclamation mark
0	034	042	22	7F	"	Doublequote
1	035	043	23	7B	#	Number sign, pound sign
0	036	044	24	5B	$	Dollar sign
1	037	045	25	6C	%	Percent sign
1	038	046	26	50	&	Ampersand
0	039	047	27	7D	'	Apostrophe, accent acute
0	040	050	28	4D	(	Left parenthesis
1	041	051	29	5D	)	Right parenthesis
1	042	052	2A	5C	*	Asterisk, star
0	043	053	2B	4E	+	Plus sign
1	044	054	2C	6B	,	Comma
0	045	055	2D	60	-	Dash, hyphen, minus sign
0	046	056	2E	4B	.	Period, dot
1	047	057	2F	61	/	Slash

Numeric characters:

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
0	048	060	30	F0	0	Zero
1	049	061	31	F1	1	One
1	050	062	32	F2	2	Two
0	051	063	33	F3	3	Three
1	052	064	34	F4	4	Four
0	053	065	35	F5	5	Five
0	054	066	36	F6	6	Six
1	055	067	37	F7	7	Seven
1	056	070	38	F8	8	Eight
0	057	071	39	F9	9	Nine

More punctuation characters:

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
0	058	072	3A	7A	:	Colon
1	059	073	3B	5E	;	Semicolon
0	060	074	3C	4C	<	Left angle bracket
1	061	075	3D	7E	=	Equal sign
1	062	076	3E	6E	>	Right angle bracket
0	063	077	3F	6F	?	Question mark
1	064	100	40	7C	@	"At" sign

Upper-case alphabetic characters (letters):

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
0	065	101	41	C1	A	
0	066	102	42	C2	B	
1	067	103	43	C3	C	
0	068	104	44	C4	D	
1	069	105	45	C5	E	
1	070	106	46	C6	F	
0	071	107	47	C7	G	
0	072	110	48	C8	H	
1	073	111	49	C9	I	
1	074	112	4A	D1	J	
0	075	113	4B	D2	K	
1	076	114	4C	D3	L	
0	077	115	4D	D4	M	
0	078	116	4E	D5	N	
1	079	117	4F	D6	O	
0	080	120	50	D7	P	
1	081	121	51	D8	Q	
1	082	122	52	D9	R	
0	083	123	53	E2	S	
1	084	124	54	E3	T	
0	085	125	55	E4	U	
0	086	126	56	E5	V	
1	087	127	57	E6	W	
1	088	130	58	E7	X	
0	089	131	59	E8	Y	
0	090	132	5A	E9	Z	

More punctuation characters:

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
1	091	133	5B	AD	[	Left square bracket
0	092	134	5C	E0	\	Backslash
1	093	135	5D	BD	]	Right square bracket
1	094	136	5E	5F	^	Circumflex, up arrow
0	095	137	5F	6D		Underscore, left arrow
0	096	140	60	79	`	Accent grave

Lower-case alphabetic characters (letters):

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
1	097	141	61	81	a	
1	098	142	62	82	b	
0	099	143	63	83	c	
1	100	144	64	84	d	
0	101	145	65	85	e	
0	102	146	66	86	f	
1	103	147	67	87	g	
1	104	150	68	88	h	
0	105	151	69	89	i	
0	106	152	6A	91	j	
1	107	153	6B	92	k	
0	108	154	6C	93	l	
1	109	155	6D	94	m	
1	110	156	6E	95	n	
0	111	157	6F	96	o	
1	112	160	70	97	p	
0	113	161	71	98	q	
0	114	162	72	99	r	
1	115	163	73	A2	s	
0	116	164	74	A3	t	
1	117	165	75	A4	u	
1	118	166	76	A5	v	
0	119	167	77	A6	w	
0	120	170	78	A7	x	
1	121	171	79	A8	y	
1	122	172	7A	A9	z	

More punctuation characters:

	.....ASCII....			EBCDIC		
Bit	Dec	Oct	Hex	Hex	Char	Remarks
0	123	173	7B	C0	{	Left brace (curly bracket)
1	124	174	7C	4F	|	Vertical bar
0	125	175	7D	D0	}	Right brace (curly bracket)
0	126	176	7E	7E	~	Tilde

Finally, one more nonprintable character:

0	127	177	7F	07	DEL	Delete, rubout

INDEX

The Laser1 Printer
Preliminary

24-0100332-0001

Published by LMI 1000 Massachusetts Avenue, Cambridge MA 02138 USA

Table of Contents

1. Introduction

1.1 General Information

The Laser 1 Printer is a high quality printer capable of producing up to 8 pages per minute on varying paper types and sizes, to include labels, colored paper, and overhead transparency film. The Laser 1 Printer uses a high-resolution dot matrix pattern (300 x 300 dots per inch). It occupies about the same amount of desk space as a typewriter. Some of its features include:

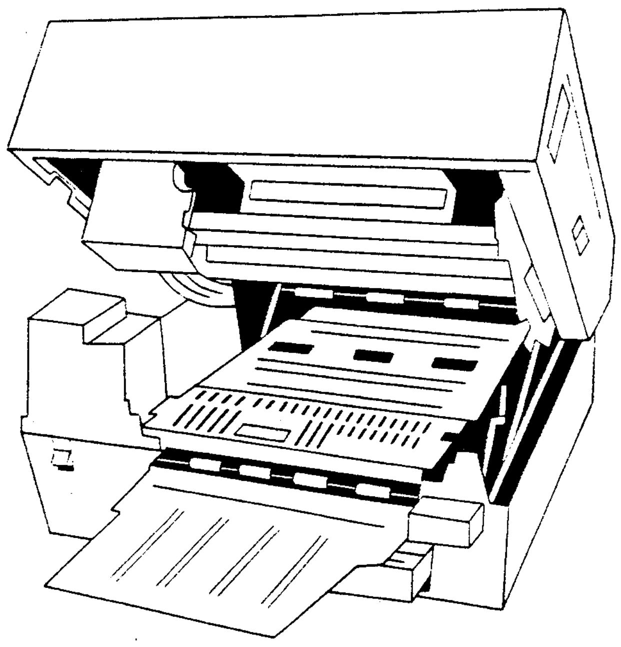
Virtually noiseless operation

Simple operator maintenance

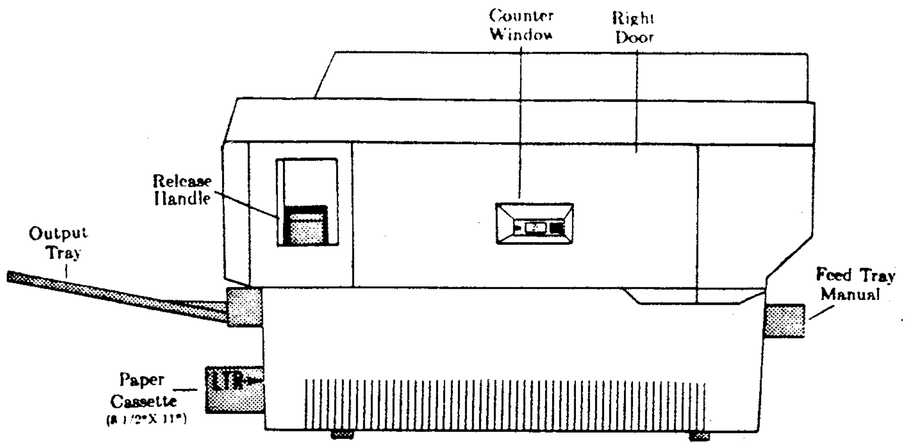
A disposable electrophotographic cartridge

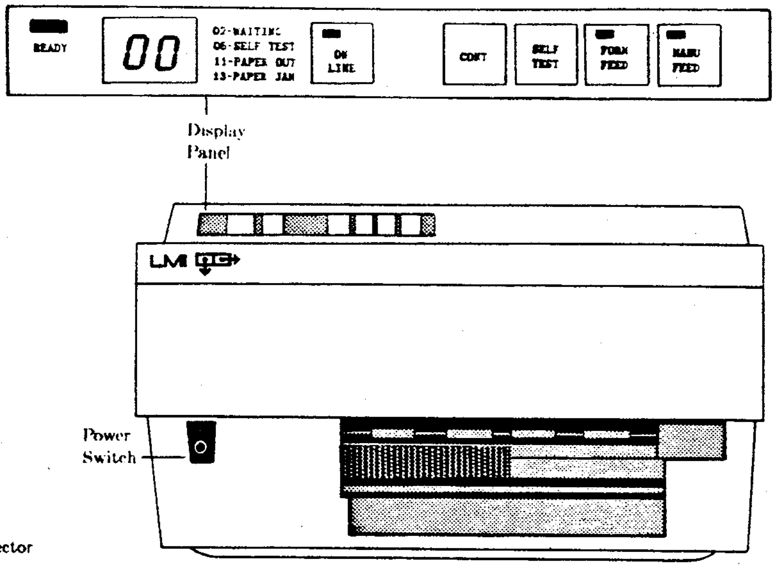
The Laser 1 Printer is shown in Figure 1 and the Printer Front and Right Side with Main Body open is shown in Figure 2.

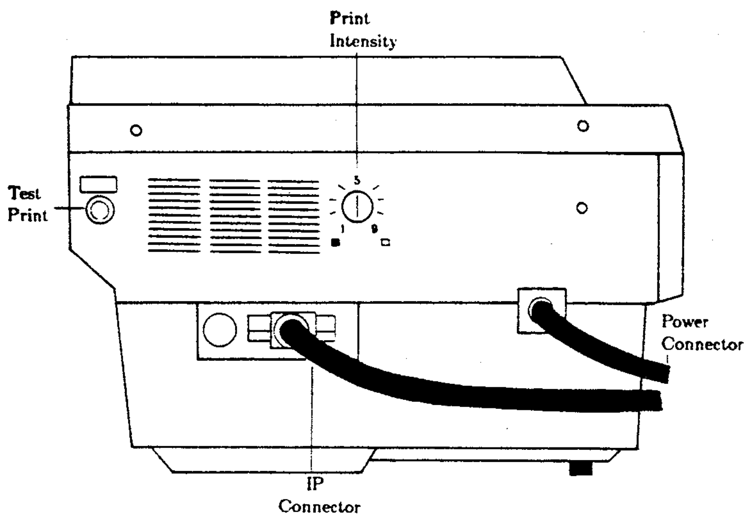
Front View with Upper Main Body Open



Printer — Right Side







Front View with display panel detailed

Back View

Figure 1. Overview of the Laser 1 Printer

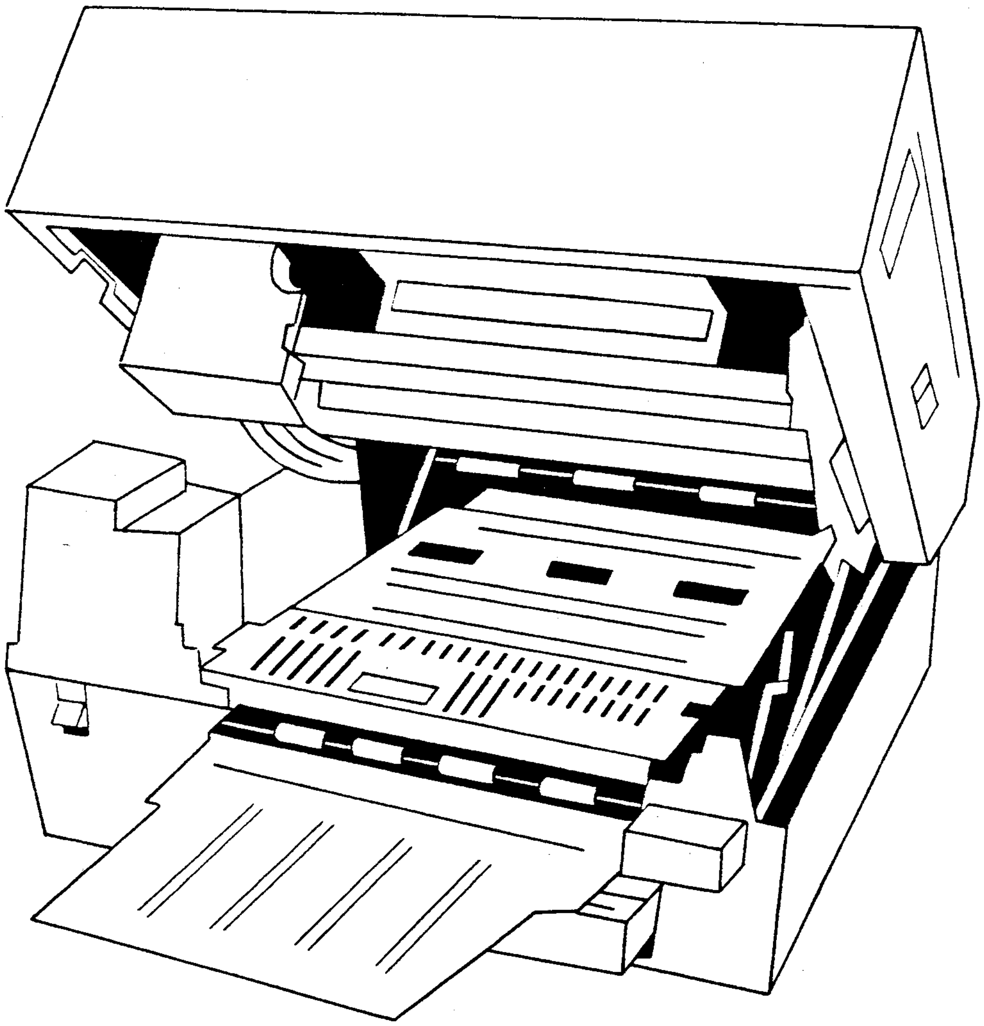


Figure 2. Printer Front and Right Side with Main Body Open

2. Printer Familiarization

2.1 Site Selection

While deciding where to locate the printer, factors to consider are power outlet locations, ambient temperature, traffic, ventilation and any other factors that can affect its operation.

2.2 Printer Illustrations

The illustrations following this section point out the important locations on your Laser 1 printer. Reviewing the illustrations will familiarize you with the different components of the printer. Figure 3 illustrates the printer front.

2.3 Printer Front

The printer front consists of the following:

1. Operator Control Panel
Controls printer operations.

Printer ON and OFF switch
Controls printer power.

2.4 Printer Right Side

Figure 4 illustrates components on the right side of the printer.

2. Right Door
Open to access EP cartridge.

3. Output Tray.
Collects printed paper.

4. Paper Input Cassette Tray
Load paper here.

5. Upper Unit Release Handle
Lever to pull up to open upper main body of the printer.

6. Font Cartridge Slot
Insert Font Cartridges here.

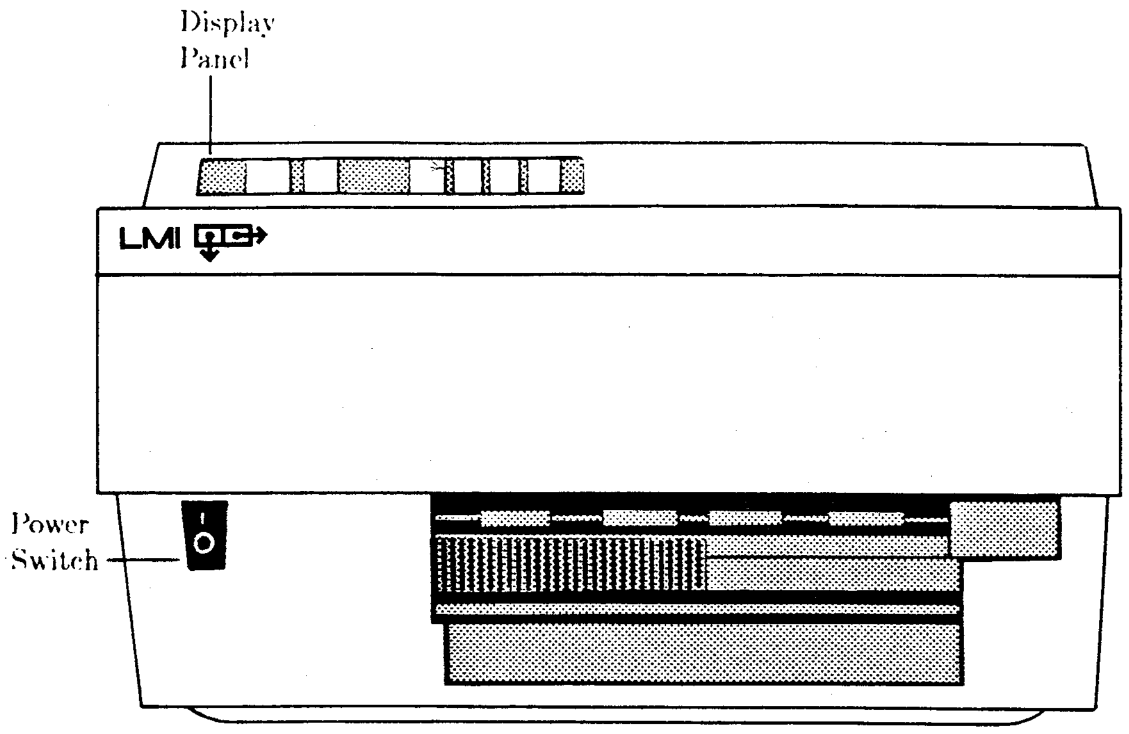


Figure 3. Printer Front

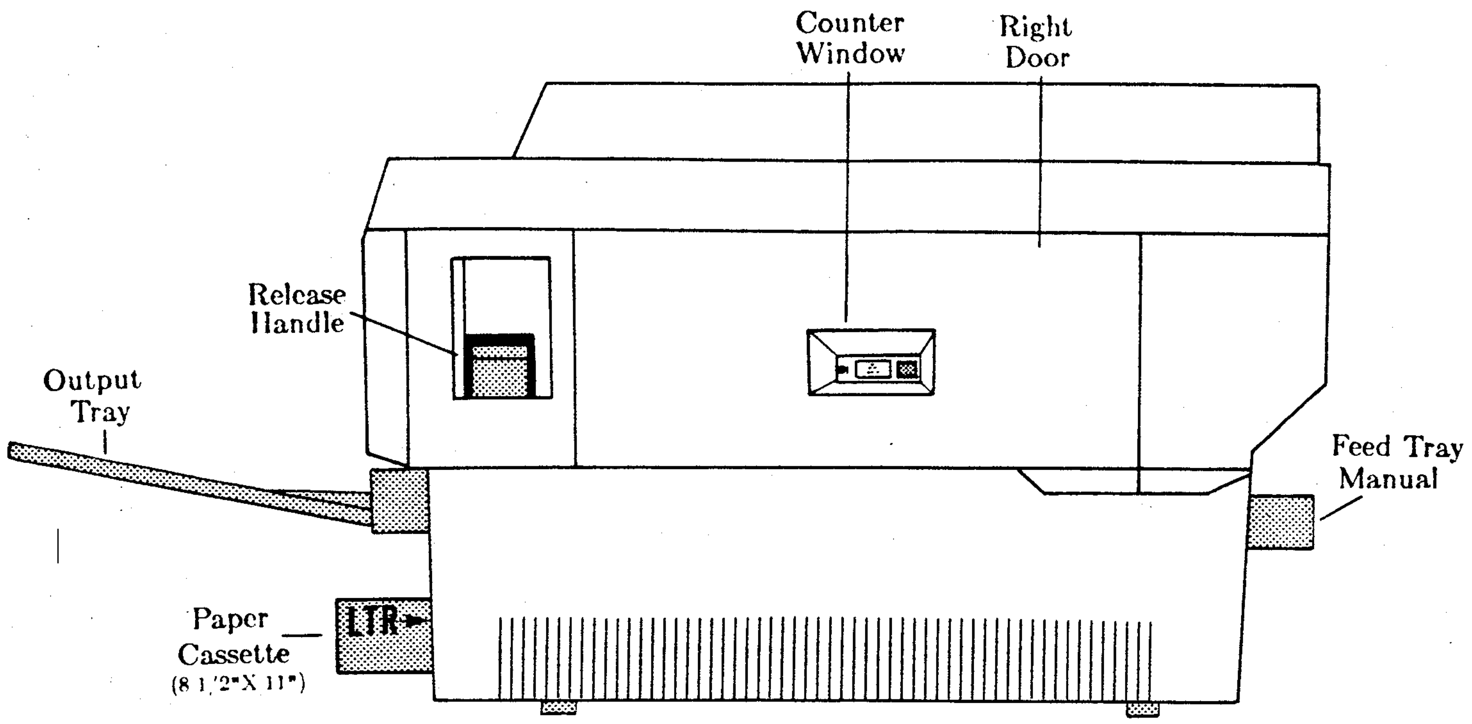


Figure 4. Printer Right Side

2.5 Printer Rear

Figure 5 illustrates the rear of the printer.

The printer rear has the following:

1. **AC Power Cord**
2. **Interface Connector**
   Plug interface cable here.
3. **Upper Main Body**
   Raise to access internal areas of the body.
4. **Print Density Dial**
   Adjusts lightness and darkness of print.
5. **Test Print Button**
   Press for printer self-test.
6. **Rear Door**
   Open to clear paper jams in the cassette feed area.
7. **EP Cartridge slot**
   The cartridge contains toner (dry ink), print drum, and primary corona.
8. **Fusing Assembly Cover**
   Lift to replace fusing roller cleaner pad, clean the fusing rollers, and remove jammed paper.

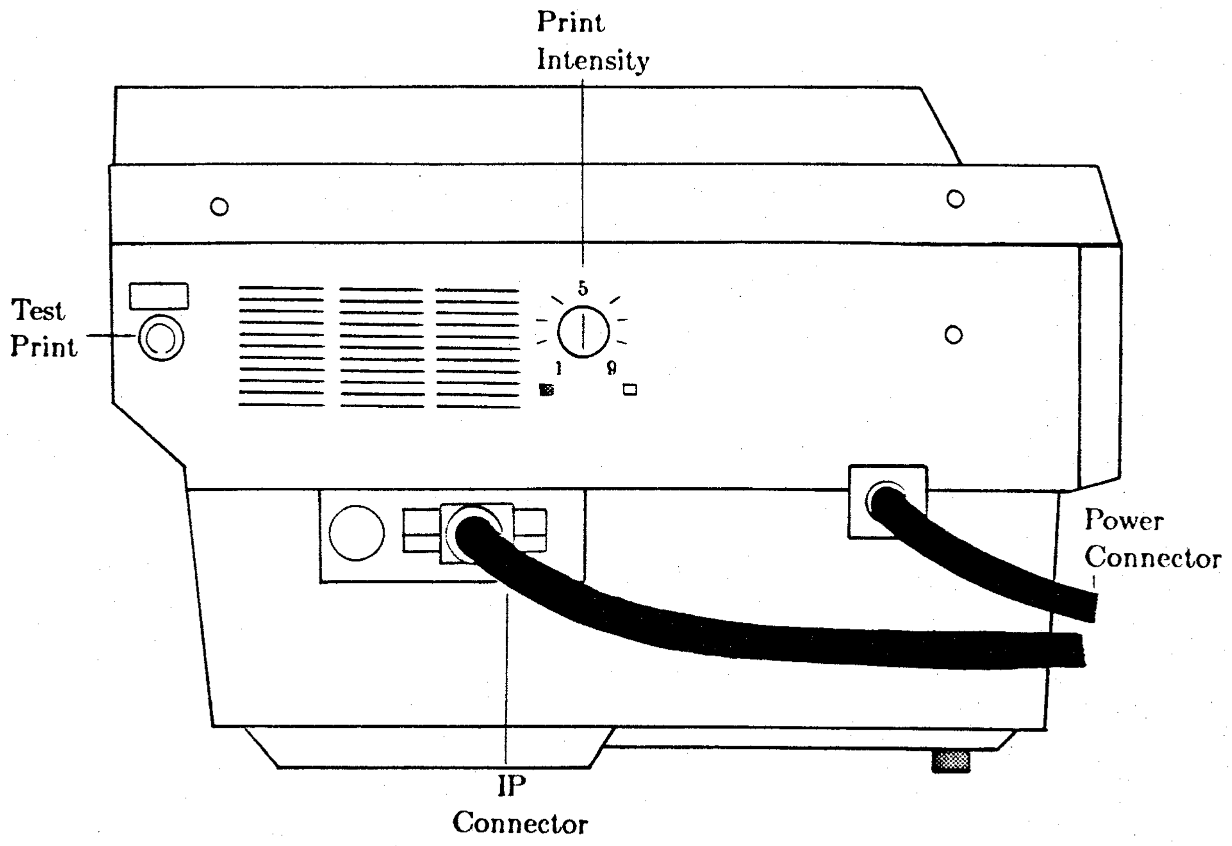


Figure 5. Printer Rear

3. Printer Operation

3.1 Loading paper into the Printer

To load paper into the printer follow these steps as shown in Figure 6.

Fan the stack of paper.

Place it in the paper cassette.

Push the stack under the retainer clips.

Insert the paper cassette into the printer noting the direction that the cassette should be fed into the machine.

3.2 The Operator Control Panel

The Laser 1 Printer has several status display and function keys on the Operator Control Panel. This panel is located on the top left front portion of the printer, and is used to control printer functions such as selecting *manual feed mode*, choosing *on-line* or *off-line*, and running *self test*. The panel *status display* provides status and error code information, keeping you informed as to the present state of the printer. These indicators are explained in the following sections and Figure 7 provides an illustration of this panel.

3.2.1 The Ready Indicator

The *green* ready indicator in the upper left hand corner of the control panel has three possible states:

On, to indicate a ready state.

Off, to indicate that the printer is in a Non Ready state.

Flashing, to indicate that the printer is warming up. During this warm up period, the *status display* reads 02.

3.2.2 The Status Display

The Status Display is a two-digit display, located on the left side of the operator control panel. This display provides printer status and printer error information. When the display number is

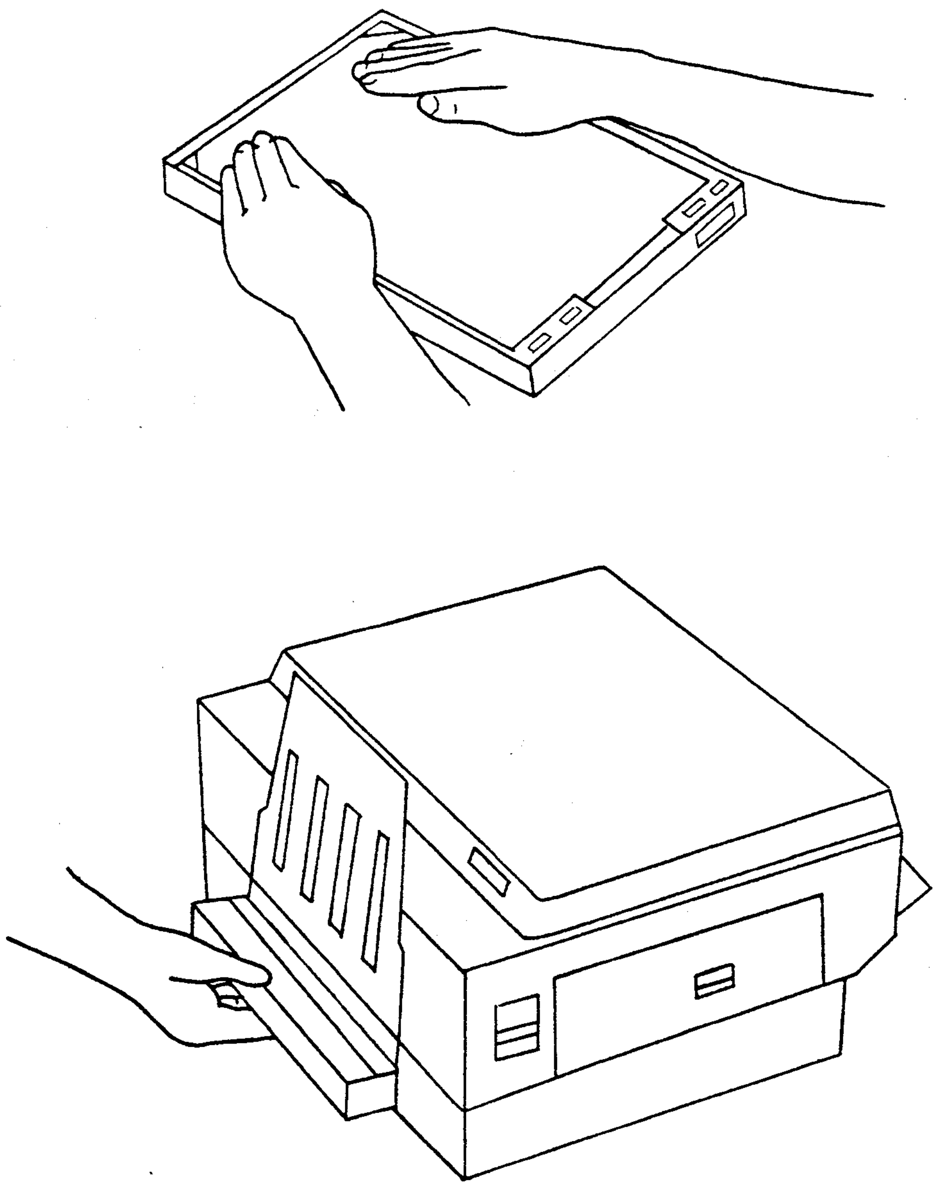


Figure 6. Loading the Paper Cassette

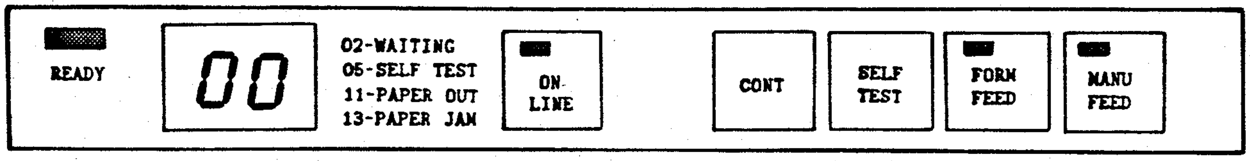


Figure 7. Operator Control Panel

not flashing, status information is being displayed; when the display number is flashing, it indicates a request or error condition.

Printer Status Information

Number	Indication
00	Printer Ready
02	Wait
05	Self-test (non-printing)
06	Printing test print (staggered characters)
15	Test Print (striped pattern)

Operator Serviceable Conditions

Number	Indication
11*	No paper in cassette (add paper)
12*	Printer engine power off (upper main body of the printer)
13*	Paper jam
14*	No EP cartridge installed
PC*	Request for different size paper cassette
PF*	Manual paper feed request (printer ready for manual feed
FC*	Check font cartridge
FE	Font cartridge removed

Error Conditions

Number	Indication
20*	Memory overflow
21*	Print overrun error
22	Receiving buffer overflow
40*	Line error
41*	Print check error
50*	Fusing assembly malfunction
51*	Beam detect malfunction
52*	Scanner malfunction
53*	Laser temperature control circuit malfunction

54*	Main motor malfunction
55*	Printer command error
60	Bus Error
61	Program ROM checksum error
62	Internal font ROM checksum error
63	D-RAM error
64	Scan buffer error
65	D-RAM controller error

Error conditions with an asterisk (*) are cleared by the CONTINUE switch.

When an error condition exists, look up the number in the above table and try to correct the problem. If the problem persists, look in the *Correcting Self-Test Errors* portion of the Maintenance and Troubleshooting section in Chapter 8.

3.3 The On-Line Key

The On-line key allows you to switch between computer control (on-line) and operator control (off-line). The *orange* indicator light on this key is on when the printer is on-line and off when the printer is off-line. When the printer is on-line, you can receive data from the Lambda. NOTE: The printer will power-up on-line.

3.4 The Continue (CONT) Key

The CONTINUE key allows you to resume printing when the printer is off-line (due to an error condition). Pressing the CONTINUE key will clear most errors and place the printer back on-line.

During a paper jam situation, the printer defaults to off-line. After clearing the paper jam, press the CONTINUE key; the printer then reprints the page on which the jam occurred and resumes printing the document.

All of the asterisk * error numbers and conditions in the *status display* table are cleared by the CONTINUE key. If an error is not cleared by the CONTINUE key, see the Self-Test portion of Section V.

3.5 The Self Test Key

Pressing the SELF TEST key causes the printer to perform internal testing of the controller and print a test printout. (If you hold down the key for more than three seconds, the printer will continue self-testing until you press the *Self Test* or *On-Line* key). The *Self Test* key is operational only when the printer is off-line.

During self-test, the number **05** is displayed on the status display. The indicator lights on the Operator Control Panel will light, indicating that Self Test is in operation. When the printing portion of the self-test begins, the number **06** is displayed. When the self-test is complete, the status number should change to **00** to indicate that the printer is ready. If any errors occur during self-test, an error number will be displayed to help you locate the problem. See the Self- Test section in Chapter 8.

3.6 The Form Feed Key

The *form feed* key is used to empty the print buffer one page at a time while the printer is off-line and in a Ready state. Whenever there is data in the buffer, the indicator on the Form Feed key lights. If the buffer contains less than a page of data, pressing the Form Feed key will print that portion of the page contained in the buffer. When Form Feed is active, the indicator light on the Form Feed key flashes.

If several pages are buffered, pressing the Form Feed several times clears the data buffer. This feature is useful, for example, when you have been printing and then go off-line. Since the data buffer can contain several pages of data, you can press the Form Feed key before turning off the printer. When the print buffer is empty, the indicator will go out, indicating that you can turn the printer off without losing data.

NOTE: If the printer is taken off-line during a data transfer, a partial page may be stored in the print buffer. In this case, if the Form Feed key is pressed to clear the print buffer, a partial page may be printed. When the printer returns on-line, the rest of the page will be printed; no data will be lost, but the page will not be formatted as intended. To avoid this problem, do not clear the print buffer (by pressing Form Feed) unless you know that the entire contents of your file have been transferred from the Lambda to the printer.

When a paper jam or other error occurs during a form feed mode, the form feed is cancelled.

When the printer is in the manual feed mode and the form feed key is pressed, paper must be manually fed into the printer.

3.7 Manual Feed

Pressing the *manual feed* key causes the printer to alternate between manual feed and cassette feed. The printer must be off-line in order to access the manual feed mode from the Operator Control Panel. When in the manual feed mode (and on-line), the indicator light on the Manual Feed key is lit, and the status display alternately flashes PF and the selected paper size.

NOTE: If no data is waiting to be transferred to the printer when the manual feed key is pressed, the status display will read **00**.

See the *Using the Manual Feed Feature* discussion in Chapter 6.

3.8 What to do before you Print

In order to print data from your Lambda system, the printer must:

- Be in the READY state (READY indicator on and status **00** displayed),
- Have the ON-LINE indicator light ON, and
- Have the interface cable properly connected from the Lambda to the connector on the left side of the printer.

If the printer is not in a READY state, check the status display to find the reason. Once the printer is ready, press the on-line key and the printer is prepared to print your document.

4. Adjusting the Print Density Dial

The *print density* dial is located on the left side of the printer. This dial is used to adjust the intensity of the printed output and has a range of 1 through 9. Turning the dial counterclockwise increases print density (makes it darker) and turning the dial clockwise decreases the print density (makes it lighter). Under normal conditions, the dial should be in the centered position (number 5). Increasing the density causes the printer to use more toner, which will shorten the life of the EP cartridge.

5. Using Font Cartridges

5.1 Useful Features

One of the useful features of the Laser 1 printer, to be provided in future releases, is its ability to accept optional font cartridges containing a variety of character styles. You will be able to select these different character styles simply by inserting a different font cartridge in the printer and varying the fonts used in a file. More information on its features will be available when it is fully supported.

5.1.1 Inserting Cartridges

To insert a font cartridge, set the printer off-line and slide the cartridge into the slot in the upper right corner of the printer. Be sure that the cartridge is firmly seated in the slot before returning the printer on-line. Return to on-line by pressing the on-line button.

5.1.2 Removing Cartridges

To remove the font cartridge, set the printer off-line, grasp the edges of the cartridge, and slowly pull it out of the slot. Press the On-line key to return to on-line after removing the cartridge.

Optional font cartridges are available from LMI.

6. Using the Manual Feed Feature

6.1 Printing Standard-Sized Paper

The manual feed feature allows paper to be fed into the printer by hand, rather than by automatically selecting paper from the paper cassette. To manually feed standard-sized pages (8 1/2 x 11, A4, B5, or legal), follow the steps listed here.

1. Set the printer off-line by pressing the on-line key until the indicator on the key is not illuminated.
2. Press the Manual Feed key on the operator control panel. The indicator on the Manual Feed key will light up, signifying that the printer is in the Manual Feed mode. The status display will alternately display PF and paper size to be fed into the slot. (L=letter, LL=legal, A4=A4, b5=B5).

   NOTE: The paper size that the printer expects is determined by the paper input cassette that is installed. If you manually feed a page that is not the size indicated on the status display, some of your data may be lost (depending on how the page is formatted).
3. Press the on-line key until the indicator on the key lights.
4. Slide paper into the manual feed slot so that the paper rides against the right edge of the opening (along the paper feed guide) and the top of the paper (or letterhead) enters the printer first (face up).
5. To feed more papers into the printer, wait a a couple of seconds after the previous page is completely drawn into the printer. Then feed another page exactly as you fed the first.
6. To exit the manual feed mode, press off-line and then press the Manual Feed key. The indicator on the key will not be lit, which indicates that you have exited the Manual Feed Mode.

Print Modes. NOTE: Odd-sized paper must be formatted as if standard paper is loaded in the printer. Otherwise, the printer does not have a reference point from which to print.

6.2 Two-Sided Printing

Using the manual feed mode, you can print on both sides of a piece of paper. To do this, follow these steps:

- With the printer off-line, enter the manual feed mode by pressing the Manual Feed key on the Operator Control Panel. The indicator on the Manual Feed key will light up.
- Press the on-line key until the indicator on the key lights up.
- Feed the first side of the paper into the printer so that the top of the page enters the printer first and the paper's right edge is against the right side of the manual feed slot. The paper will feed into the printer and the first side will be printed.
- Take the printed page from the output paper tray, flip it over so that the printed side is facing down, and manually feed the paper into the printer. If the paper is curled after printing the first side, straighten it out before feeding it in to print the second side.

7. Operation from Lisp: Commands

7.1 General Information

This is the first release version of the LISP software designed to provide printouts of LISP source files. Once installed, it is used as part of the existing hardcopy facility, which also drives the support software for our dot matrix printers. This allows the Laser Printer to become the Lambda's default printer, permitting the user to employ familiar commands to use the Laser Printer. The Laser Printer can also be accessed from other machines on the network. All spooling, however, will be performed on the Lambda that has the Laser Printer physically connected to its serial port (port B).

7.2 Printing from ZMACS

The Laser Printer is accessed from ZMACS using standard printer commands such as the command to print a resident buffer:

```
META-X print buffer
```

and the command to print a file:

```
META-X print file
```

Since the default printer is the Laser Printer, these commands will send the specified file or buffer to the Laser Printer. (See the ZMACS Manual for further details on these commands). Please note that fonts are **not** supported in this version of the software.

7.3 Printing from a LISP Listener

Once more, standard printing functions are employed to send *files* or *streams* to the Laser Printer. The most commonly used of these appear below:

HARDCOPY-FILE *filename* &OPTIONAL &KEY *orientation* Function

This causes *filename* to be printed on the Laser Printer.

HARDCOPY-STREAM *stream* Function

This function prints from stream until input is exhausted.

7.4 Screen Dumps

There are only 59K bytes of memory currently available on the Laser1 for manipulating raster graphics. Despite this disadvantage, screen dumps via the ⟨TERMINAL⟩ Q commands are supported as follows:

⟨TERMINAL⟩ Q

Typing ⟨TERMINAL⟩ Q at the LISP listener will print out an entire screen dump. The lack of memory in the printer, however, results in the screen being split and printed out on two pages.

⟨TERMINAL⟩ 1Q

⟨TERMINAL⟩ 1Q prints out a dump of the current window. The size of the window will determine whether or not the output is split onto two sheets or not.

⟨TERMINAL⟩ 3Q

This command was designed for use with the Laser 1 printer. By optimizing on the amount of white space on the screen, it will print out an entire screen dump on a single sheet when possible. "Average" LISP Listener screen images will not be split. However, the following should be noted:

1. No border line will be printed.
2. All screen dumps attempted in reverse video will be split onto two pages.
3. All ZMACS windows will be split onto two pages.

7.5 Relevant Variables

Commonly used variables are detailed below:

SI:*DEFAULT-PRINTER* printer machine Variable

This symbol stores the location and type of printer that this machine should use for normal text printouts.

On machines that access the Laser Printer, the variable should be set as follows:

```
(SETQ SI:*DEFAULT-PRINTER* '(:LASER1 "MACHINE"))
```

where *machine* is the name of the Lambda with the printer physically attached to its serial port. The Lambda with the Laser Printer attached to its serial port **must** have this variable set as described.

To properly print bit maps:

```
(SETQ si:*default-bit-array-printer* '(:laser1 "lama"))
```

8. Maintenance

8.1 Introduction

This section explains the preventive maintenance procedures that are necessary to keep your printer running at peak performance. Refer to Chapter 9, "Troubleshooting," if your printer has problems not described here.

8.2 Periodic Maintenance

In order to keep your printer running smoothly and printing high- quality documents, you should observe the following maintenance procedures:

- Replacing the EP cartridge
- Cleaning the corona wires
- Replacing the separation belt
- Maintaining general cleanliness in the machine area

8.2.1 Replacing the EP Cartridge

The EP catridge (shown in Figure 8) periodically needs to be replaced. Determine if the Cartridge needs replacement by looking at the color of the indicator, visible through the window in the right door. As the drum (inside the cartridge) rotates, the color of the indicator changes to indicate the usable life of the cartridge.

Each cartridge contains enough toner (dry ink) to make about 3,000 letter-size or A4 prints. If many pages with a high ratio of black to white are printed, the toner is used more quickly. White stripes may appear on your pages before the printing capacity indicator turns red. If this happens, replace the EP cartridge.

The life of the EP cartridge can be slightly extended by removing the cartridge and gently rocking it back and forth to spread the toner evenly. Insert the cartridge into the printer and run 3 or 4 test prints to check the image before printing a document.

When the EP cartridge needs replacing, follow these steps:

- With the upper main body of the printer open, open the printer's right door, pull out the used EP cartridge and discard it.
- Holding the cartridge by the handle, turn it 45 degrees in both directions (about 5 times).
- After gently shaking the cartridge, insert it into the printer.
- Flex the black tab on the EP cartridge and pull out the tab completely to remove the attached sealing tape.
- Close the printer's right door.

8.2.2 Changing the Fusing Roller Cleaner Pad

The location of the Roller Cleaner Pad is shown in Figure 9. To change the Cleaner pad follow these steps:

- Raise the *fusing assembly* and slide the green-handled fusing roller cleaner pad to the right. Slide a new cleaner pad into the slot and lower the fusing assembly cover. NOTE: A new fusing roller cleaning pad comes in the same box as the EP cartridge.
- CAUTION: The area around the fusing assembly gets hot during printing, so be cautious when installing a new cleaner pad.

NOTE: If the fusing rollers (located under the green fusing assembly cover) are dirty, turn the power switch OFF and, after the printer has cooled down, clean the rollers with a damp cloth.

8.3 General Cleaning

The printer should be kept in general state of cleanliness. Any visible toner should be wiped away with a damp cloth and then re-wiped with a dry cloth.

There are three areas that need regular cleaning to keep the Laser 1 Printer functioning at peak performance:

The primary corona wire
The transfer corona wire
The transfer guide

Cleaning maintenance only requires a few minutes. The steps to follow for cleaning each of the items mentioned above is described below.

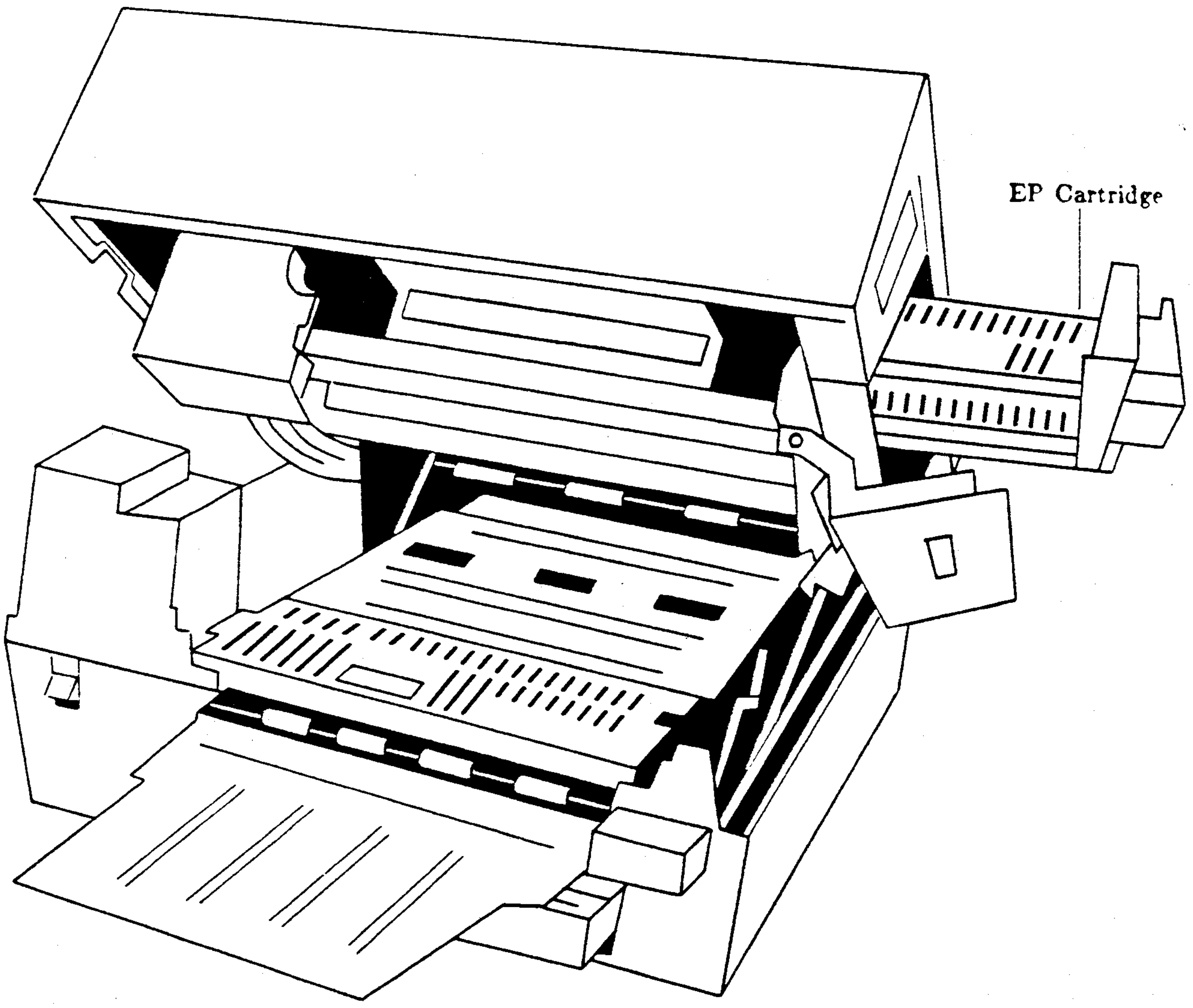


Figure 8. Inserting the EP Cartridge into the Printer

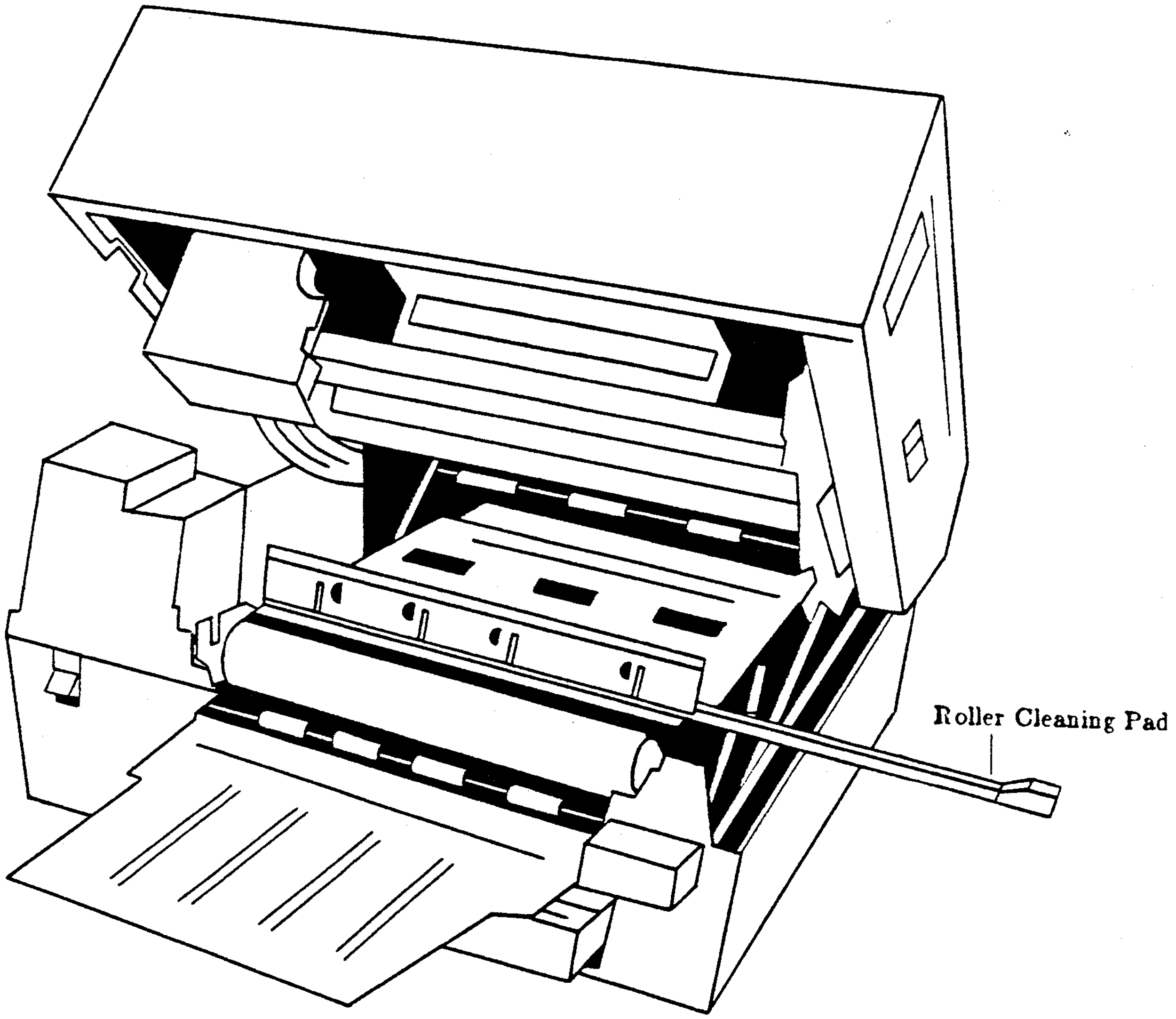


Figure 9. Changing the Roller Cleaning Pad

8.3.1 Cleaning the Primary Corona Wire

To clean the Primary Corona Wire, follow these steps:

Open the printer by pressing the upper unit release lever and lifting the upper main body of the printer.

Pull out the EP cartridge.

Insert the green-handled wire cleaner into the long slot of the EP cartridge (beside the shutter) and move it back and forth several times in the slot. (The cleaner displaces the thin protective plastic sheet.) This step cleans the *primary corona wire* within the EP cartridge.

After cleaning the primary wire, re-insert the EP cartridge into the printer.

8.3.2 Cleaning the Transfer Corona Wire

To clean the transfer corona wire, follow these steps:

With the upper main body of the printer open, dip a cotton swab in alcohol and gently wipe the *transfer corona wire*.

Repeat this process until the swab no longer has any residue on it.

8.3.3 Cleaning the Transfer Guide

To clean the transfer guide follow these steps:

With the printer open, dampen a cloth (it should be moist, not wet) and wipe the transfer guide clean.

Close the printer.

9. Troubleshooting the Laser 1 Printer

9.1 Printer Problems

Printer problems range from forgetting to plug in the power cable to serious electronic or mechanical problems. For the latter, you will need to call the LMI Customer Assistance Line. This number is designed to help you to determine if the problem is one that can be easily corrected by you or if a service call is required. The following information gives some problems, their symptoms, and simple solutions.

9.2 Potential Problems

- Is the AC power cord attached properly and plugged into an outlet of the correct voltage (voltage is noted on the label to the left of the printer's AC power cord receptacle).
- Is the power switch in the ON position?
- Are the interface cable connectors seated properly on both the printer and the Lambda end?
- Is the printer on-line (is the on-line indicator lit?)
- Are there any error numbers displayed on the status display? If so, consult the Self Test and Correcting Self Test errors section in this chapter and follow the corrective procedure there.

If the above checklist still does not solve the problem, then the problem can be of a specific type such as a paper jam, used-up EP cartridges, dirty primary and transfer coronas, or a broken separator belt. Most of these problems can be solved by referring to the sections for these that describe how to maintain your printer.

The printer's self test feature enables you to diagnose the majority of problems that occur. Read the "Self Test" section in this chapter for more information.

Many problems can be solved by simply thinking the problem through. Following procedures listed in this *Manual* can save you time and prevent a customer assistance call.

9.3 Self Test

For printer and controller self-tests, follow these steps:

- With the printer off-line and in a **READY** state, press the **TEST PRINT** button located on the left side of the printer. After a few seconds of delay, a striped printout will be printed for each time you press the **test print** button. Check the printout to see that all the lines are uniform and clear, and that there are not light spots, smudges or other irregularities. If the printout is not satisfactory, check the "Solving Print Quality Problems" discussed in this chapter for possible causes.
- Press the **SELF-TEST** key on the Operator Control Panel. After a few seconds of delay, the printer will print a staggered character font pattern. Check to see that all of the characters are clear and well-formed, and that there are no noticeable light spots, blotches or streaks on the paper. If the printout has any of the above mentioned problems, repeat the self-test to see if these problems still exist. If the problem still exists, check the section on "Correcting Self Test Errors" in this chapter for possible cause.
- Check the status display on the Operator Control Panel to see if any error numbers are displayed. If an error number is displayed, check the "Correcting Self Test Errors" section in this chapter for information.

9.4 Correcting Self-Test Errors

If an error number flashes on the status display when running self-test, (or even when printing), consult the table below to find out what you can do to correct the error. If you cannot correct the error, contact LMI customer assistance.

11 A flashing number **11** means that either the paper cassette is empty or that the cassette is not loaded into the printer. After resolving either of the above problems, press the CONTINUE switch on the Operator Control Panel. NOTE: If the wrong size paper cassette is being used, "**PC**" will flash on the status display when you attempt to print.

12 A flashing number **12** indicates that the upper main body of the printer is not closed properly. Close the printer until it is firmly latched. Press the **CONTINUE** key.

13 A flashing number **13** on the status display indicates that a paper jam has occurred. Open the upper main body of the printer and check for jammed paper. Remove the jammed paper, close the printer and press the **CONTINUE** key. Data on the jammed page will be reprinted automatically. NOTE: The printer must be opened to clear the paper jam error. For more information see the section on "Paper Jams" in this chapter.

14 A flashing number **14** indicates that the EP cartridge is either not installed or not correctly installed in the printer. To correct this error, insert the EP cartridge or check to be sure that it is properly installed. Press the **CONTINUE** key to resume

printing.

15 A flashing number **15** indicates that the Laser 1 printer is in self-test mode. When self-test is complete, printing can be continued by pressing the `ON LINE` or `CONTINUE` key on the Operator Control Panel.

PC When this message flashes, the status display alternates between PC and a paper size number requested by the system (L=letter size, LL=legal, A4=A4, or b5=B5). To correct this problem, insert the requested cassette and press the `CONTINUE` key; if the requested paper cassette is not inserted, pressing the `CONTINUE` key will cause the printer to ignore the paper cassette request. NOTE: If the requested paper size does not match the installed cassette, the `CONTINUE` key will allow you to print on the loaded paper. However, the output will be formatted for the requested paper size and a possibility exists for a clipped image.

PF When this message flashes, a manual paper feed has been requested. This status display will alternately flash **PF** and a paper size. Insert a sheet of the requested paper size into the printer's manual feed slot.

NOTE: Inserting a paper size other than that requested on the status display may cause clipping of your image.

To exit the manual feed mode, press the `ON LINE` key to put the printer off-line and then press the **`MANUAL FEED`** key. Begin printing from the paper cassette by pressing the CONTINUE key.

FE A flashing **FE** on the status display indicates that the font cartridge was removed when the printer was on-line or when form feed was being used (or during self-test). To clear the error condition, re-insert the font cartridge into the printer, turn the power switch to the OFF position to reset the printer, and then switch the power switch to the ON position.

FC A flashing **FC** indicates that a font cartridge was removed or replaced during the formatting of a page. Re-insert the removed cartridge and depress the `CONTINUE` switch. The current page will be formatted and then printed as intended.

20 A flashing number **20** indicates a memory overflow error to indicate that there is more data being received from the Lambda than will fit on one page. To continue printing, press the `CONTINUE` key. Only the amount of data that fits on your page will be printed.

21 The flashing number **21** indicates that the page formatting process is not fast enough for the printer. Press the `CONTINUE` key to continue printing. NOTE: There may be some data loss on the page that was being formatted when the error occurred.

22 This error indicates that the printer's receiving buffer has overflowed during a busy state. Pressing the `CONTINUE` key resumes printing, but results in loss of data.

40 This error indicates that a data error (parity, framing, or line overrun) has occurred during the reception of data from the Lambda. The error may also indicate an incorrect configuration. To continue printing, press the **CONTINUE** key.

NOTE: This error may also occur if you power-up the Lambda while the printer is on-line. If this happens, press the **CONTINUE** key to clear the error. If this error occurs repeatedly, check the interface connector to insure that it is connected firmly to the printer and Lambda. If the error still occurs after ensuring that the interface connector is snugly seated and the configuration is correct, call LMI Customer Assistance.

41 This error indicates that a temporary error has occurred on the print page. To correct the error, remove the paper from the output paper tray and press the **CONTINUE** key to resume printing. The page where the error occurred will be reprinted automatically. If this error occurs repeatedly, call LMI.

50 This error number indicates a fusing assembly malfunction. The printer cannot immediately recover from this type of error. Wait approximately 10 minutes and then press the **CONTINUE** key to resume printing. If this error occurs repeatedly, call LMI.

51 A flashing number **51** indicates a beam detect malfunction. Press the **CONTINUE** key to resume printing. If this error occurs repeatedly, call LMI.

52 Error number **52** indicates a scanner malfunction. Press the **CONTINUE** key to resume printing. If this error occurs repeatedly, call LMI.

53 A flashing number **53** indicates a malfunction in the laser temperature control circuitry. The printer cannot immediately recover from this error. Wait approximately 10 minutes and then press the **CONTINUE** key to resume printing. If this is a consistent problem, call LMI.

54 Error number **54** indicates a main motor malfunction. Pressing the **CONTINUE** key clears the error and resumes printing. If this is a recurring problem, call LMI.

55 Error number **55** indicates a printer command error, meaning that commands cannot be exchanged between the print engine and its controller. Press **CONTINUE** to resume printing. If this error occurs repeatedly, call LMI.

60 This error indicates a bus error caused by a circuit malfunction. Switch the power to the OFF position to reset the printer and then press the ON switch. If this is a consistent problem, call LMI.

61 A number **61** flashing on the status display indicates a checksum error on the controller's program ROM. Switch the power switch to the OFF position to reset the printer and then set the power switch back to the ON position. If this is a constant problem, call LMI.

62 Error number **62** indicates an internal font ROM checksum error. Switch the power switch first to the OFF position and then to the ON position to clear the error. If this is a continuing problem, call LMI.

63 Error number **63** indicates a dynamic RAM error (read, write or parity error). Switch the printer power to OFF and then ON. If this is a continuing problem, call LMI.

64 Error number **64** indicates a scan buffer error. Switch the power switch first to the OFF and then to the ON position. If this error occurs repeatedly, call LMI.

65 Error number **65** indicates a D RAM controller error. Switch the power to the OFF position and then switch it back to the ON position. If this error occurs repeatedly, call LMI.

9.5 Solving Print Quality Problems

The majority of print quality problems can be solved by observing the "General Cleaning" procedure described earlier in this chapter. You should also use manufacturer-approved paper, and check to see that the status indicator on the EP cartridge is not red (which indicates that the cartridge needs replacement).

If you are experiencing a problem with print quality, put the printer off-line and press the `TEST PRINT` button a few times to see if the problem persists. If it does, read through the list of possible print problems below. Each situation has specified steps to follow to eliminate the problem. If the problem still exists after following the appropriate steps, call LMI.

9.5.1 Blank Printout

- Has the EP cartridge been inserted correctly in the Printer? If not, re-insert the cartridge and run a self-test.
- Is the indicator on the EP cartridge red? If so, replace the EP cartridge.
- Has the sealing tape been removed from the EP cartridge? If not, remove the tape and try the self-test procedure again. If this fails, call LMI.

9.5.2 Complete Image is Light

- Is the print density dial set with its dot at the top (position number 5)? If not, set the dial to number 5 and repeat the self-test process.
- Does print quality improve when new paper is used? Be sure that the paper that you are using meets the paper specifications for your printer.

9.5.3 Complete Image is Dark

- Is the print density dial set correctly (near the middle setting)? If not, set it correctly and repeat the self-test procedure. If this does not improve the image, call LMI.

9.5.4 Black Image

- Is the primary corona wire broken (the wire located under the vinyl flap of the EP cartridge)? If so, replace the EP cartridge. If replacing the EP cartridge does not correct the problem, call LMI.

9.5.5 Stained Strip Along Right Side of Paper

- Is the separation belt or the area around the belt dirty? If so, clean the dirty area and run the self-test procedure. If this does not eliminate the problem, replace the EP cartridge.

9.5.6 Stains on the back of the Paper

- Is the area around the manual paper feed slot dirty? If so, clean it first with a damp cloth and then wipe with a dry cloth.
- Is there toner on any of the printer rollers? If so, clean them first with a damp cloth and wipe with a dry cloth.
- Is there toner on or around the transfer corona assembly? If so, clean it with a wet cloth and then wipe with a dry cloth.
- Is there toner on the underside of the EP cartridge? If so, clean the cartridge with a damp cloth, and then wipe with a dry cloth.
- Is the fusing roller cleaning pad dirty? If so, replace the fusing roller cleaner pad. See the EP cartridge instructions on how to do this.

9.5.7 Dark Vertical Lines (In Direction of Paper Feed)

- Is the fusing roller cleaning pad dirty? If so, replace the pad as described in the EP cartridge instructions.
- If this does not eliminate the problem, call LMI.

9.5.8 Sharp Horizontal Black Lines (Cross Feed Direction)

- Call your LMI representative.

9.5.9 Foggy Vertical Stripes (Paper Feed Direction)

- Try cleaning the primary corona wire. If this doesn't work, change the EP cartridge.

9.5.10 White Horizontal Lines (On Black Background)

- Does the paper being used meet the specifications? If not, switch to correct paper and call LMI if the problem is not resolved.

9.5.11 Thin Vertical White Lines or Stripes (Paper Feed Direction)

- Is the status indicator on the EP cartridge red? If so, replace the EP cartridge.
- If not, remove the cartridge and gently rock it back and forth to spread the toner evenly within the cartridge. Insert the cartridge into the printer and run self-test to see if the problem is corrected.
- NOTE: Shaking the cartridge may leak toner and stain on some of the documents to follow. Print 5 or 6 test prints until the stain disappears.
- Is the fusing roller cleaning pad dirty? If so, replace the pad as described the EP cartridge procedure described earlier.
- Does the print quality improve after the transfer corona wire is cleaned ? See section 8.3.2, "Cleaning the Transfer Corona Wire".
- If cleaning does not resolve the problem, replace the EP cartridge.

9.5.12 Faulty Registration

- Is the leading edge of the paper curled excessively? If so, straighten the edge of the paper before using it or add new paper.
- Is the correct paper being used? If not, try changing the paper. If this does not solve the problem, call LMI.

9.5.13 Poor Fusing (Toner Smears when Rubbed)

- Is manufacturer-approved paper being used? If not, change to approved paper and try printing again.
- If the problem persists, call LMI.

9.5.14 Image Waviness or Distortion

- Call LMI.

9.5.15 Printer Does Not Power Up (Indicator Not Lit)

- Is the power cord plugged into the AC outlet?
- Is the upper main body of the printer firmly closed?
- If the above corrective action does not work, call LMI.

9.6 Clearing Paper Jams

When paper flows through the printer, it passes through four main areas as shown in Figure 10. Follow these steps to clear a paper jam:

Manual Feed area

Cassette Feed area

Separation/feeder area

Fusing/delivery area

When normal paper flow through the printer is obstructed, a number (13) will flash on the status display. To clear the jammed paper from the machine, follow these steps:

- Press the release lever and open the main body of the printer. NOTE: Some parts of the printer are sensitive to light, so close the upper main body of the printer immediately after removing the paper jam.
- Look for the paper jam in the separation/feeder area. If the jam is here, carefully remove the paper.
- If the paper is not found in the separation/feeder area, open the green fusing assembly cover and check for jammed paper. Remove the paper by **gently** pulling it out of the printer.
- If the paper jam is not found in the separation/feeder area, open the rear door and check for jammed paper. **Gently** pull the paper out of the printer and close the access door.
- Press the `CONTINUE` key on the Operator Control Panel to resume printing. The page that was being printed during the jam will be automatically reprinted.

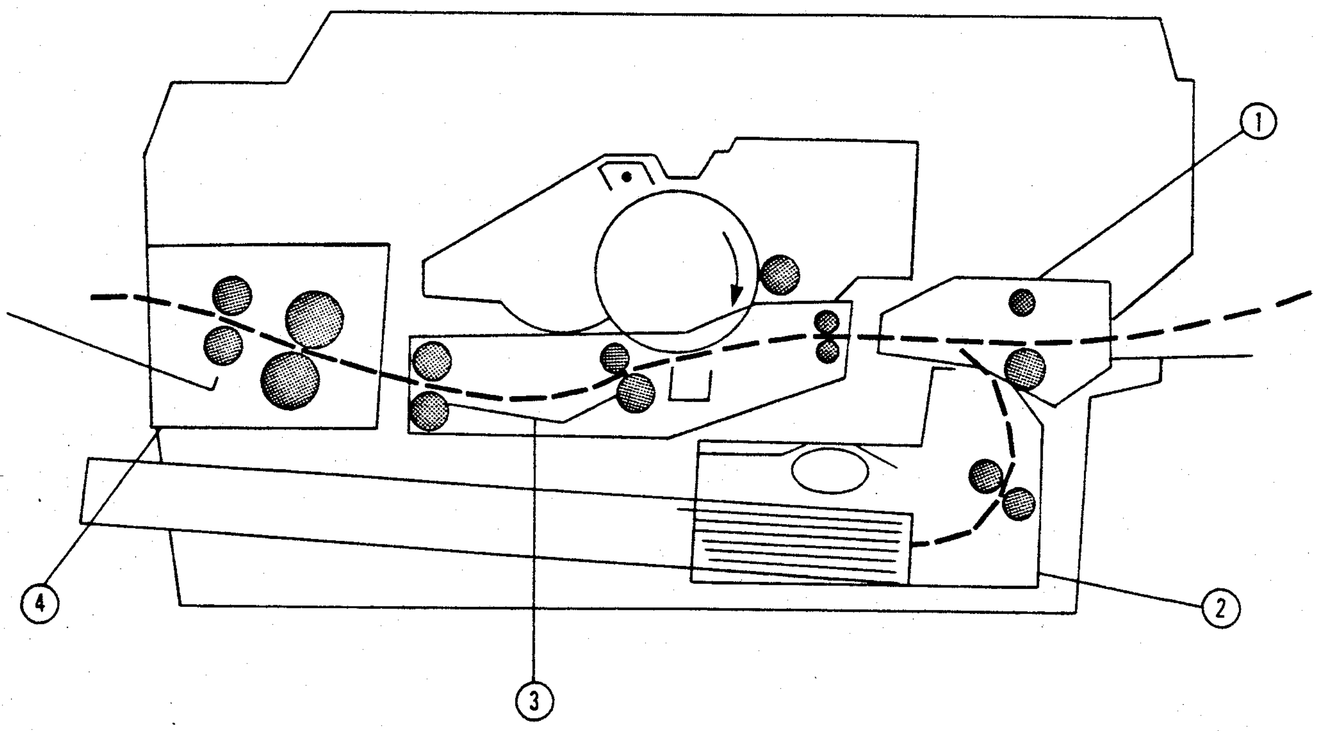


Figure 10. Paper Path